CU-CSSC-90-17

CENTER FOR SPACE STRUCTURES AND CONTROLS

COMPUTATIONAL METHODS AND SOFTWARE SYSTEMS FOR DYNAMICS AND CONTROL OF LARGE SPACE STRUCTURES

by

K. C. Park, C. A. Felippa,
C. Farhat and E. Pramono

July 1990

COLLEGE OF ENGINEERING
UNIVERSITY OF COLORADO
CAMPUS BOX 429
BOULDER, COLORADO 80309

COMPUTATIONAL METHODS AND SOFTWARE SYSTEMS FOR DYNAMICS AND CONTROL OF LARGE SPACE STRUCTURES

K.C. PARK, C.A. FELIPPA, C. FARHAT AND E. PRAMONO

Department of Aerospace Engineering Sciences and
Center for Space Structures and Controls
University of Colorado, Boulder, CO 80309-0429

July 1990

Report No. CU-CSSC-90-17

Final Report on Grant NAG1-756, funded
by NASA Langley Research Center

TABLE OF CONTENTS

TABLE OF CONTENTS (Continue)

ENCLOSED PAPERS (Continued)

SUMMARY

This is a final report on the tasks supported by NASA Langley Research Center under Grant NAG1-756, **Computational Methods and Software Systems for Dynamics and Control of Large Space Structures**. The report covers progress to date, projected developments in the final months of the grant and conclusions. Pertinent reports and papers that have not appeared in scientific journals (or have not yet appeared in final form) are enclosed.

The grant has supported research in two key areas of crucial importance to the computer-based simulation of large space structure. The first area involves multibody dynamics (MBD) of flexible space structures, with applications directed to deployment, construction and maneuvering. The second area deals with advanced software systems, with emphasis on parallel processing. The latest research thrust in the second area, as reported here, involves massively parallel computers.

Task 1: MULTIBODY DYNAMICS

Background

This is a continuing research task that began in June 1986 and has progressed steadily over the past three years. The work has emphasized the following research components:

(1) Formulation of flexible multibody dynamics in a computationally oriented context.

(2) Formulation, implementation and evaluation of flexible three-dimensional beam elements capable of arbitrary motions and implementable in energy-conserving time integration methods.

(3) Development of a library of joint constraints to connect beam elements.

(4) Development, formulation and evaluation of energy-conserving time integration procedures, with emphasis on explicit-implicit partitioned solution algorithms for treating translational, rotational and constraint degrees of freedom in a staggered manner.

(5) Parallel implementation of multibody dynamics, including interconnection topology analysis and direct time integration.

(6) Completion of joint constraint library with contact-impact effects.

Over the past year work has concentrated on areas (4), (5) and (6). The principal investigator in areas (1) through (5) is Professor K. C. Park, whereas area (6) is jointly supervised by Professors Park and Felippa. Three doctoral students have carried out research in these areas: Jin-Chern Chiou (fully supported by this grant), Janice Downer (supported by a NASA fellowship) and Horacio de la Fuente (partly supported by this grant).

Following is a summary of accomplishments in areas (4) through (6), which are treated more fully in the enclosed reports (References 1-5).

Staggered Solution Procedures for MBD

An efficient staggered solution procedure for treating MBD systems has been developed, tested and implemented. The MBD equations of motion are partitioned so that the constraint forces appear as independent variables that can be integrated in time, separately from the mechanical variables. The latter are in turn partitioned into translational and rotational variables. The resulting partitioned equations of motion are integrated by a two-stage stabilized algorithm for updating both the translational coordinates and the angular velocities. Details of this procedure are given in Ref. 1, included in this report. The application of these procedure to simulation of flexible MBD systems composed of three-dimensional beams is described in Ref. 2, which is also included in this report.

MBD Topology Analysis for Parallel Implementation

A parallel partitioning scheme based on physical coordinate variables was developed to eliminate constraint forces and yield the MBD equation of motion in terms of independent coordinates. This scheme features an explicit determination of independent coordinates and the parallel computation of the null space of the constrained Jacobian matrix. This work is described in Ref. 3, which is included in the present report.

Parallel Direct Time Integration of MBD

Using the topological analysis developed under the previous task, a two-stage staggered algorithm for parallel computations has been developed, implemented and tested on a shared-memory parallel computer. The solution scheme features a new Schur-complement-based parallel preconditioned CG algorithm. This solution scheme is a "spin out". This work is described in Ref. 3.

Development of Contact-Impact Algorithms

This task began in May 1990 because of delayed funding and is in progress at the time of writing. Contact impact is represented by a fictitious, time-varying penalty spring that is designed to absorb the impulse of the contacting bodies in the form of a "penalty spring energy". This energy is released totally or partially on separation (partial release is used to model dissipation effects) and eventually the spring disappears. This new technique offers implementation advantages in that it can be easily accommodated in a variable step explicit time integration and this appears well suited to implementation on massively parallel computers. Preliminary results on simple impact problems are encouraging as regards general physical behavior as well as energy conserving characteristics.

Task 2: FINITE ELEMENT COMPUTATIONS ON A MASSIVELY PARALLEL COMPUTER

Background

This task represents the final phase of the software systems thrust. It was started in July 1989. The principle investigators are Professors C. Farhat and C.A. Felippa. Post-doc Research Associate E. Pramono has presently worked full-time on this project, which has also supported a graduate student (L. Crivelli) half-time. The main objective is the evaluation of the suitability of the Connection-Machine 2 (CM-2), a massively parallel computer, for large-scale finite element computations with emphasis on static analysis.

This work did not begin from scratch, but has substantially benefited from prior efforts. Investigation of the potential of the CM-2 for explicit dynamic calculations began in 1987 under NRL funding. This work involved Professors Farhat and K.C. Park, and post-doc

Research Associate N. Sobh. Work was carried out on the CM-2 computer at NRL, which is a half configuration of 32768 processors. The results of this study are presented in an enclosed report (Ref. 6), which is to appear shortly in *International Journal of Numerical Methods in Engineering.* Portions of that work have appeared in Ref. 7, which is also included.

In 1988 DARPA donated a small CM-2 (8192 processors) to the University of Colorado. The machine is presently installed at the National Center for Atmospheric Research (NCAR) and connected to the Campus Unix network. Although only one eighth of a full configuration, the increased availability and our deployment of real-time on-line graphics have substantially improved our ability to develop and test software. The CM-2 is not an easy machine to program because of its unconventional nature and the initial support of only two major programming languages with parallel constructions: CM-Lisp and C*. Virtually all programming has been done in C*, which is an object oriented superset of C and C++.

In August 1989 Dr. Sobh left us to take a faculty position at Old Dominion University. Dr. Pramono, whose prior experience in parallel processing had been on shared memory machines (especially Cray 2, Alliant and Convex using the Force Preprocessor) had to take over and gradually became an expert on the Connection Machine over the past six months.

Progress

Our work on the CM-2 to date has concentrated on the following software modules.

Decomposer. A general-purpose finite element model decomposer, described in Ref. 3, that takes as input an arbitrary mesh description, and produces a set of finite element data structures that can be loaded within one generic CM-2 chip.

Mapper. A general purpose mapper that assigns each of the data structures produced by the decomposer to a well defined chip. The goal of this allocation strategy is to reduce the distance that information has to travel between neighboring finite elements.

Residual Evaluator. This is a computational kernel that controls the direct calculation of element residuals, where "direct" means that no element stiffness matrices are evaluated. This kernel interacts with both a transient dynamics algorithm based on Central Difference, as well as an iterative solver based on Jacobi-Preconditioned Conjugate Gradients.

Element Library. This includes a 3D 2-node truss, a 3D 2-node beam, a 3D 8-node brick, a 2D 4-node quadrilateral and a 4-node ANS shell element. The shell element has been the latest one incorporated in this library, and testing was completed during May 1990.

Visualization. A parallel visualization kernel that operates in real time and which displays both wire frame representations of the initial and deformed mesh and shaded contour-value finite element plots as they are being computed.

Parallel I/O Manager. A kernel used to archive the computed results on the CM-2 data vault. It is based on the Parallel I/O Manager written by E. Pramono and described in Ref. 8.

These software modules together comprise a massively parallel prototype finite element code that effectively embeds MIMD computations on a SIMD hardware architecture.

Conclusions

Preliminary results using the prototype code with emphasis on truss and frame structures are reported in References 6 through 10. In general, it has been found that this highly parallel processor can outperform vector supercomputers such as the Cray family on explicit computations but not on implicit ones.

Several features distinguish the CM-2 from earlier SIMD hypercubes. On the hardware side we note the impressive number of crunching power and the fast parallel I/O capabilities. On the software side we note the virtual processor concept, which may be viewed as the dual of the better known virtual memory concept.

Mesh decomposition and processor-to-element mapping are the fundamental software modules that hold the key to massively parallel finite element computations. A given mesh is partitioned into 16 element subdomains that correspond to the 16-processor chips of the CM-2. This partitioning is carried out in a way that minimizes the number of nodes at the interface between the subdomains. As a result, only those processors that are mapped onto finite elements at the subdomain boundary communicate with processors packaged onto finite elements at the subdomain boundary communicate with processors packaged on different chips. Moreover, this partitioning is such that the bandwidth of the resulting subdomain is large enough to allow efficient use of the 12 interchip wires.

The mapping algorithm attempts at reducing the distance information has to travel over the communication network. It searches iteratively for an optimal mapping through a 2-step minimization of the communication costs associated with candidate mappings.

The following is a summary of the key conclusions reported in the referenced papers.

(1) The current CM-2 processor memory size of 64 Kbits penalizes high order elements in the sense that only small VP (virtual processor) ratios can be achieved. Thus the current configuration favors simpler elements. (This restriction should disappear in

the CM-3 model, which will have 1Mbit of memory per processor and an aggregate computing power of over 1000 Gflops.)

(2) Three-dimensional and higher-order finite elements induce longer communication times.

(3) Mesh irregularities slow down the computation speed in various ways.

(4) The Data Vault is very effective at reducing I/O time.

(5) The Frame Buffer is ideal for real-time visualization.

(6) The Virtual Processor concept outperforms substructuring.

Ongoing Work

We have found that the CM-2 can outperform the Cray-2 on *explicit* calculations for which sustained rates over 1 Gigaflop are possible. Given the intrinsic scalability of the massively parallel architecture (for example, the 1 Teraflop CM-3 under development for DARPA) there is little question as to the future potential for that class of computations which arise naturally in dynamic simulations. Projections are for 100-1000 times what the fastest Cray can achieve.

On the other hand, *implicit* calculations arise naturally in the solutions of static problems. This class of calculation places a higher burden on communication, which has a detrimental effect on performance. For such algorithms the vector supercomputers still outperform the CM-2. Semi-iterative methods such as the conventional Conjugate Gradient (CG) also suffer to some degree from communications overhead since information has to be gathered from shared finite element nodes in residual calculations.

Over the past six months, Professor Farhat in collaboration with Dr. Roux of ONERA (France) has developed an unconventional form of the CG algorithm called the "hybrid" or "tearing" method. The primary objective in this development is to reduce communication overhead on local memory parallel computers. A secondary objective is to reduce the number of iterations for convergence. The present version of algorithm is described in some detail in Refs. 11, 12 and 13. The initial version was coded in Fortran augmented with the Force preprocessor and tested on the Cray YMP. These tests provided confidence in the convergence characteristics on static problems involving up to 48,000 equations. A subsequent version was ported to the Los Alamos iPSC Hypercube, on which the reduced communication overhead was verified. As final tests, we plan to recode the algorithm in C^* for the CM-2 and compare with the conventional CG implementation. Because of the local memory limitations, however, the domain decomposition on the CM-2 is done at the element level.

Final Benchmarking Work

During the period of March to date (July 1990) we have benchmarked large-scale static problems on the CM-2 versus the Cray 2 and Cray YMP. The results are being analyzed at the time of the writing and will be subsequently reported in the literature.

REFERENCES

References marked with an asterisk (*) are enclosed in the present report.

1. * K. C. Park, J. C. Chiou and J. D. Downer, "Staggered Solution Procedures for Multibody Dynamics Simulation," December 1989, submitted to *J. Guidance & Control.*
2. * J. D. Downer, K. C. Park and J. C. Chiou, "A Computational Procedure for Multibody Systems Including Flexible Beam Dynamics," presented at the SDM Conference, Long Beach, CA April 1990; manuscript submitted to *Computer Methods in Applied Mechanics and Engineering.*
3. * J. C. Chiou, K. C. Park and C. Farhat, "A Natural Partitioning Scheme for Parallel Simulation of Multibody Systems," to be presented at the SDM Conference, Dec 1990; manuscript to be submitted to *Computer Methods in Applied Mechanics and Engineering.*
4. * K. C. Park and J. C. Chiou, "Stabilization of Computational Procedures for Constrained Dynamical Systems," *Journal of Guidance Controls and Dynamics*, **11**, (1988) pp. 365-370.
5. * K. C. Park, J. C. Chiou and J. D. Downer, "Explicit-Implicit Staggered Procedures for Multibody Dynamics Analysis," *J. of Guidance, Controls and Dynamics*, **13**, (1990) pp. 562-570.
6. * C. Farhat, N. Sobh and K.C. Park, "Transient Finite Element Analysis on 65536 Processors: The Connection Machine," *Int. J. Num. Meth. Engrg.*, in press.
7. * C. Farhat, "Which Parallel Finite Element Algorithm for which Architecture and which Problem," *Proceedings Session in Computational Structural Mechanics and Multidisciplinary Optimizations*, ed. by R.V. Grandhi, W.H. Stroud and V.B. Venkayya, ASME Winter Meeting San Francisco, Dec 10-15 (1989), submitted to *Engineering Computations.*
8. C. Farhat, N. Sobh and K.C. Park, "Dynamic Finite Element Simulation on the Connection Machine," *Int. J. High Speed Computing*, **1**, No. 2 (1989) pp. 289-302.
9. C. Farhat, "On the Mapping of Massively Parallel Processors onto Finite Element Graphs," *Computers and Structures*, **32**, No. 2 (1989) pp. 347-354.
10. C. Farhat, E. Pramono and C.A. Felippa, "Towards Parallel I/O in Finite Element Computations," *Int. J. Num. Meth. Engrg.*, **28**, (1989) pp. 2541-2553.
11. C. Farhat and F.X. Roux, "An Unconventional Domain Decomposition Method for an Efficient Parallel Solution of Large-Scale Finite Element Systems," presented at the *SIAM Symposium on Parallel Computation*, Copper Mountain, CO., April 1990
12. * C. Farhat and F. Roux, "A Method of Finite Element Tearing and Interconnecting and its Parallel Solution Algorithm," May 1990, submitted to *SIAM J. Scientific Computations.*
13. * C. Farhat and M. Geradin, "Using a Reduced Number of Lagrange Multipliers for Assembling Parallel Incomplete Field Finite Element Approximations," July 1990, to be submitted to *Computer Methods in Applied Mechanics and Engineering.*

Staggered Solution Procedures for Multibody Dynamics Simulation

K. C. Park, J. C. Chiou, and J. D. Downer
Department of Aerospace Engineering Sciences
and Center for Space Structures and Controls
University of Colorado at Boulder
Boulder, CO 80309-0429, USA

I. Introduction

Simulation of multibody dynamics systems – such as robotic manipulators, automobiles maneuvering and satellites deployment – remains a challenge to the dynamist due to its increasing roles in design improvements, control and safe operation. Because of substantial progress made during the past three decades in formulation[1-19], constraint treatment and solution techniques[21-36] and the availability of multibody dynamics simulation packages[37-42], it has now become almost a routine practice to perform realistic modeling and assessment of some practical problems such as mechanical linkages and manipulations of robotic arms if multibody components consist mostly of rigid bodies, discrete springs and dampers (see, e.g., Haug[15]). However, substantial advances in modeling, formulation and computational methods are necessary in order to develop a real-time simulation capability for ground vehicle maneuvering dynamics, robotic manipulations and space structures deployment/assembly.

Specifically, improved modeling of flexibility for localized motions and geometric nonlinearities, material nonlinearities and contact/friction phenomena, robust and accurate treatment of the system constraint conditions and efficient use of emerging computer hardware/software technology continue to offer intense research opportunities. Thus, the development of a real-time multibody dynamics simulation capability requires a concerted integration of various modeling, formulation and computational aspects. These include: selection of a data structure for describing the system topology, computerized generation of the governing equations of motion, implementation of suitable solution algorithms, incorporation of constraint conditions and easy interpretation of the simulation results. Of these, this chapter is concerned with three computational aspects of multibody dynamics simulation: direct time integration of the governing equations of motion, stabilization of constraint solution process and their computer implementation aspects.

From the computational viewpoint, multibody dynamics (MBD) problems are distinct from the structural dynamics problems in that the solution of MBD problems must also

satisfy, at each time integration step, the attendant kinematic and equilibrium constraints. This has motivated many dynamists to develop various techniques, in addition to direct integration algorithms, for accurately and efficiently handling the system constraints. Hence, reliability and cost of existing MBD simulation packages have been strongly affected by how efficiently and accurately the constraints are preserved during the numerical solution stage.

In general, there have been two types of direct time integration algorithms for the transient response analysis of dynamical systems: explicit and implicit algorithms (see, e.g., Hughes and Belytschko[43], Park[44] and Belytschko, Englemann and Liu[45]). Currently, implicit algorithms appear to be favored by many MBD specialists when both the generalized coordinates and the constraint forces are treated as the unknowns. In this case, the corresponding formulations incorporate the system constraints by the Lagrange multipliers method. It has been well known that the resulting Newton-like solution matrix is stiff. This has led to implicit time discretization of the constraint-augmented equations and simultaneous solution of both the generalized coordinates and the Lagrange multipliers. This approach has been extensively investigated by Gear[21], Baumgarte[22,29] Orlandea, Chase and Calahan[23], Petzold[27], Nikravesh[31], among others. Because these methods solve both the generalized coordinates and the constraint forces simultaneously, they will be called the *simultaneous solution methods* in this chapter.

On the other hand, if the constraints are eliminated so as to reduce the number of unknowns, it is possible for one to employ either implicit or explicit algorithm. For this situation, one may invoke either a geometric or algebraic procedure to streamline the resulting equations of motion if the system topology is an open tree. In essence, geometric procedures have utilized an open-tree topology such as the use of the incidence matrix by Wittenburg[10] and the body array matrix by Huston[19]. Some of the proposed algebraic procedures include the singular decomposition by Walton et al[20], the use of the generalized speed of Kane and Levinson[20], the coordinate partitioning technique by Wehage and Haug[28], the selection of independent coordinates through the natural-coordinate formulation of Garcia de Jalon et al[33] and the so-called order-N procedures of Armstrong[11], Hollerbach[12], Schwertassek and Roberson[17], Orin, et al[25], among others.

As the complexity of MBD systems increases, the simultaneous solution methods have become less attractive. This is due to matrix ill-conditioning especially for the so-called *index* two and higher index problems (see, e.g., Ref. 27 and Brenan, Campbell and Petzold[46] for the definition of *index* for constraint characterization), divergence of the solution away from the constraint conditions, and ultimately, due to a large size of the equations that must be handled. As an alternative to the simultaneous solution methods, a series of computational methods that employ a *divide-and-conquer* strategy have been developed, which are termed as *partitioned solution procedures* presented in Park[47], Felippa and Park[48] and Park and Felippa[49]. As an example, partitioned solution procedures allow one to analyze fluid-structure interaction problems with two separate single-field analysis packages, namely, the structural dynamics module and the fluid dynamics analyzer. At each time integration step, one may advance the solution of structural equations of motion by treating the fluid coupling term as an external force. Once the structural coordinates are advanced, the fluid state variables can be advanced by treating the structural coupling

terms as a source term. A naive partitioned procedure, however, can suffer from a loss of accuracy as well as computational stability. Thus, a combination of equation augmentation and stabilization should be devised to recover the accuracy loss and maintain unconditional stability. Such a solution procedure is in contrast to a practice of embedding both the structural and fluid dynamics attributes into a combined analysis program.

The numerical solution procedure for MBD systems which we advocate in this chapter is termed a *staggered MBD solution porcedure* that solves the generalized coordinates in a separate module from that for the constraint force. This requires a reformulation of the constraint conditions so that the constraint forces can also be integrated in time. A major advantage of such a partitioned solution procedure is that additional analysis capabilities such as active controller and design optimization modules can be easily interfaced without embedding them into a monolithic program. To this end, the rest of the chapter is organized as follows.

After introducing the basic equations of motion for MBD system in the next section, Section III briefly reviews some constraint handling techniques and introduces the staggered stabilized technique[34,35] for the solution of the constraint forces as independent variables.

The numerical direct time integration of the equations of motion is described in Section IV. As accurate damping treatment is important for the dynamics of space structures, we have employed the central difference method and the mid-point form of the trapezoidal rule since they engender no numerical damping. This is in contrast to the current practice in dynamic simulations of ground vehicles by employing a set of backward difference formulas[46]. First, the equations of motion is partitioned according to the translational and the rotational coordinates. This sets the stage for an efficient treatment of the rotational motions via the singularity-free Euler parameters. The resulting partitioned equations of motion are then integrated via a two-stage explicit stabilized algorithm for updating both the translational coordinates and angular velocities[34]. Once the angular velocities are obtained, the angular orientations are updated via the mid-point implicit formula employing the Euler parameters.

When the two algorithms, namely, the two-stage explicit algorithm for the generalized coordinates and the implicit staggered procedure for the constraint Lagrange multipliers, are brought together in a staggered manner, they constitute a staggered explicit-implicit procedure which are summarized in Section V. Section VI presents some example problems and discussions concerning several salient features of the staggered MBD solution procedure are offered in Section VII.

II. Governing Equations of Motion

The Lagrangian equations of motion for mechanical systems that are free from any constraint can be written, for the generalized coordinate component u_i, as

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{u}_i} - \frac{\partial L}{\partial u_i} = Q_i, \quad i = 1 \ldots n. \tag{1}$$

where L is the system Lagrangian, t is the time, $(\dot{})$ denotes time differentiation and Q_i is the generalized applied force. It is well-known that, if there are m-constraint conditions

imposed on $\{u_i, \quad i = 1 \ldots n\}$, the above equation must be modified as

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{u}_i} - \frac{\partial L}{\partial u_i} = Q_i + \sum_{k=1}^{m} \lambda_k B_{ki}, \quad i = 1 \ldots n, \tag{2}$$

where λ is the Lagrange multiplier and B_{ki} is the i-th gradient component of the k-th constraint equation, *viz*, for configuration constraints

$$\Phi_k(\mathbf{u}) = 0, \quad B_{ki} = \frac{\partial \Phi_k}{\partial u_i}, \quad k = 1 \ldots m \tag{3}$$

and for motion constraints

$$\Phi_k(\mathbf{u}, \dot{\mathbf{u}}) = 0, \quad B_{ki} = \frac{\partial \Phi_k}{\partial \dot{u}_i}, \quad k = 1 \ldots m. \tag{4}$$

Therefore, regardless of the nature of constraints one may express the equations of motion with constraints in the following form:

$$\begin{bmatrix} M & B^T \\ B & 0 \end{bmatrix} \begin{pmatrix} \ddot{\mathbf{u}} \\ \lambda \end{pmatrix} = \begin{pmatrix} Q \\ c \end{pmatrix} \tag{5}$$

where M is a positive-definite matrix and c depends on the nature of constraints. For example, for configuration constraints we have

$$c = -\frac{\partial}{\partial u}\left(\frac{\partial \Phi}{\partial u}\dot{u}\right) - 2\frac{\partial}{\partial t}\left(\frac{\partial \Phi}{\partial u}\dot{u}\right) - \frac{\partial^2 \Phi}{\partial t^2} \tag{6}$$

and for motion constraints

$$c = -\frac{\partial \Phi}{\partial t}. \tag{7}$$

An implicit time integration formula to solve (5) may be written as

$$\begin{cases} \dot{\mathbf{u}}^n = \delta \ddot{\mathbf{u}}^n + \mathbf{h}_{\dot{u}}^n \\ \mathbf{u}^n = \delta \dot{\mathbf{u}}^n + \mathbf{h}_u^n \end{cases} \tag{8}$$

where δ is a stepsize that is dependent on the choice of formula, and $\mathbf{h}_{\dot{u}}^n$ and $\mathbf{h}_u^n$ are formula-dependent historical vectors that consist of past-step solution components[48,50]. As an example, the trapezoidal rule has the following δ and historical vectors

$$\begin{cases} \delta = h/2 \\ \mathbf{h}_{\dot{u}}^n = \dot{\mathbf{u}}^{n-1} + \delta \ddot{\mathbf{u}}^{n-1} \\ \mathbf{h}_u^n = \mathbf{u}^{n-1} + \delta \dot{\mathbf{u}}^{n-1} \end{cases} \tag{9}$$

where h is the time-step increment.

Substitution of (8) into (5) yields

$$\begin{bmatrix} M & \delta^2 B^T \\ B & 0 \end{bmatrix} \begin{pmatrix} \mathbf{u}^n \\ \lambda^n \end{pmatrix} = \begin{pmatrix} r_u^n \\ r_\lambda^n \end{pmatrix} = \begin{pmatrix} Mh^n + \delta^2 Q^n \\ Bh^n + \delta^2 c \end{pmatrix} \tag{10}$$

In practice, in order to avoid pivoting and to maintain high accuracy, the solution of the above difference equations is carried out as follows. First, since M is nonsingular for properly formulated dynamical problems, one computes

$$\mathbf{u}_u = M^{-1}\mathbf{r}_u^n, \quad C = M^{-1}B^T, \quad A = BC \tag{11}$$

and factors A. Second, one obtains λ^n by solving

$$\lambda^n = A^{-1}(B\mathbf{u}_u - \mathbf{r}_\lambda^n)/\delta^2 \tag{12}$$

Finally, $\mathbf{u}^n$ is obtained from

$$\mathbf{u}^n = \mathbf{u}_u - \delta^2 C\lambda^n \tag{13}$$

It should be noted that the accuracy loss associated with the factoring of an ill-conditioned matrix $BA^{-1}B^T$ and the subsequent backsubstitutions can severely influence the solution accuracy of not only the Lagrange multipliers but also the generalized coordinates as seen from (12) and (13). This has motivated many numerical analysts to undertake the development of methods for differential-algebraic systems as the recent monograph[46] and references therein attest to their rich numerical properties. It is generally agreed that the present status of differential-algebraic methods yield robust solutions for problems of *index one*, but can suffer from inaccurate solutions of the Lagrange multipliers for higher *index* problems. Observe that many practical multibody dynamics problems are characterized by index greater than one. Hence, the need to compute accurately the constraint forces remains a challenge. For instance, for lock-up mechanisms that are activated when truss structures are fully deployed in space often introduce stiff responses with nearly singular state of $BM^{-1}B^T$. It is with these problems for which more robust constraint computation algorithms are called for.

One way to improve the accuracy of constraint force computations is to adopt index reduction strategies as discussed in Ref. 46. However, index reduction inevitably introduces additional system degrees of freedom in the resulting differential-algebraic equations, thus destroying the matrix sparsity of (5) in addition to the increased size of the matrix B. In what follows we present an alternative approach based on a parabolic regularization of the equations for the Lagrange multipliers, which preserves the first row of (5) and enables us to solve λ from the parabolic differential equations.

III. Constraint Handling Techniques

As alluded to in Introduction, techniques for handling the system constraints constitute a major part of solution procedures for the numerical simulation of multibody dynamics systems. In this section, we will first review the coordinate partitioning technique, Baumgarte's technique and the penalty technique. The staggered stabilization procedure

which we advocate will then be described in detail. A distinct feature of the staggered stabilization procedure is that it can be implemented in a stand-alone module, thus can be interfaced not only with the equation solver for rigid-body systems but with that for flexible-body systems as well.

A. Coordinate Partitioning Technique

In the coordinate partitioning[28,33] or singular decomposition technique[20,30], one selects a rank sufficient part of **B** and partitions it as

$$B = \lfloor B_i \quad B_e \rfloor, \quad \mathbf{u} = \lfloor \mathbf{u}_i \quad \mathbf{q}_e \rfloor \tag{14}$$

where the rank of $B_i(m \times m)$ is m and the subscripts (i, e) refer to *internal* and *external* variables, respectively. First, we express $\mathbf{u}_i$ in terms of $\mathbf{u}_e$ as

$$\mathbf{u}_i^n = B_i^{-1}(\mathbf{r}_\lambda^n - B_e \mathbf{u}_e^n) \tag{15}$$

Since we have

$$\lfloor -B_e^T B_i^{-T} \quad I \rfloor \begin{Bmatrix} B_i^T \\ B_e^T \end{Bmatrix} = 0 \tag{16}$$

The first row of (10) reduces to

$$(M_e + T^T M_i T)\mathbf{q}_e^n = \mathbf{r}_e^n \tag{17}$$

where

$$T = B_i^{-1} B_e, \quad M = \begin{bmatrix} M_i & 0 \\ 0 & M_e \end{bmatrix}, \quad \mathbf{r}_u^n = \begin{Bmatrix} \mathbf{r}_{u_i}^n \\ \mathbf{r}_{u_e}^n \end{Bmatrix} \tag{18}$$

and

$$\mathbf{r}_e^n = \mathbf{r}_{u_e}^n - T^T \mathbf{r}_{u_i}^n + T^T M_i B_i^{-1} \mathbf{r}_\lambda^n \tag{19}$$

Once one obtains $\mathbf{u}_e^n$, one can obtain $\mathbf{u}_e^n$ from (15) and similarly λ from (12). Note that even though (17) has a smaller dimension than that of (10a), its left-hand side matrix is in general full since T given by (18a) is in general full. Hence, unless T is a constant matrix, one must refactor the solution matrix in (17) whenever a new T is formed.

B. Baumgarte's Technique

Baumgarte's technique[22,29] is based on the observation that the errors committed in computing the constraint conditions (3) or (4) can either be critically damped out or exponentially decreased as the integration process continues. Mathematically, this can be stated for the configuration constraint equation(3) as

$$\ddot{\Phi} + 2\alpha\dot{\Phi} + \beta\Phi = 0 \tag{20}$$

or the motion constraint equation(4) as

$$\dot{\Phi} + \gamma\Phi = 0 \tag{21}$$

In terms of the general constraint equation augmentation as given by (5b), the preceding stabilization is equivalent to modifying c in (5b) accordingly. Hence, the technique can be implemented within the standard augmented form of the equations of motion (5). However, if $BM^{-1}B^T$ is ill-conditioned, which can happen since B is in general state-dependent, the accuracy of generalized constraint force, λ, can be considerably degraded. This can occur if any two rows of B are physically similar (i.e., when two members form a straight line) or numerically close during three-dimensional orientations.

C. Penalty Technique

In the two constraint handling techniques outlined so far, the objective was to satisfy the constraint condition

$$\Phi = 0 \tag{22}$$

whose differentiated forms were augmented to the equations of motion. In the penalty procedure, one adopts

$$\lambda = \frac{1}{\epsilon}\Phi, \quad \epsilon \to 0 \tag{23}$$

as the basic constraint equations instead of the twice-differentiated form adopted in (5).

It is noted that the penalty formulation tacitly assumes that there will be violations of the constraint condition in actual computations as discussed in Lanczos[51]. If one substitutes (23) into the governing equations of motion, the resulting equation becomes

$$M\ddot{\mathbf{u}} + \frac{1}{\epsilon}B^T\Phi = Q \tag{24}$$

A major drawback of the above penalty procedure is that, once an error is committed in computing λ, there is no compensation scheme by which the drifting of the numerical solution can be corrected. This has led to the development of a staggered stabilized procedure as described below.

D. Staggered Stabilization Procedure

To illustrate this procedure we will consider the case of nonholonomic constraints. Instead of substituting the penalty expression directly into the governing equations of motion, first we differentiate (23) once to obtain

$$\dot{\boldsymbol{\lambda}} = \frac{1}{\epsilon}(\mathbf{B}\ddot{\mathbf{u}} + \frac{\partial \Phi}{\partial t}) \tag{25}$$

where we assume the penalty parameter, ϵ, to be constant.

Second, we obtain for $\ddot{\mathbf{u}}$ from (5a) in the form

$$\ddot{\mathbf{u}} = \mathbf{M}^{-1}(\bar{\mathbf{Q}} - \mathbf{B}^T\boldsymbol{\lambda}) \tag{26}$$

and substitute it into (25) to yield

$$\epsilon\dot{\boldsymbol{\lambda}} + \mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T\boldsymbol{\lambda} = \mathbf{r}_\lambda = \mathbf{B}\mathbf{M}^{-1}\bar{\mathbf{Q}} + \frac{\partial \Phi}{\partial t} \tag{27}$$

Notice that the homogeneous part of the above stabilized equation in terms of the generalized constraint forces, $\boldsymbol{\lambda}$, has the following companion eigenvalue problem:

$$(\gamma + \mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T/\epsilon)\mathbf{y} = 0 \tag{28}$$

where $\{\gamma_k, \quad k = 1 \ldots m\}$ are the eigenvalues of the homogeneous operator for the new stabilized constraint equations (27). Since γ_k also dictates how the errors in the constraint forces will diminish with time, the errors committed in the constraint conditions will decay with their corresponding different response time constants. This physically oriented stabilization property of the present technique is in contrast to that of Baumgarte's technique wherein all the error components diminish according to a single time constant.

Third, this technique enables one to solve for $\boldsymbol{\lambda}$ from the stabilized differential equation (27). Specifically, one now has two coupled equations, one set for the generalized coordinates $\mathbf{u}$ and the other for the generalized constraint forces $\boldsymbol{\lambda}$, which are recalled here from (5a) and (27) for the case of nonholonomic constraints:

$$\begin{bmatrix} \mathbf{M} & 0 \\ 0 & \epsilon \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{u}} \\ \dot{\boldsymbol{\lambda}} \end{Bmatrix} + \begin{bmatrix} 0 & \mathbf{B}^T \\ 0 & \mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T \end{bmatrix} \begin{Bmatrix} \dot{\mathbf{u}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \bar{\mathbf{Q}} \\ \mathbf{r}_\lambda \end{Bmatrix} \tag{29}$$

Note that the above coupled equations directly provide the desired differential equations for a pair of $\lfloor \mathbf{u} \quad \boldsymbol{\lambda} \rfloor$.

For holonomic constraints, one has several stabilization possibilities. The one we have chosen is to integrate the governing equations of motion once to obtain

$$\dot{\mathbf{u}}^n = \delta \mathbf{M}^{-1}(\bar{\mathbf{Q}}^n - \mathbf{B}^T\boldsymbol{\lambda}^n) + \mathbf{h}_{\dot{u}}^n \tag{30}$$

which is substituted into

$$\dot{\boldsymbol{\lambda}} = \frac{1}{\epsilon}(\mathbf{B}\dot{\mathbf{u}} + \frac{\partial \Phi}{\partial t}) \tag{31}$$

to yield:

$$\epsilon\dot{\boldsymbol{\lambda}}^n + \delta\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T\boldsymbol{\lambda}^n = \mathbf{B}(\delta\mathbf{M}^{-1}\bar{\mathbf{Q}}^n + \mathbf{h}_{\dot{u}}^n) + \frac{\partial \Phi}{\partial t} \tag{32}$$

It is observed that, even if $\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T$ is almost singular, this stabilization technique as derived in (27) and (32) would not cause numerical difficulty in computing $\boldsymbol{\lambda}$ since the solution iteration matrix becomes $(\epsilon + \delta\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T)$ for nonholonomic cases and $(\epsilon + \delta^2\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T)$ for holonomic cases. It is noted that one must choose ϵ in such a way to maintain robust solution when $\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T$ becomes ill-conditioned by choosing $\epsilon \sim c/|(\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T)^{-1}| \cdot |\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T|$ where c is the solution accuracy desired for λ.

Integration of the above equation by the mid-point implicit rule yields the following difference equation:

$$\begin{cases} (\epsilon\mathbf{I} + \frac{h}{4}\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T)\boldsymbol{\lambda}^{n+1/4} = \frac{h}{4}(\mathbf{r}_\lambda^{n+\frac{1}{2}} + \mathbf{r}_\lambda^n) + \epsilon\boldsymbol{\lambda}^n \\ \boldsymbol{\lambda}^{n+1/2} = 2\boldsymbol{\lambda}^{n+1/4} - \boldsymbol{\lambda}^n \end{cases} \tag{33}$$

It has been shown that the staggered stabilized procedure for the solution of the constraints offers not only a modular software package to treat the constraints but also has been found to yield more robust solutions compared to the techniques proposed by Baumgarte as reported in Park and Chiou[35]. In particular, even when $\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T$ becomes nearly singular, the staggered stabilized procedure (33) gives stable and acceptable solutions whereas the constraint forces computed by the Baumgarte's technique diverge.

IV. Solution Algorithms for Generalized Coordinates

In addition to the choice of implicit and explicit formulas, the recognition that the equations of motion for multibody systems with constraints are not ordinary differential equations (ODEs) (see, e.g., Petzold[27]) has placed a unique requirement in the selection of solution algorithms for multibody dynamics problems. From the user's viewpoint, one has the option of either employing one of the available ODE packages (see Enright[32] for existing ODE packages) or building a special solution module. It should be noted that, since the integration of angular velocity vector does not lead to angular orientations, one must solve a set of kinematical equations to obtain the desired angular orientations.

In this section we describe an explicit-implicit transient analysis algorithm that exploits the special kinematical relationships of the generalized rotational coordinates vs. the angular velocity, namely, the Euler parameters[34]. The integration of the translational coordinates and the angular velocity is accomplished by the central difference formula. It should be mentioned that the use of the central difference formula does impose a stepsize restriction due to its stability limit ($\omega_{max}h \leq 2$) where ω_{max} is the highest angular velocity of the system components for rigid-body systems or the highest frequency of the entire flexible members for flexible-body systems. The simplicity of its programming effort and robustness of its solution results can often become compelling enough to adopt an explicit formula, which is the view taken here.

In conventional structural dynamics analysis, explicit time integration of the equations of motion by the central difference formula involves the following two updates per step:

$$\begin{cases} \dot{\mathbf{u}}^{n+1/2} = \dot{\mathbf{u}}^{n-1/2} + h\ddot{\mathbf{u}}^n \\ \mathbf{u}^{n+1} = \mathbf{u}^n + h\dot{\mathbf{u}}^{n+1/2} \end{cases} \tag{34}$$

Unfortunately, this simplistic procedure is not directly applicable to the rotational part of the equations of motion as ω is not directly integrable, except for some special kinematic configurations. This motivates us to partition $\dot{\mathbf{q}}$ into the translational velocity vector, $\dot{\mathbf{d}}$, which is directly integrable and the angular velocity vector, ω, which is not, and treat them differently, viz.:

$$\ddot{\mathbf{u}} = \begin{Bmatrix} \ddot{\mathbf{d}} \\ \dot{\omega} \end{Bmatrix}, \qquad \dot{\mathbf{u}} = \begin{Bmatrix} \dot{\mathbf{d}} \\ \omega \end{Bmatrix} \tag{35}$$

The equations of motion (5a) can be partitioned according to the above partitioning:

$$\begin{bmatrix} M_d & 0 \\ 0 & M_\omega \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{d}} \\ \dot{\omega} \end{Bmatrix} = \begin{Bmatrix} Q_d \\ Q_\omega \end{Bmatrix} \tag{36}$$

where

$$\left\{ \begin{array}{c} Q_d \\ Q_\omega \end{array} \right\} = \left\{ \begin{array}{l} \mathbf{f}_d - \mathbf{D}_d(\dot{\mathbf{d}}) - \mathbf{S}_d(\mathbf{d},\mathbf{e}) - \mathbf{B}_d^T\boldsymbol{\lambda} \\ \mathbf{f}_\omega - \mathbf{D}_\omega(\boldsymbol{\omega}) - \mathbf{S}_\omega(\mathbf{d},\mathbf{e}) - \mathbf{B}_\omega^T\boldsymbol{\lambda} \end{array} \right\} \tag{37}$$

in which the subscripts (d,ω) refer to the translational and the rotational motions, respectively, $\mathbf{f}$ is the external force vector, $\mathbf{D}$ is the generalized damping force including the centrifugal force, $\mathbf{S}$ is the internal force vector including member flexibility, $\mathbf{q}$ is the angular orientation parameters, $\mathbf{B_d}$ and $\mathbf{B}_\omega$ are the partition of the combined gradient matrices of the constraint conditions (3) or (4) that are symbolically expressed as

$$\mathbf{B} = \mathbf{B}_N + \mathbf{B}_H, \qquad \boldsymbol{\lambda} = \boldsymbol{\lambda}_N + \boldsymbol{\lambda}_H \tag{38}$$

To effect the body-by-body integration for the rotational degrees of freedom, we partition $\dot{\boldsymbol{\omega}}$ further into

$$\dot{\boldsymbol{\omega}} = \lfloor \dot{\boldsymbol{\omega}}^1, \dot{\boldsymbol{\omega}}^2, \ldots, \dot{\boldsymbol{\omega}}^P \rfloor^T \tag{39}$$

where $\dot{\boldsymbol{\omega}}^{(j)}$ is a (3×1) angular acceleration vector for the j-th body,

$$\dot{\boldsymbol{\omega}}^{(j)} = \lfloor \omega_1^{(j)}, \omega_2^{(j)}, \omega_3^{(j)} \rfloor \tag{40}$$

We now present the update algorithm for both translational and rotational coordinates.

A. Update of Translational and Angular Velocity

First, assume that $\mathbf{d}^{n+1/2}$ and $\mathbf{q}^{n+1/2}$ are already computed so that we can compute $\ddot{\mathbf{d}}^{n+1/2}$ and $\dot{\boldsymbol{\omega}}^{n+1/2}$ by (36), namely,

$$\left\{ \begin{array}{c} \ddot{\mathbf{d}}^{n+1/2} \\ \dot{\boldsymbol{\omega}}^{n+1/2} \end{array} \right\} = -M^{-1} \left\{ \begin{array}{l} \mathbf{D}_d^{n+\frac{1}{2}} + \mathbf{S}_d^{n+\frac{1}{2}} - \mathbf{B}_d^T\boldsymbol{\lambda}^{n+\frac{1}{2}} \\ \mathbf{D}_\omega^{n+\frac{1}{2}} + \mathbf{S}_\omega^{n+\frac{1}{2}} - \mathbf{B}_\omega^T\boldsymbol{\lambda}^{n+\frac{1}{2}} \end{array} \right\} \tag{41}$$

Second, we update the translational velocity and the angular velocity vectors at the step (n+1) by

$$\left\{ \begin{array}{l} \dot{\mathbf{d}}^{n+1} = \dot{\mathbf{d}}^n + h\ddot{\mathbf{d}}^{n+1/2} \\ \boldsymbol{\omega}^{n+1} = \boldsymbol{\omega}^n + h\dot{\boldsymbol{\omega}}^{n+1/2} \end{array} \right. \tag{42}$$

Third, we update the translational displacement, $\mathbf{d}$, by

$$\mathbf{d}^{n+3/2} = \mathbf{d}^{n+1/2} + h\dot{\mathbf{d}}^{n+1} \tag{43}$$

However, the updating of the angular orientation requires somewhat involved computations. To this end, we will employ the Euler parameters and update them accordingly.

B. Update of Euler Parameters and Angular Velocity

As mentioned in conjunction with a direct use of (34) for integrating the rotational equations of motion, it is necessary for one to introduce a set of generalized coordinates whose time rate can be related to the angular velocity. To this end, we employ the four-parameter Euler representation of the angular velocity for each body as (see, e.g., Wittenburg[10]):

$$\dot{\mathbf{q}} = \frac{1}{2}\begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & -\tilde{\boldsymbol{\omega}} \end{bmatrix} \mathbf{q} = A(\omega)\mathbf{q}, \quad \mathbf{q} = \lfloor q_0 \quad q_1 \quad q_2 \quad q_3 \rfloor^T \tag{44}$$

that is subject to the constraint:

$$\mathbf{q}^T\mathbf{q} = 1 \tag{45}$$

where

$$\tilde{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}, \qquad \boldsymbol{\omega} = \lfloor \omega_1 \quad \omega_2 \quad \omega_3 \rfloor^T \tag{46}$$

and the nodal-designation superscript is omitted for notational simplicity.

We adopt the mid-point implicit procedure to integrate the Euler parameters:

$$\begin{cases} \dot{\mathbf{q}}^{n+1} = \mathbf{A}(\boldsymbol{\omega}^{n+1}) \cdot \mathbf{q}^{n+1} \\ \mathbf{q}^{n+1} = \mathbf{q}^{n+1/2} + \frac{h}{2}\dot{\mathbf{q}}^{n+1} \\ \mathbf{q}^{n+3/2} = 2\mathbf{q}^{n+1} - \mathbf{q}^{n+1/2} \\ (\mathbf{q}^{n+3/2})^T \cdot \mathbf{q}^{n+3/2} = 1 \end{cases} \tag{47}$$

It should be noted that the mid-point implicit update is no more costly than any explicit as the solution matrix inversion can be explicitly obtained.

Finally, once $\mathbf{q}^{n+3/2}$ is computed from (47), it is often required to compute the body-fixed basis vector, $\mathbf{b} = \lfloor \mathbf{b}_1 \quad \mathbf{b}_2 \quad \mathbf{b}_3 \rfloor^T$ in terms of the inertial basis vectors, $\mathbf{e} = \lfloor \mathbf{e}_1 \quad \mathbf{e}_2 \quad \mathbf{e}_3 \rfloor^T$. These two vectors are related by

$$\mathbf{b} = \mathbf{R}\mathbf{e} \tag{48}$$

where

$$\mathbf{R} = \begin{bmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & 2(q_0^2 + q_3^2) - 1 \end{bmatrix} \tag{49}$$

C. Update of $\dot{\mathbf{d}}, \omega, \mathbf{d}, \mathbf{q}$ at the $(n+2)$–step

So far we have advanced from the step (n+1) to the step (n+3/2). In other words, we have advanced only half of the total step. For the next step, viz, the step (n+2) from (n+3/2), we employ the following sequence of computations:

$$\begin{Bmatrix} \ddot{\mathbf{d}}^{n+1} \\ \dot{\omega}^{n+1} \end{Bmatrix} = -M^{-1} \begin{Bmatrix} \mathbf{D}_d^{n+1} + \mathbf{S}_d^{n+1} - \mathbf{B}_d^T \boldsymbol{\lambda}^{n+1} \\ \mathbf{D}_\omega^{n+1} + \mathbf{S}_\omega^{n+1} - \mathbf{B}_\omega^T \boldsymbol{\lambda}^{n+1} \end{Bmatrix} \tag{50}$$

$$\begin{cases} \dot{\mathbf{d}}^{n+3/2} = \dot{\mathbf{d}}^{n+1/2} + h\ddot{\mathbf{d}}^{n+1} \\ \omega^{n+3/2} = \omega^{n+1/2} + h\dot{\omega}^{n+1} \end{cases} \tag{51}$$

$$\begin{cases} \mathbf{d}^{n+2} = \mathbf{d}^{n+1} + h\dot{\mathbf{d}}^{n+3/2} \\ \dot{\mathbf{q}}^{n+3/2} = \mathbf{A}(\omega^{n+3/2})\mathbf{q}^{n+3/2} \\ \mathbf{q}^{n+3/2} = \mathbf{q}^{n+1} + \frac{h}{2}\dot{\mathbf{q}}^{n+3/2} \\ \mathbf{q}^{n+2} = 2\mathbf{q}^{n+3/2} - \mathbf{q}^{n+1} \\ (\mathbf{q}^{n+2})^T \mathbf{q}^{n+2} = 1 \end{cases} \tag{52}$$

Note that we do not use $\mathbf{d}^{n+3/2}$ and $\mathbf{q}^{n+3/2}$ in advancing from the step (n+3/2) to the present step (n+2) in computing $\mathbf{d}^{n+2}$ and $\mathbf{q}^{n+2}$. Instead, we employ $\mathbf{d}^{n+1}$ and $\mathbf{q}^{n+1}$, hence the name *two-stage staggered explicit procedure*[34]. The net result is that, even though we take a full step (h instead of h/2), we only advance half the step at a time. In other words, we evaluate the acceleration and the angular acceleration vectors twice for each full step.

V. Implementation

We will now outline the implementation aspects of the the partitioned MBD solution procedure. The procedure is implemented into two separate integration modules: generalized-coordinate integrator (CINT) and Lagrange multiplier solver (LINT). The generalized-coordinate integrator employs a two-stage modified form of the central difference method for updating the angular velocity vector and the mid-point implicit rule for updating the angular orientations via the Euler parameters. The Lagrange multipliers solver adopts a staggered form of the mid-point implicit method.

A. Generalized-Coordinate Integrator (CINT)

The module receives $f_\lambda^n = \mathbf{B}^T\boldsymbol{\lambda}^n$ from LINT and advances the solution of the MBD equation (1) from time $\mathbf{t}^n$ to $\mathbf{t}^{n+1}$. At each integration step, CINT performs the following computations.

Given: $\mathbf{p}^n = (\dot{\mathbf{d}}^{n-1/2},\ \mathbf{d}^n,\ \omega^{n-1/2},\ \mathbf{q}^n)$ and $\mathbf{g}^n = (\omega^n, f_\lambda^n = \mathbf{B}^T\boldsymbol{\lambda}^n))$

Compute: $\ddot{\mathbf{d}}^n$ and $\dot{\omega}^n$ by (41)

Advance:

$$\begin{cases} \dot{\mathbf{d}}^{n+1/2} = \dot{\mathbf{d}}^{n-1/2} + h\ddot{\mathbf{d}}^n \\ \mathbf{d}^{n+1} = \mathbf{d}^n + h\dot{\mathbf{d}}^{n+1/2} \end{cases} \tag{53}$$

$$\begin{cases} \omega^{n+1/2} = \omega^{n-1/2} + h\dot{\omega}^n \\ \tilde{\mathbf{q}}^{n+1/2} = \frac{1}{\Delta}[\mathbf{I} + \frac{h}{2}\mathbf{A}(\omega^{n+1/2})] \cdot \mathbf{q}^n, \quad \Delta = 1 + \frac{h^2}{4}(\omega_1^2 + \omega_2^2 + \omega_3^2) \\ \mathbf{q}^{n+1} = 2\tilde{\mathbf{q}}^{n+1/2} - \mathbf{q}^n, \quad (\mathbf{q}^{n+1})^T \cdot \mathbf{q}^{n+1} = 1 \end{cases} \tag{54}$$

Output: $\mathbf{p}^{n+1} = (\dot{\mathbf{d}}^{n+1/2}, \mathbf{d}^{n+1}, \omega^{n+1/2}, \mathbf{q}^{n+1})$

Module Invoke: Call CINT $(\mathbf{p}^n, \mathbf{g}^n, h, \mathbf{p}^{n+1})$

where h is the stepsize and $\mathbf{A}(\omega)$ is given by

$$\mathbf{A}(\omega) = \frac{1}{2}\begin{bmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0 \end{bmatrix} \tag{55}$$

and $\tilde{\mathbf{q}}^{n+1/2}$ is an intermediate vector and (54c) must be solved to obtain $\mathbf{q}^{n+1}$ so as to to satisfy the linear dependency constraint, $\mathbf{q}^T\mathbf{q} = 1$.

B. Lagrange Multiplier Solver (LINT)

This module receives $(\dot{\mathbf{d}}, \mathbf{d}, \omega, \mathbf{q})$ from CINT and performs the following computations.

Given: $\ell^{n+1/2} = (\dot{\mathbf{d}}^{n+1/2}, \mathbf{d}^{n+1/2}, \omega^{n+1/2}, \mathbf{q}^{n+1/2}, \boldsymbol{\lambda}^n)$

Compute: $\mathbf{B}^{n+1/2}$, $\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T$ and $\mathbf{r}_\lambda^{n+1/2}$ by (3) and (4)

Advance:

$$\begin{cases} \tilde{\boldsymbol{\lambda}}^{n+1/4} = (\epsilon\mathbf{I} + \frac{h}{4}\mathbf{B}M^{-1}B^T)^{-1}(\epsilon\boldsymbol{\lambda}^n + \frac{h}{4}(\mathbf{r}_\lambda^n + \mathbf{r}_\lambda^{n+1/2})) \\ \boldsymbol{\lambda}^{n+1/2} = 2\tilde{\boldsymbol{\lambda}}^{n+1/4} - \boldsymbol{\lambda}^n \\ \mathbf{f}_\lambda^{n+1/2} = (\mathbf{B}^{n+1/2})^T \cdot \boldsymbol{\lambda}^{n+1/2} \end{cases} \tag{56}$$

Output: $\boldsymbol{\lambda}^{n+1/2}$, $\mathbf{f}_\lambda^{n+1/2}$

Module Invoke: Call LINT $(\ell^{n+1/2}, h, \boldsymbol{\lambda}^{n+1/2}, \mathbf{f}_\lambda^{n+1/2})$

C. Two-Stage Explicit-Implicit Staggered Procedure

In order to evaluate $\dot{\omega}^{n+1}$, ω^{n+1} must be known. Notice from the preceding section that only $\dot{\omega}^{n+1/2}$ is available. Because inaccurate treatments of the gyroscopic damping and the centrifugal force terms can lead quickly to computational instability in computing $\dot{\omega}^{n+1}$, it is not advisable to obtain ω^{n+1} by extrapolating with $\omega^{n+1/2}$ and $\omega^{n-1/2}$. To mitigate

this difficulty, we advance only to the next half step, at each CINT and LINT call. This is illustrated as follows:

$t = t^n$
Call CINT ($\mathbf{p}^n$, $\mathbf{g}^n$, h, $\mathbf{p}^{n+1}$)
Call LINT ($\ell^{n+1/2}$, h, $\boldsymbol{\lambda}^{n+1/2}$, $\mathbf{f}_\lambda^{n+1/2}$)
$t = t^n + h/2 \quad (n \leftarrow n + 1/2)$
Call CINT ($\mathbf{p}^{n+1/2}$, $\mathbf{g}^{n+1/2}$, h, $\mathbf{p}^{n+3/2}$)
Call LINT (ℓ^{n+1}, h, $\boldsymbol{\lambda}^{n+1}$, $\mathbf{f}_\lambda^{n+1}$)
$t = t^n + h$

Note that

$$\mathbf{g}^{n+1/2} = (\boldsymbol{\omega}^{n+1/2},\ f_\lambda^{n+1/2})$$

together with

$$\mathbf{p}^{n+1/2} = (\dot{\mathbf{d}}^n,\ \mathbf{d}^{n+1/2},\ \boldsymbol{\omega}^n,\ \mathbf{q}^{n+1/2})$$

provides the necessary input data to compute $\ddot{\mathbf{d}}^{n+1/2}$ and $\dot{\boldsymbol{\omega}}^{n+1/2}$ in the second call of CINT in the above calling sequence. In summary, the present procedure requires two function evaluations and two λ–solutions per each full step, hence the name "two-stage explicit-implicit staggered procedure".

VI. Numerical Examples

The two modules, the generalized coordinate integrator (CINT) and the Lagrange multipliers solver (LINT), have been implemented in Fortran 77. In solving the following three example problems, we have incorporated the constraint conditions through the use of Lagrange multipliers instead of eliminating the constraints. It is therefore necessary to solve the governing equations of motion in a way that satisfies the constraint equations. Hence, efficient and accurate solutions of these problems will confirm not only the viability of the present integration procedure for the solution of the multibody equations of motion with or without constraints but also the constraint stabilization procedure in their combined totality.

A. Plane Three-Link Manipulator

The first problem tested is a simplified version of the seven-link manipulator deployment problem[52]. The three links are initially folded and, for modeling simplicity, between the two joints is a coil spring which resists a constant deploying force at the tip of the third link. Also, the left-hand end of the first link is fixed through the same coil spring to the wall. These three coil springs are to be *locked up* once the links are deployed straight. The deployment sequence of the manipulator is illustrated in Fig. 1. The time-discretized difference equations both for Baumgarte's technique and the staggered stabilization technique have been solved at each time increment by a Newton-type iterative procedure to meet

a specified accuracy level. Hence, the performance of the two techniques can be assessed by the average number of iterations taken per time increment. This is presented in Fig. 2 for the accuracy of 10^{-4}. Notice that the staggered stabilization technique requires on the average about 4.5 iterations per step, whereas Baumgarte's technique requires about 22 iterations per step.

Note that Baumgarte's technique fails to converge for time, $t \approx 1.1$ as manifested in Fig. 2 because the rows in $\mathbf{B}$ become numerically dependent upon one another when the links are in a straight configuration. This corroborates the theoretical prediction of non-convergence whenever the solution matrix, $\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T$, for Baumgarte's technique (see Eqs.(5b), (20) and (21)) becomes singular. On the other hand, the staggered stabilization technique still converges within 30 iterations, because it overcomes this singularity difficulty, since $\dot{\boldsymbol{\lambda}}$ still exists, as can be seen from Eqs. (27) and (32).

It should be noted that, in order to avoid such ill-conditioning, one must differentiate the constraint equations once or twice more and recast the resulting higher-order constraint equations in terms of first-order equations with increased number of equations. This process is known as an index reduction strategy[46]. Thus, one must restructure the augmented equations of motion (5) with the net result of increased solution variables. Other techniques involve singular value decompositions, e.g., as advocated by Führer and Leimkuhler[53]. On the other hand, the present staggered stabilization technique overcomes the ill-conditioning difficulty without restructuring the governing equations of motion. Instead, the constraint equations are enforced in a separate module by the parabolically regularized equations for the Lagrange multipliers as derived in (27) and (32).

Although not reported here, the same relative performance has been observed for different accuracy levels, i.e., for the accuracy of 10^{-5} and 10^{-6}.

From this test problem, we conclude that the staggered stabilization technique yields both improved accuracy over and greater computational robustness than the Baumgarte technique. In addition, the staggered stabilization technique offers software modularity in that the solution of the constraint force, λ, can be carried out separately from that of the generalized displacement, $\mathbf{q}$. The only data each solution module needs to exchange with the other is a set of vectors, plus a common module to generate the gradient matrix of the constraints, $\mathbf{B}$. However, one should be cautioned not to extrapolate blindly to complex problems the results of the present simple examples. Further judicious experiments are needed in applying the present staggered stabilization technique to complex production-level problems before it can be adopted for general applications in multibody dynamic simulations.

B. Three-Dimensional Double Pendulum

The second problem with which we have tested the present procedure is a spatially moving double pendulum as shown in Fig. 3. The governing equations of motion become those of two separate rigid bars, except they are connected by two spherical joints. From Fig. 3

we have the the following quantities:

$$\Phi^i = \dot{\mathbf{d}}^i - \frac{1}{2}\omega^i \times \mathbf{z}^i = 0, \quad i = 1,\ 2. \tag{57}$$

$$\mathbf{M} = \text{diag}\{\mathbf{m}^1,\ \mathbf{J}^1,\ \mathbf{m}^2,\ \mathbf{J}^2\} \tag{58}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{I} & \frac{1}{2}\tilde{\mathbf{z}}^1\times & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & -\frac{1}{2}\tilde{\mathbf{z}}^1\times & -\mathbf{I} & -\frac{1}{2}\tilde{\mathbf{z}}^2\times \end{bmatrix} \tag{59}$$

$$\mathbf{F}\omega = \begin{Bmatrix} f^1 \\ f^2 \end{Bmatrix}, \qquad f^i = -\begin{bmatrix} 0 \\ 0 \\ 0 \\ \omega_2\omega_3(J_2 - J_3) \\ \omega_3\omega_1(J_3 - J_1) \\ \omega_1\omega_2(J_1 - J_2) \end{bmatrix}^i, \quad i = 1,\ 2. \tag{60}$$

$$\ddot{u}^i = \{\ddot{d},\ \dot{\omega}\}^i, \qquad \ddot{d}^i = [\ddot{x},\ \ddot{y},\ \ddot{z}]^T, \qquad \dot{\omega}^i = [\dot{\omega}_1,\ \dot{\omega}_2,\ \dot{\omega}_3]^T \tag{61}$$

$$\lambda = [\lambda_1,\ \lambda_2,\ \lambda_3,\ \lambda_4,\ \lambda_5,\ \lambda_6]^T \tag{62}$$

In the preceding equations, $\frac{1}{2}\mathbf{z}$ is the vectorial distance from the center of the bar to the spherical joint constraints, $\mathbf{m}$ and $\mathbf{J}$ are the three translational and rotatory inertia matrices, $\tilde{\mathbf{z}}$ is the skew symmetric matrix formed by the three components of $\mathbf{z}$, $\times$ implies a vector cross multiplication, and the superscript designates the i-th bar.

The pendulum is originally positioned in a gravity field with initial horizontal angular velocities ($\omega_z^{(1)} = \omega_z^{(2)} = 1$). Figure 4 shows the spatial trajectories of the two mass centers as projected on the horizontal surface and on the vertical plane. It is noted that the two trajectories form a similar pattern. The constraint forces and angular velocities, although not reported herein, exhibit patterns that are analogous in their characteristics for the two joints and two mass centers, respectively.

We have performed convergence studies by using different stepsizes h. Numerical evaluations indicate, as with the rigid-link problem, that when the stepsize samples more than 20 per period, the present procedure yields both good accuracy and stability.

C. Open-Loop Torque for Three-Link Manipulator

The third problem is a three-link manipulator maneuvering under a specified nonholonomic tip velocity constraint. For this problem, both rigid links and flexible links with four beam elements per link have been investigated. The flexible beam was modeled with a constant-strain Timoshenko beam element that allows large rotations. The three joints are modeled as spherical ones and the Lagrange multipliers have been introduced to enforce

the joint constraints and well as the nonholonomic constraint at the manipulator tip. The trajectories of the manipulator and the tip velocity specification are shown in Figs. 5 and 6. The corresponding joint torques for the rigid and flexible links are also shown in Figs. 7 and 8, respectively. Note that even though there exists little difference in the two trajectories of the rigid and flexible cases, there are significant differences in the open-loop joint torques. These will play an important role in the design of controller for vibration suppression in the manipulator arms.

VII. Discussions

In this chapter, we have presented a computational procedure for direct integration of the multibody dynamical (MBD) equations with constraints.

Because of its step-advancing nature, the procedure is labeled as a two-stage staggered explicit-implicit algorithm: explicit for solving the generalized coordinates (CINT) and implicit for Lagrange multipliers to incorporate constraints (LINT). Our numerical experiments indicate that it is essential to enforce the linear dependency constraint condition on the Euler parameters at each integration step.

Numerical experiments reported herein and additional applications conducted so far indicate that the present procedure yields robust solutions if the stepsize gives more than twenty samples for the period of the apparent highest response frequency of a given multibody system. Hence, the present procedure appears to have accomplished the following:

- For closed loop multibody systems and/or problems with complex topology wherein it is practically inadvisable to eliminate the constraints, the present procedure facilitates a straightforward construction of the governing equations of motion with appropriate constraints. The generalized coordinates and the system open and closed loop Lagrange multipliers can then be solved by the present procedure in a partitioned manner.
- For problems that involve lock-up mechanisms or similar discontinuities, the present procedure appears to overcome numerical difficulties encountered in using the Baumgarte stabilization. This may be an important impetus for applying the present procedure for the simulation of deployment dynamics of space structures.
- The angular velocity is obtained by an adaptation of the central difference algorithm in a two-stage form and the update of angular orientations is based on the Euler parameters by adopting the mid-point implicit formula. Both of the integrators conserve the system energy, which is important when the multibody simulation package is to be interfaced with an active control synthesis module. This is because stability margins of active control systems are sensitive to the system damping characteristics either physical or numerical.
- The present MBD solution procedure is implemented into two separate modules: the generalized coordinates solver (CINT) and the constraint Lagrange multiplier solver (LINT). Hence, the task for interfacing of the present MBD solution modules with additional capabilities such as active controller, observer and other analysis and design software modules becomes relatively straightforward. Such software architecture is

in contrast to most of the existing programming practice wherein several analysis capabilities are embedded into a single monolithic program.

Applications of the present procedure to flexible multibody systems are currently being carried out and preliminary results are quite encouraging. We hope to report on the results of flexible-body dynamics as well as on large-scale multibody problems in the near future.

Acknowledgements

The work reported herein was supported by NASA/Langley Research Center under Grant NAG-1-756. The authors wish to thank Dr. Jerry Housner for his keen interest and encouragement during the course of the present work.

References

1. Hooker, W. and Margulies, G., "The Dynamical Attitude Equations for an N-body Satellite," J. Astronautical Science, Vol. 12, 1965, pp. 123-12.

2. Roberson, R. and Wittenburg, J., "A Dynamical Formalism for an Arbitrary Number of Interconnected Rigid Bodies with Reference to the Problem of Satellite Attitude Control," Proc. the Third Int. Congress of Automatic Control, Butterworth, London, 1965.

3. Roberson, R., "A Form of the Translational Dynamical Equations for Relative Motion in Systems of Many Non-Rigid Bodies," Acta Mech. Vol. 14, 1972, pp. 297-308.

4. Huston, R. L. and Passerello, C. E., "On Lagrange's Form of d'Alembert's Principle," The Matrix and Tensor Quarterly, Vol. 23, 1973, pp. 109-112.

5. Boland, P., Samin, J. and Willems, P., "Stability Analysis of Interconnected Deformable Bodies in a Topological Tree," AIAA J., Vol. 12, 1974, pp. 1025-1030.

6. Likins, P., "Analytical Dynamics and Nonrigid Spacecraft Simulation," Jet Propulsion Laboratory, Technical Report 32-1593, Pasadena, Ca., 1974.

7. De Veubeke, B. F., "The Dynamics of Flexible Bodies," Int. J. Engng. Sci., Vol. 14, 1976, pp. 895-913.

8. Jerkovsky, W., "The Transformation Operator Approach to Multisystem Dynamics, Part I: The General Approach," The Matrix and Tensor Quarterly, Vol. 27, 1976, pp. 48-59.

9. Ho, J. Y. L., "Direct Path Method for Flexible Multibody Spacecraft Dynamics," Journal of Spacecraft and Rockets, Vol. 14, No. 2, 1977, pp. 102-110.

10. Wittenburg, J., Dynamics of Systems of Rigid Bodies, B. G. Teubner, Stuttgart, 1977.

11. W. W. Armstrong, "Recursive Solution to the Equations of Motion of an N-Link Manipulator," Proc. 5th World Congress, Theory of Machines, Mechanisms, Vol. 2, 1979, pp. 1343-1346.

12. Hollerbach, J., M., "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity," IEEE Trans on Systems, Man, and Cybernetics, SMC-10, 1980, pp. 730-736.

13. Kane, T. and Levinson, D., "Formulation of Equations of Motion for Complex Spacecraft," J. Guidance and Control, Vol. 3, 1980, pp. 99-112.

14. Keat, J. E., "Dynamical Equations of Body Systems with Applications Space Structure Deployment", PhD Thesis, MIT, 1983.

15. Haug, E. J. (ed.), Computer Aided Analysis and Optimization of Mechanical System Dynamics, Springer-Verlag, Berlin 1984.

16. Roberson, R. E. and Schwertassek, R., Dynamics of Multibody Systems, Springer-Verlag, New York, 1984.

17. Schwertassek, R. and Roberson, R. E., "A State-Space Dynamical Representation for Multibody Mechanical Systems, Part II," Acta Mechanica, Vol. 51, 1984, pp. 15-29.

18. Bianchi, G. and Schielen, W. (eds), Dynamics and Multibody Systems, Springer-Verlag, Berlin, Heidelberg, 1986.

19. Huston, R. L., Lecture Notes on Dynamics, Preprint, University of Cincinnati, 1988.

20. Walton, W. C. and Steeves, E. C., "A New Matrix Theorem and Its Application for Establishing Independent Coordinates for Complex Dynamical Systems with Constraints," NASA TR-R326, 1969.

21. Gear, C. W., "Simultaneous Numerical Solution of Differential/Alge-braic Equations," IEEE Trans. Circuit Theory, CT-18, 1971, pp. 89-95.

22. Baumgarte, J. W., "Stabilization of Constraints and Integrals of Motion in Dynamical Systems," Comp. Meth. Appl. Mech. Engr., Vol. 1, 1972, pp. 1-16.

23. Orlandea, N., Chase, M. A. and Calahan, D. A., "A Sparsity-Oriented Approach to the Dynamic Analysis and Design of Mechanical Systems - Part I and II," Trans. ASME, J. Eng. for Industry, Ser. B, Vol. 99, 1977, pp. 773-784.

24. Lötstedt, P., "On a Penalty Function Method for the Simulation of Mechanical Systems Subject to Constraints," Royal Institute of Technology, TRITA-NA-7919, Stockholm, Sweden, 1979.

25. Orin, D. E., D. E., McGhee, R. B., Vukobratovic, M. and Hartoch, G., "Kinematic and Kinetic Analysis of Open-chain Linkages Utilizing Newton-Euler Methods," Math. Biosc., Vol. 43, 1979, pp. 106-130.

26. Huston, R. L. and Kamman, J. W., "A Discussion on Constraint Equations in Multibody Dynamics," Mech. Res. Comm., Vol. 9, 1982, pp. 251-256.

27. Petzold, L., "Differential/Algebraic Equations are not ODEs," SIAM J. Sci. Stat. Comp., Vol. 3, 1982, pp. 367-384.

28. Wehage, R. A. and Haug, E. J., "Generalized Coordinate Partitioning for Dimension Reduction in Analysis of Constrained Dynamic Systems," ASME J. of Mech. Design, Vol. 104, 1982, pp. 247-255.

29. Baumgarte, J. W., "A New Method of Stabilization for Holonomic Constraints," Journal of Applied Mechanics, Vol. 50, 1983, pp. 869-870.

30. Führer, C. and Wallrapp, O., "A Computer-Oriented Method for Reducing Linearized Multibody System Equations by Incorporating Constraints," Comp. Meth. Appl. Mech. Engr., Vol. 46, 1984, pp. 169-175.

31. Nikravesh, P. E., "Some Methods for Dynamic Analysis of Constrained Mechanical Systems: a Survey," in: Computer Aided Analysis and Optimization of Mechanical System Dynamics (E. J. Haug, ed.), NATO ASI series, F9, Springer-Verlag, Berlin, 1984, pp. 351-367.

32. Enright, W. H., "Numerical Methods for Systems of Initial Value Problems - The State of the Art," in: Computer Aided Analysis and Optimization of Mechanical System Dynamics(E. J. Haug, ed.), NATO ASI series, F9, Springer-Verlag, Berlin, 1984, pp. 309-322.

33. Garcia de Jalon, J., Unda, J., Avello, A. and Jimenez, J. M., "Dynamic Analysis of Three-Dimensional Mechanisms in Natural Coordinates," Journal of Mechanisms, Transmissions and Automation in Design, Vol. 109, 1987, pp. 460-465.

34. Park, K.C., Chiou, J.C. and J. D. Downer, "A Computational Procedure for Large Rotational Motions in Multibody Dynamics ," Proc. the 29th Structures, Dynamics and Materials Conference, AIAA Paper No. 88-2416, AIAA, 1988, pp.1593-1601 (also to appear in J. Guidance, Control and Dynamics).

35. Park, K. C. and Chiou, J. C., "Stabilization of Computational Procedures for Constrained Dynamical Systems," Journal of Guidance, Control and Dynamics, Vol. 11, July-August 1988, pp. 365-370.

36. Geradin, M. and Cardona, A., "Kinematic and Dynamics of Rigid and Flexible Mechanisms Using Finite Elements and Quaternion Algebra," Computational Mechanics, Vol. 4, 1989, pp. 115-135.

37. Bodley, C. S., Devers, A. D., Park, A. C. and Frish, H. P., "A Digital Computer Program for the Dynamic Interaction Simulation of Controls and Structures (DISCOS)," NASA Technical Paper 1219, 1978.

38. The ADAMS User's Guide, Mechanical Dynamics, Inc., Ann Arbor, Mich., 1979.

39. Schwertassek, R., "Der Roberson/Wittenburg Formalismus and das Programmsystem MULTIBODY zur Rechnersimulation von Mehrköpersystemen," Report DFVLR-FB-78/08, DFVLR, Köln, 1978.

40. Huston, R. L., Harlow, M. W. and Gausewitz, N. L., "User's Mannual for UCIN-EULER - A Multipurpose, Multibody Systems Dynamics Computer Program," NTIS Report AD-A120403, 1982.

41. Haug, E. J., Lance, G. M., Nikravesh, P. E., Vanderploeg, M. J. and Wehage, R. A., DADS (Dynamic Analysis and Design Systems), Computer Aided Design Software Inc., Oakdale, Iowa, 1985.

42. Housner, J. M., McGowan, P. E., Abrahamson, A. L. and Powel, M. G., "The LAT-DYN User's Mannual," NASA TM 87635, NASA/Langley Research Center, January 1986.

43. Hughes, T. J. R. and Belytschko, T., "A Prècis of Developments in Computational Methods for Transient Analysis," Journal of Applied Mechanics, **50**, 1983, 1033-1041.

44. Park, K. C., "Transient Analysis Methods in Computational Methods," *Finite Elements: Theory and Applications* (ed. D. L. Dwoyer, M. Y. Hussaini and R. G. Voigt), Springer-Verlag, 1988, 240-267.

45. Belytschko, T., Englemann, B. E. and Liu, W. K., "A Review of Recent Developments in Time Integration," in: State-of-the-Art Surveys on Computational Mechanics (Noor, A. K. and Oden, J. T., editors), ASME, 1989, pp. 185-200.

46. Brenan, K. E., Campbell, S. L. and Petzold, L. R., *The Numerical Solution of Initial Value Problems in Ordinary Differential-Algebraic Equations*, Elsevier Science Publishing Co., 1989.

47. Park, K. C., "Partitioned Analysis Procedures for Coupled-Field Problems: Stability Analysis," Journal of Applied Mechanics, Vol. 47, 1980, pp. 370-378.

48. Felippa, C. A. and Park, K. C., 'Staggered Transient Analysis Procedures for Coupled Mechanical Systems," Computer Methods in Applied Mechanics and Engineering, Vol. 24, 1980, pp. 61-111.

49. Park, K. C. and Felippa, C. A., "Partitioned Analysis of Coupled Systems," in Computational Methods for Transient Analysis, T. Belytschko and T. J. R. Hughes (eds.), Elsevier Pub. Co., 1983, pp. 157-219.

50. Felippa, C. A. and Park, K. C., "Computational Aspects of Time Integration Procedures in Structural Dynamics, Part 1: Implementation," Journal of Applied Mechanics, Vol. 45, 1978, pp. 595-602.

51. Lanczos, L., The Variational Principles of Mechanics, 4th ed., University of Toronto Press, 1970, pp. 141-147.

52. Housner, J. M., "Convected Transient Analysis for Large Space Structure Maneuver and Deployment," AIAA-84-1023-CP, *Proc. 25th Structures, Structural Dynamics and Material Conference*, Part 2, 14-16 May 1984, Palm Springs, pp. 616-619.

53. Führer, C. and Leimkuhler, B., "Formulation and Numerical Solution of the Equations of Constrained Mechanical Motion," Technical Report DFVLR-FB 89-08, DFVLR, D-5000 Köln 90, 1989.

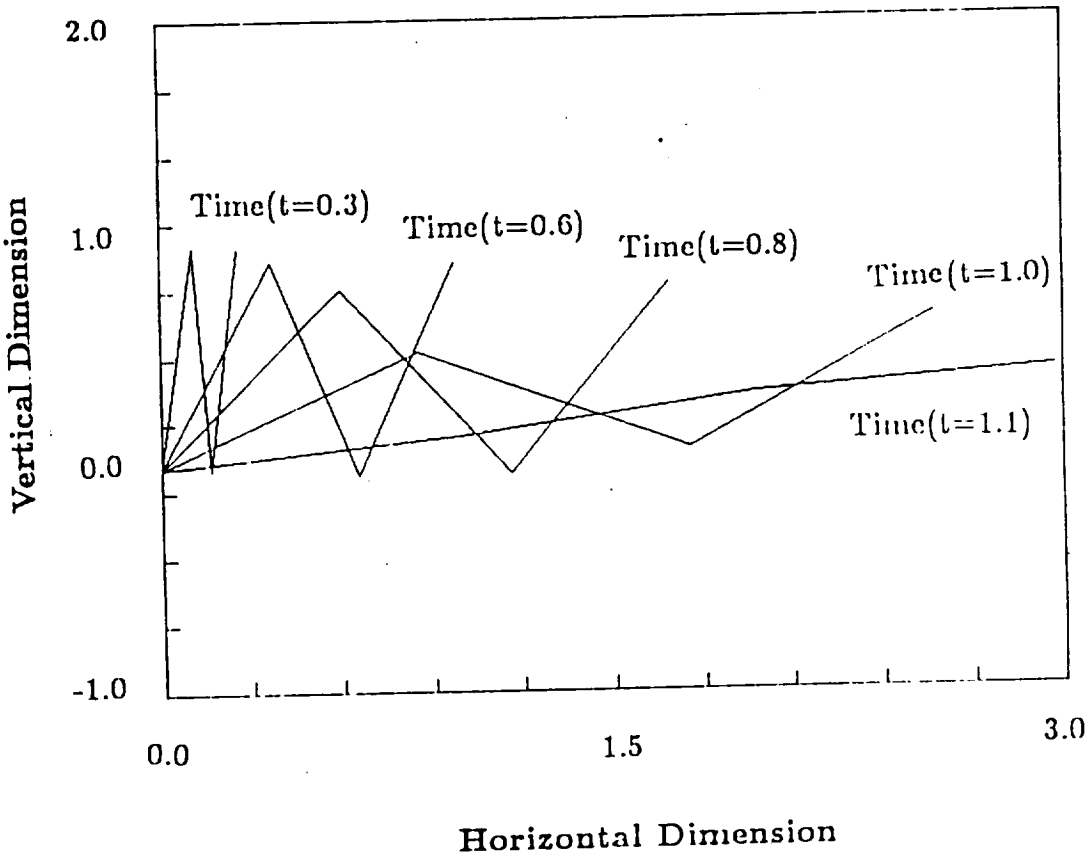
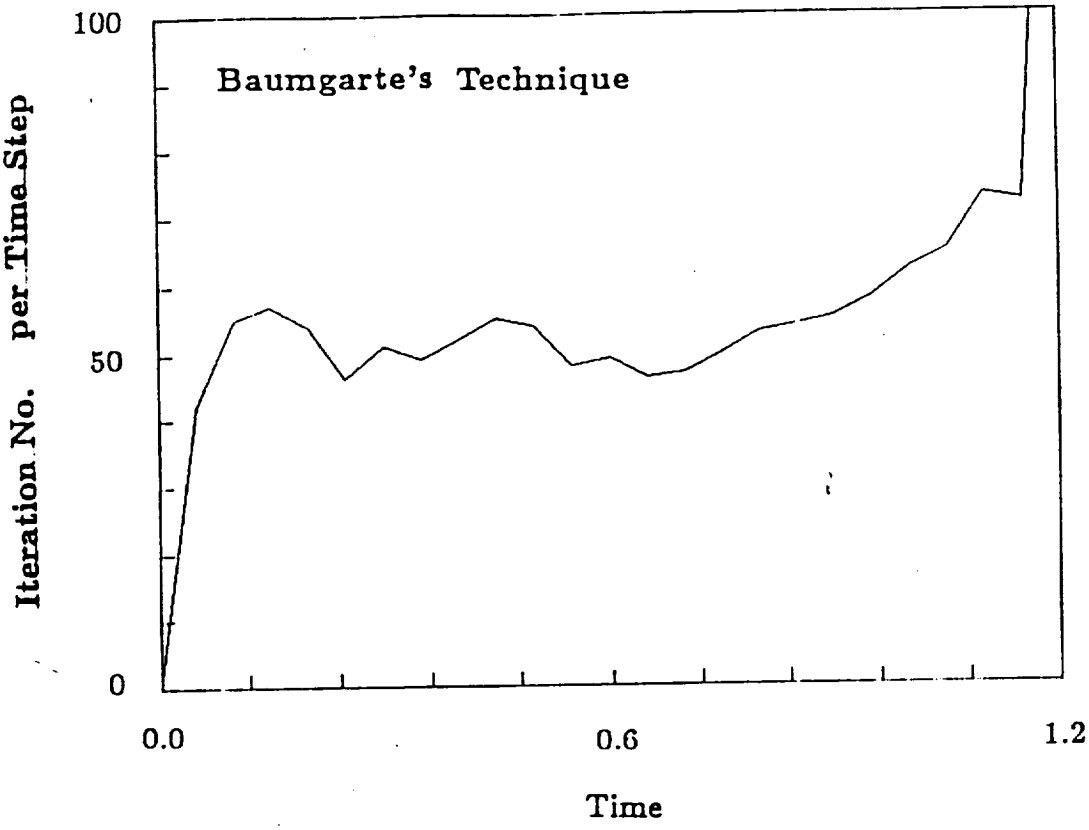


Fig. 1 Deployment of Three-Link Remote Manipulator



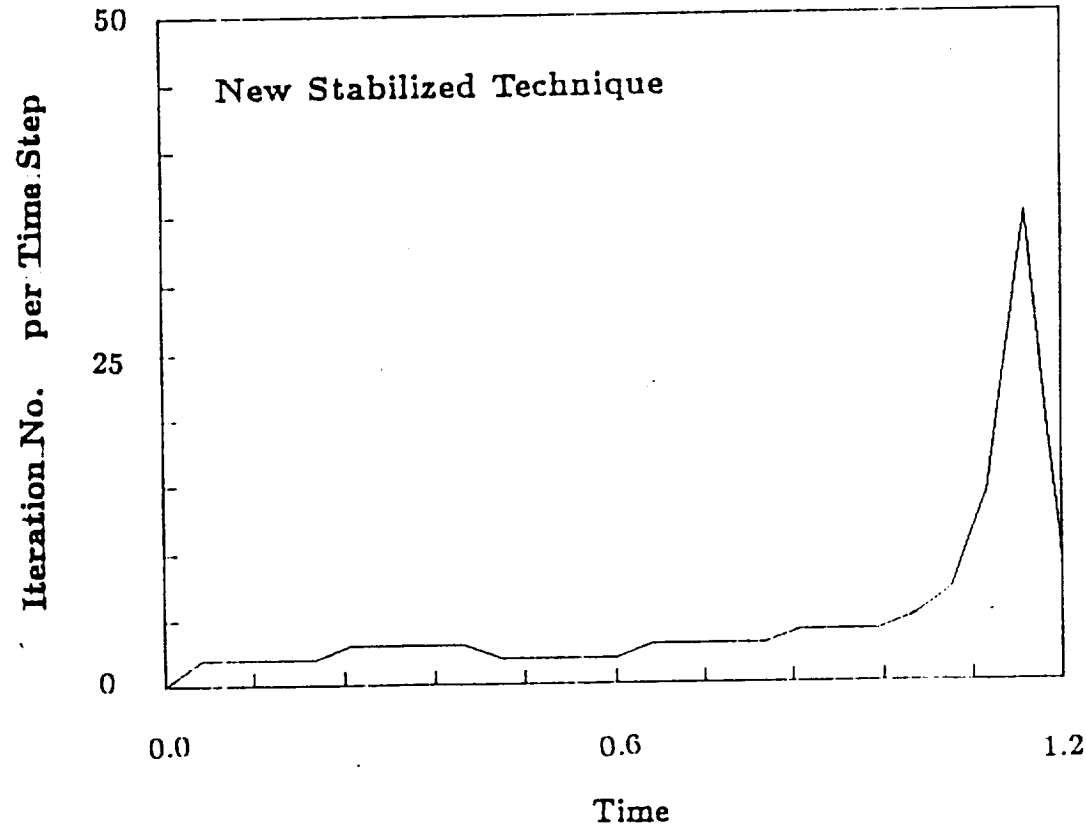
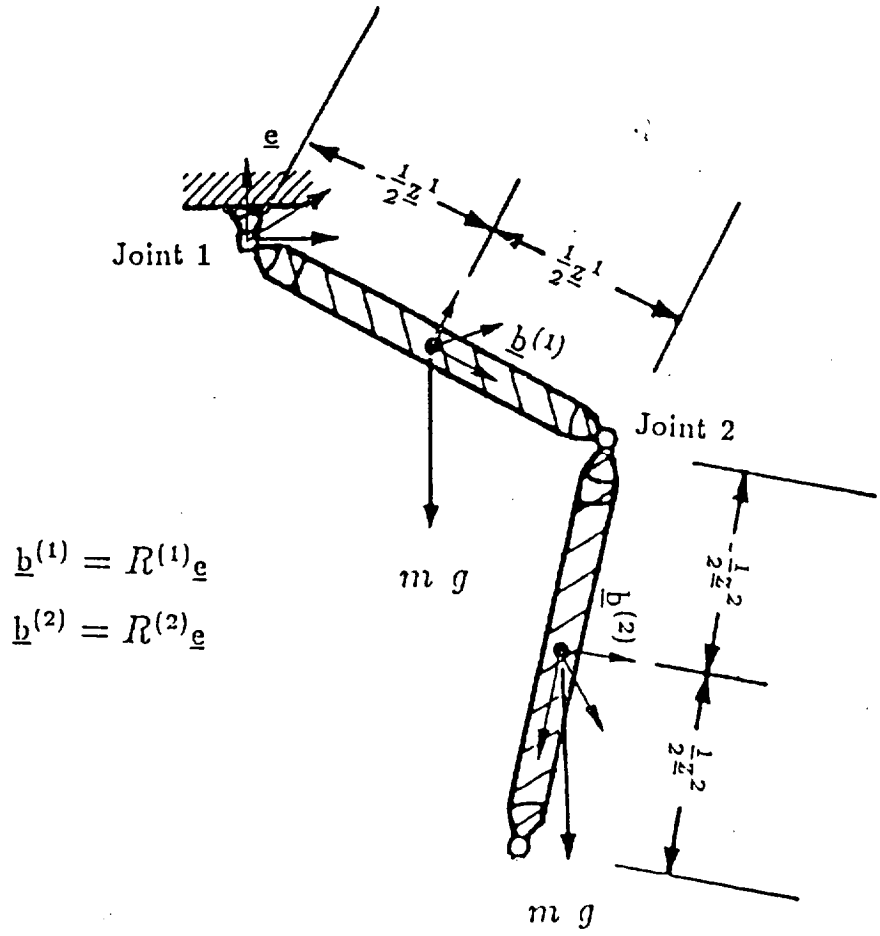


Fig. 2 Performance of Two Stabilization Techniques for Three-Link Remote Manipulator (Solution Accuracy=10^{-6})



$$\underline{b}^{(1)} = R^{(1)}\underline{e}$$

$$\underline{b}^{(2)} = R^{(2)}\underline{e}$$

Fig 3 Double Pendulum with Spatial Joints

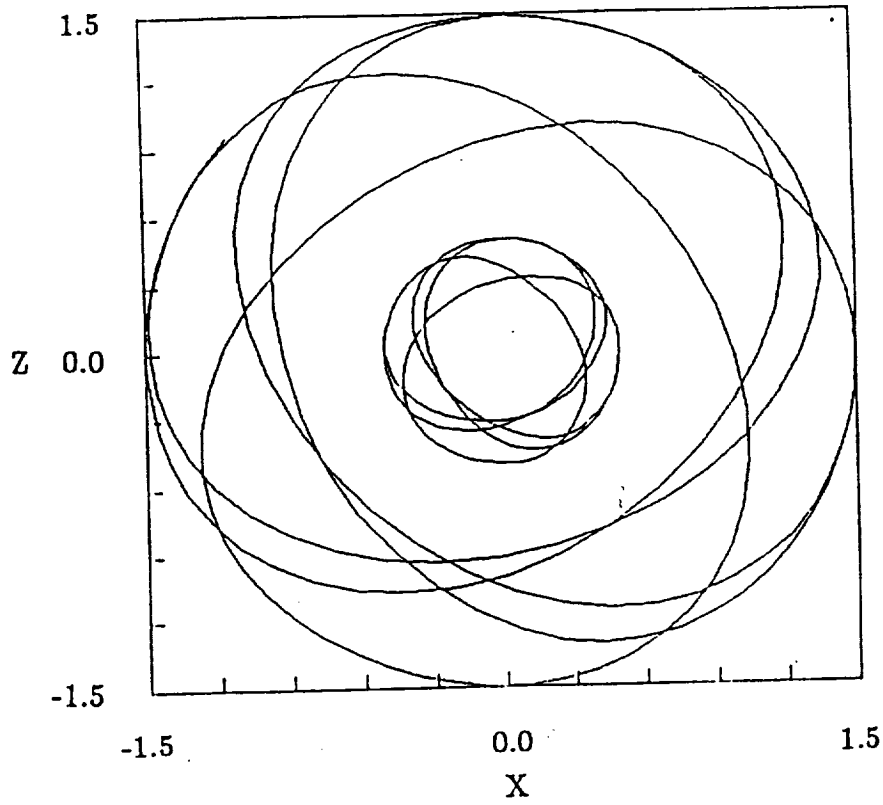


Fig. 4a Trajectories of double pendulum on X-Z plane

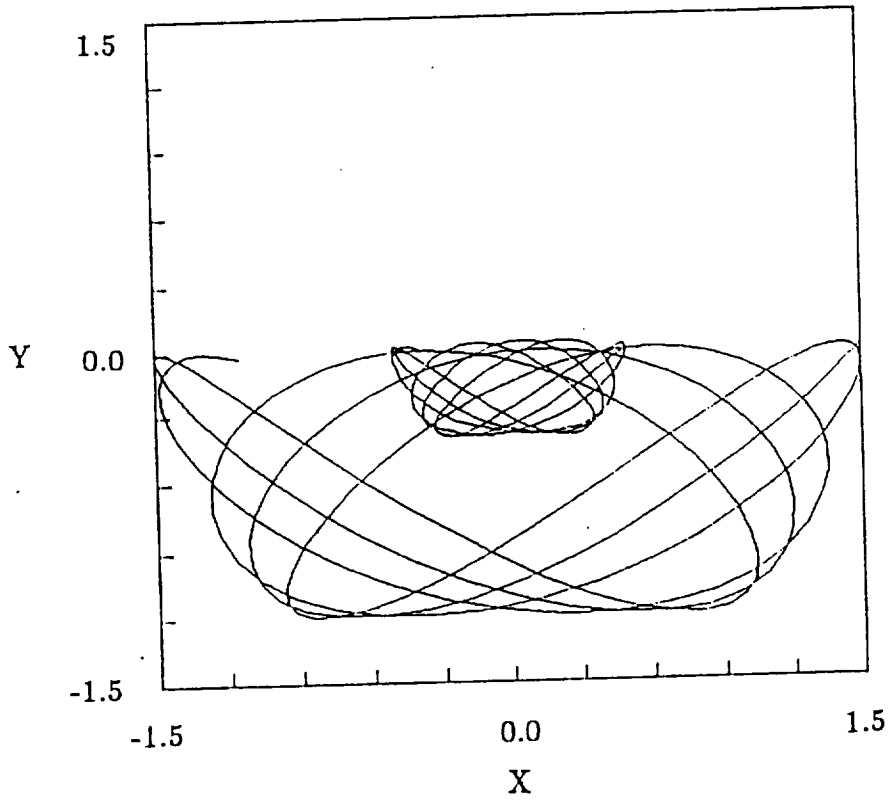


Fig. 4b Trajectories of double pendulum on X-Y plane

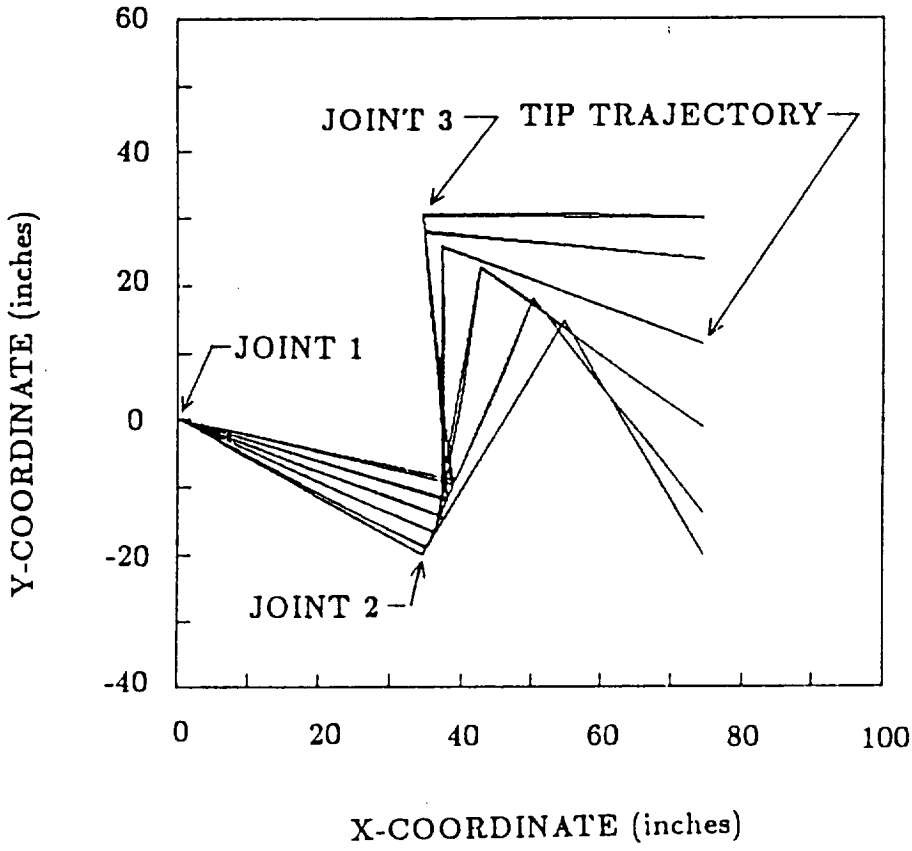


Fig. 5 Crane Tip Trajectory of Rigid and Flexible Members

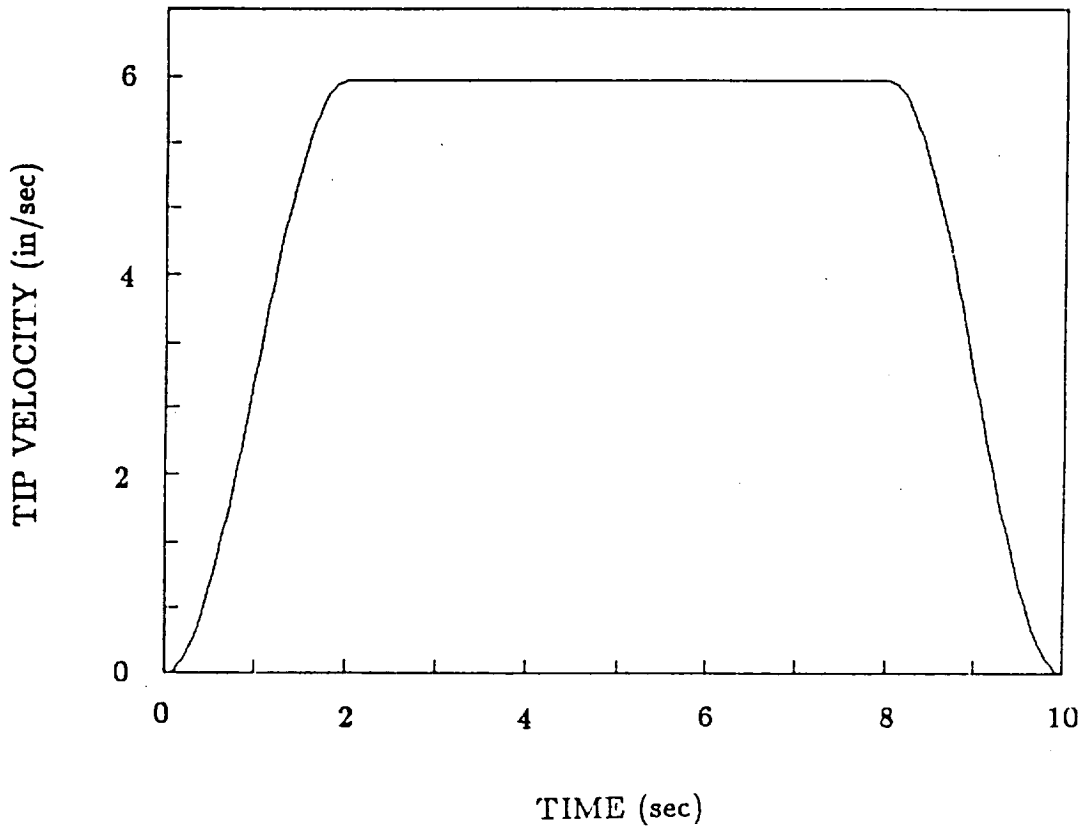


Fig. 6 Crane Tip Velocity of Rigid and Flexible Members

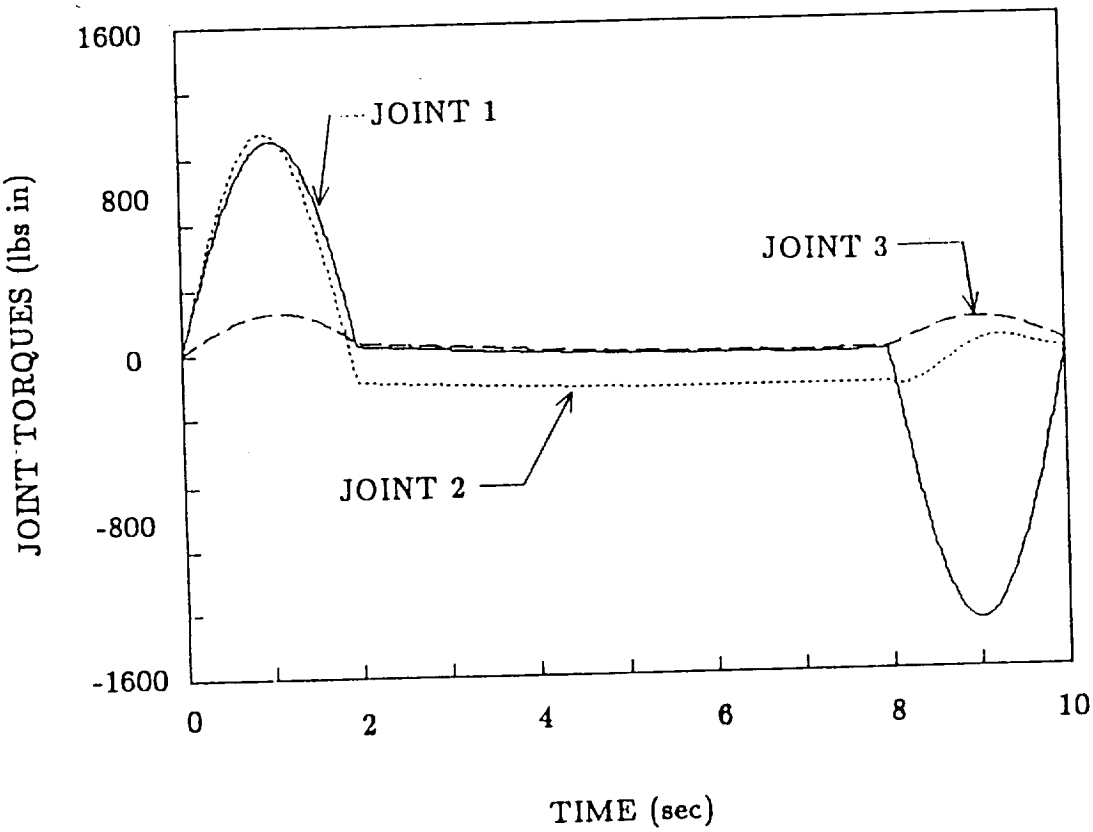


Fig. 7 Crane Joint Torque (Rigid Members) vs. Time

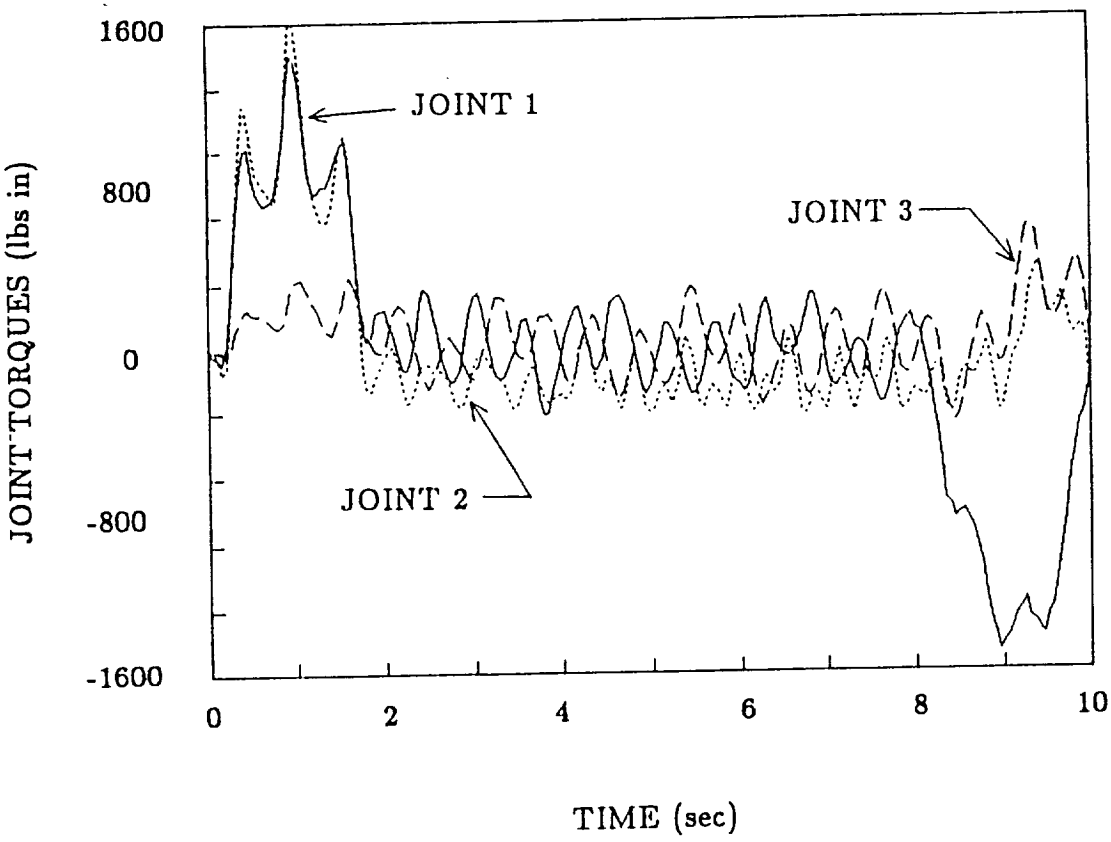


Fig. 8 Crane Joint Torque (Flexible Members) vs. Time

A Computational Procedure for Multibody Systems Including Flexible Beam Dynamics

J. D. Downer, K. C. Park, and J. C. Chiou
Department of Aerospace Engineering Sciences
and Center for Space Structures and Controls
University of Colorado at Boulder
Boulder, CO 80309-0429, USA

Abstract

A computational procedure suitable for the solution of equations of motions for flexible multibody systems has been developed. The flexible beams are modeled using a fully nonlinear theory which accounts for both finite rotations and large deformations. The present formulation incorporates physical measures of conjugate Cauchy stress and covariant strain increments. As a consequence, the beam model can easily be interfaced with real-time strain measurements and feedback control systems. A distinct feature of the present work is the computational preservation of total energy for undamped systems; this is obtained via an objective strain increment/stress update procedure combined with an energy-conserving time integration algorithm which contains an accurate update of angular orientations. The procedure is demonstrated via several example problems.

1. Introduction

The simulation of flexible multibody systems is becoming an increasingly important tool for the design and operation of many engineering applications. Typical examples of such systems include deployable space structures, high precision machine dynamics and robotics, and other problems containing controlled positioning of structural components. The components of these articulated structures typically undergo large relative displacements and rotations in order to carry out the intended operations. To perform the desired kinematic motions, various types of mechanical joints are introduced to constrain the relative motion between the various components. New technology needs of both the space and robotics industries have increased the demand for accurate numerical simulations of the effect of component flexibility on the performance of multibody systems. A significant coupling between the gross structural motion and the elastic deformation can be experienced by typical applications in which lightweight structures with higher flexibility are required to operate with greater positioning accuracy and at higher speeds. To capture this phenomenon, a realistic mathematical model of the structural component that can readily be incorporated into a general multibody dynamics methodology is necessary.

Two basic approaches, the floating frame approach and the nonlinear continuum approach, exist for the modeling of flexible components within a general multibody system.

The floating frame approach introduces a moving reference frame to follow some overall mean rigid body motion of the beam; the elastic deformation of the beam is then described relative to this moving reference[1-6]. With this approach, the classical multi-rigid body analysis was extended to include structural flexibility by superposing existing linear deformation descriptions onto the rigid motions of the floating reference frame[7,8]. The definition of such a mean axis system and the corresponding deformation modes within the general context of the finite element method has been presented[9-11]. To minimize the number of elastic coordinates, coordinate transformations from the physical elastic coordinates to modal coordinates were performed within the multibody dynamics context [12], and static correction modes were used in conjunction with the normal modes of vibration to account for reaction forces and torques transmitted to the components through joint connections[13,14]. An alternative choice of a floating reference frame for finite element applications, termed the convected coordinate system, was introduced as a simple separation of the rigid body motion and the structural deformation for a given finite element[15-18]. All of these studies, however, are limited by the assumption of linear deformation theory which is inadequate to capture certain nonlinear phenomena. Nonlinear deformation theories must be taken into account for such instances as the geometric stiffening of a spinning beam[19,20] in which the structural component experiences a centrifugal force as well as applications in which the components necessarily have low mass and very high flexibility. Extensions of the original approach to model the nonlinear effects include the substructuring technique in which the component is further partitioned into substructures each with a local reference frame where normal vibration and static correction modes can then be used to model the deformation[21], and the finite element incorporation of a nonlinear Green strain measure[22,23]. The resulting equations of motion of the floating frame approach, written in terms of a set of reference coordinates and a set of relative elastic coordinates, inherently contain a complex coupling of the gross motion and the elastic deformation modes.

Recently, a fully nonlinear continuum approach to describe the dynamics of the flexible beam has been pursued[24-28]. Through the use of finite-deformation rod theories[29-32], the approach is capable of directly accounting for both finite rotation kinematics and large deformations of the beam component. Since the motion due to rigid rotations of the beam is not distinguished from that due to deformations, the need for a floating reference frame is completely obviated and the component inertia is identical in form to that of a rigid body. The inherent nonlinear coupling between the gross body motion and the elastic deformation is transferred to the stiffness part of the equations of motion. The key to the successful adoption of this approach is to develop a computational procedure for the nonlinear internal force term that preserves rigid body motions.

The aim of this paper is to incorporate the nonlinear continuum formulation of the spatial beam motion into a general multibody dynamics software methodology. The present formulation employs a convected coordinate representation of physical Cauchy stresses and corresponding set of physical strains. This representation naturally lends itself to the "software in the real-time experiment" loop as sensors measure only physical quantities.

Another advantage of the formulation is that the degrees of freedom of the beam component embody both the rigid and flexible deformation motions. The task for incorporating the multibody system constraints becomes straightforward, and the equations of motion for an arbitrary configuration of flexible beams and rigid bodies can automatically be generated in terms of an identical set of physical coordinates. Numerical solution procedures for the integration of spatial kinematic systems can then be directly applied to these physical coordinates. Such a universal treatment is not applicable within the context of the floating frame approach as the reference and elastic coordinate definitions are of highly different character.

The rest of the paper will be organized as follows. Section 2 will detail the kinematic description of the continuum beam in which the total motion is referred directly to the inertial reference frame. The principle of virtual work of a continuum as specialized to the spatial motion of the beam component is detailed in Section 3. The subsequent finite element discretization of the beam component and overall multibody system equations are then presented. Section 4 will summarize the staggered procedure for the integration of multibody dynamic systems. The virtual work expression is used to derive the method of computation of the internal force, and Section 5 will address this algorithmic treatment of the nonlinear stiffness operator. Section 6 will present some example problems demonstrating the software capabilities.

2. Beam Kinematics

The present formulation adopts an inertial reference frame for describing the translational motions and a body-fixed frame for the rotational motions. The consequence of this description is that the translational and rotational variables embody information due to both rigid rotations and deformations of the beam. The configuration of the beam, as shown in Figure 1, is completely characterized using a position vector locating the neutral axis of the beam from the inertial origin and a body-fixed frame representing the orientation of the cross-section with respect to the inertial reference frame. The position vector $\mathbf{r}$ locating an arbitrary particle point on the beam is thus described as

$$\mathbf{r} \;=\; (\, X \,+\, u \,)^T \, \mathbf{e} \;+\; \ell^T \, \mathbf{b} \tag{2.1}$$

where "boldface" symbols represent three subscripted vectors and the normal type symbols represent three components of a given vector; $\mathbf{e} = \{ \mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3} \}^T$ represents the three orthogonal vectors defining the inertial reference frame; $\mathbf{b} = \{ \mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3} \}^T$ represents the body-fixed reference frame which is attached to and rotates with the beam cross section; $X = \{ X_1, X_2, X_3 \}^T$ represents the inertial components of the original neutral axis position; $u = \{ u_1, u_2, u_3 \}^T$ represents the inertial components of the subsequent total translational displacement of the neutral axis, and $\ell^T = \{ 0, \ell_2, \ell_3 \}$ are the body-fixed components of the distance from the beam neutral-axis to the material point located on the deformed beam cross-section. It is noted that the beam cross-section is allowed to

rotate such that it is not necessarily perpendicular to the neutral axis in order to model transverse shear deformations. Warping deformation of the cross-section is not taken into consideration.

In order to derive the necessary time derivatives for the description of the large rotation dynamics, we employ the well known formula[33]:

$$\frac{d}{dt} \equiv \frac{d^e}{dt} = \frac{d^b}{dt} + \omega \times \tag{2.2}$$

where ω is the angular velocity vector and the superscripts e and b indicate that the derivatives are to be those observed in the inertial (space) and body (rotating) system of axes respectively. The above is expressed in the matrix form to act on the body frame components of a given vector

$$\frac{d}{dt} = \frac{d^b}{dt} + \tilde{\omega} \ , \tag{2.3}$$

and the velocity and acceleration of the position vector (2.1) are

$$\begin{aligned} \frac{d\mathbf{r}}{dt} &= \frac{du^T}{dt}\mathbf{e} + \ell^T\tilde{\omega}^T\mathbf{b} \\ \frac{d^2\mathbf{r}}{dt^2} &= \frac{d^2u^T}{dt^2}\mathbf{e} + \ell^T\left(\frac{d^b\tilde{\omega}^T}{dt} + \tilde{\omega}^T\tilde{\omega}^T\right)\mathbf{b} \ . \end{aligned} \tag{2.4}$$

Given the following relation between the **b**-basis and the **e**-basis

$$\mathbf{b} = \mathbf{R}\,\mathbf{e} \tag{2.5}$$

where $\mathbf{R}$ is a (3×3) orthogonal transformation matrix, the body frame components of the skew-symmetric angular velocity tensor ($\tilde{\omega}^T$) are

$$\tilde{\omega}^T = \frac{d\mathbf{R}}{dt}\mathbf{R}^T \ . \tag{2.6}$$

A conjugate virtual rotation tensor is defined analogous to the above as

$$\delta\tilde{\alpha}^T = \delta\mathbf{R}\mathbf{R}^T \ , \tag{2.7}$$

and the variation of the position vector (2.1) is given as

$$\delta\mathbf{r} = \delta u^T\mathbf{e} + \ell^T\delta\tilde{\alpha}^T\mathbf{b} \ . \tag{2.8}$$

The equations of motion as derived from the stated beam kinematic description will be discussed next.

3. Spatial Beam Equations of Motion

The principle of virtual work, which is simply a 'weak' or variational form of Cauchy's differential equations of motion for the equilibrium of a given set of particles of a continuum, is stated as[34]

$$\int_V \delta r_i \, \rho \ddot{r}_i \, dV \;+\; \int_V \sigma^e_{ij} \, \frac{\partial \delta r_i}{\partial x_j} \, dV \;=\; \int_V \delta r_i f_i \, dV \;+\; \int_S \delta r_i t_i \, dS \quad . \qquad (3.1)$$

The cartesian coordinates x_i represent the particle position after some deformation has taken place, δr_i a kinematically admissible virtual displacement, $\ddot{r}_i$ the acceleration, f_i the external force per unit mass, and t_i the stress vector acting on a surface with outward normal components n_i. Likewise, σ^e_{ij} represents the cartesian components of the Cauchy stress tensor, and ρ is the mass density. The expression is tailored for the continuum beam by using the kinematic relations (2.1), (2.4), and (2.8) for the components x_i, δr_i, and $\ddot{r}_i$ respectively. As well as providing the basis for a finite element approximation techniques, the variational formulation readily lends itself to the derivation of incremental strain-displacement relations as deduced from the derivatives of the virtual displacement components. The present formulation employs a physical stress measure defined as a force per unit deformed area and the conjugate physical strain increments based on the deformed coordinates. As such, the formulation can be recast into a convected coordinate system moving with the beam, thus simplifying the stress and strain computational procedures. The practical advantages of such a formalism are in real-time software simulation experiments as the computed physical quantities correspond to the actual stress/strain measurements of the sensors located and operating on the deformed structure.

For notational convenience and subsequent finite element discretization, the principle of virtual work is expressed in the following operator form:

$$\delta F^I \;+\; \delta F^S \;=\; \delta F^E \;+\; \delta F^T \qquad (3.2)$$

where the inertia operator δF^I, internal force operator δF^S, external force operator δF^E, and traction operator δF^T are identified from (3.1). Explicit expressions for the various operators incorporating the large rotation beam kinematics are derived in Sections 3.1 to 3.3. The finite element discretizations are given in Section 3.4, and the incorporation of the beam formulation into the multibody dynamics framework is discussed in Section 3.5.

3.1 Spatial Beam Inertia Operator

The inertia operator was defined from (3.1) as

$$\delta F^I \;=\; \int_V \rho \, \delta r_i \ddot{r}_i \, dV \;=\; \int_V \rho \, \delta \mathbf{r} \cdot \ddot{\mathbf{r}} \, dV \qquad (3.3)$$

from which an expression can be derived directly from the kinematic equations (2.4) and (2.8). If the origin of the body-fixed basis is located at the centroid of the cross-section, the following simple expression results for δF^I:

$$\delta F^I \;=\; \int_s \{\; \delta u^T \;\; \delta\alpha^T \;\} \left\{ \begin{array}{c} \rho A \, \frac{d^2 u}{dt^2} \\ \\ J \, \frac{d^b \omega}{dt} \;+\; \tilde{\omega}\, J\, \omega \end{array} \right\} ds \tag{3.4}$$

where

$$\int_A \rho \, \tilde{\ell}\tilde{\ell}^T \, dA \;=\; \mathbf{J}$$

represents the inertia tensor of the beam cross-section and ds represents the remaining integration to be performed over a beam length parameter. The translational inertia is completely decoupled from the rotary inertia and is of the same form as that seen in rigid body dynamics. This is due to the dual choice of the translational displacements measured in the inertial basis and the angular velocity measured in the body-fixed basis located at the center of mass of the cross-section.

3.2 Spatial Beam Internal Force Operator

The internal force operator was defined in (3.1) as

$$\delta \mathbf{F^S} \;=\; \int_V \frac{\partial \delta r_i}{\partial x_j} \, \sigma^e_{ij} \, dV \tag{3.5}$$

identifying as conjugate quantities the virtual displacement gradient and the Cauchy stress tensor. This form of the internal force along with the beam kinematic description will be used to deduce a set of virtual strain-displacement relations that are invariant to rigid body motions. The corresponding conjugate stress tensor will be obtained from an objective incremental procedure that relates incremental strains obtained from the virtual strain tensor to Cauchy stress increments. Thus the internal force term will be derived completely from the original definition of the beam kinematics without making an a priori definition of the existing strains or stresses.

A physically appealing decomposition of the stress and virtual strain tensors into an alternative beam reference frame which lies tangent to the deformed neutral axis is introduced to provide conceptual simplifications in the derivation and subsequent computations. Certain stress states referenced to this convected frame are kinematically required to vanish in a beam formulation. When applied to the convected frame stress components, this choice also leads the task of stress update computations to a simple additive procedure. To this end, we introduce a convected reference frame, denoted by **a**, which is related to the inertial reference frame **e** by

$$\mathbf{a} \;=\; \mathbf{T}\,\mathbf{e} \;, \qquad \mathbf{a} \;=\; \{\; \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3 \;\}^T \;\; . \tag{3.6}$$

For implementation purposes within the context of the finite element method, the convected frame will be constant on the element level and thus is similar in concept to that introduced by Belytschko et al.[15,16]. It is noted that this reference frame does not coincide with the body frame **b** attached to the cross-section. The relative difference between these two reference frames is represented by the rotation matrix S which models the effects of transverse shear and torsion deformation as

$$\mathbf{b} \;=\; \mathbf{S}\,\mathbf{a} \;, \qquad \mathbf{R} \;=\; \mathbf{S}\,\mathbf{T} \;, \tag{3.7}$$

and the latter interdependence between the rotation matrices is established.

The internal force operator, originally characterized by the inertial frame components of the Cauchy stress tensor (σ^e_{ij}) and conjugate virtual displacement gradient, will equivalently be expressed in terms of the convected frame components of the stress tensor (σ^a_{ij}) and a corresponding convected virtual displacement gradient as

$$\delta F^S \;\equiv\; \int_V \frac{\partial \delta r_i}{\partial x_j}\,\sigma^e_{ij}\,dV \;=\; \int_V T_{mi}\,\frac{\partial \delta r_i}{\partial \xi_k}\,\sigma^a_{mk}\,dV \quad . \tag{3.8}$$

The symmetric portion of the transformed deformation gradient is used to define the virtual strain tensor $\delta\varepsilon^a_{mk}$ as

$$\delta\varepsilon^a_{mk} \;\equiv\; \frac{1}{2}\;\left(\; T_{mi}\,\frac{\partial \delta r_i}{\partial \xi_k} \;+\; T_{ki}\,\frac{\partial \delta r_i}{\partial \xi_m}\;\right) \tag{3.9}$$

which is an objective tensor invariant to arbitrary rigid body motions. The internal force, written in terms of the convected frame tensors, will be expressed in vector format as

$$\delta F^S \;=\; \int_V \delta\varepsilon^a_{mk}\,\sigma^a_{mk}\,dV \;=\; \int_V \left\{\,\sigma_{\xi\xi} \quad \widehat{\sigma}_{\xi\eta} \quad \widehat{\sigma}_{\xi\zeta}\,\right\} \left\{ \begin{array}{c} \delta\varepsilon_{\xi\xi} \\ \delta\widehat{\varepsilon}_{\xi\eta} \\ \delta\widehat{\varepsilon}_{\xi\zeta} \end{array} \right\} dV \tag{3.10}$$

where the notation

$$\xi_i \;=\; \{\,\xi_1,\xi_2,\xi_3\,\} \;=\; \{\,\xi,\eta,\zeta\,\}\;, \qquad \widehat{(\,)}_{ij} \;=\; (\,)_{ij} \;+\; (\,)_{ji}$$

denotes the coordinates of the convected reference frame and the engineering shear strain definitions respectively. The rest of the convected frame strain components

$$\delta\varepsilon_{\eta\eta} \quad , \quad \delta\varepsilon_{\zeta\zeta} \quad , \quad \delta\widehat{\varepsilon}_{\eta\zeta}$$

are identically equal to zero due to the original assumptions of the beam kinematics.

A set of virtual strain-displacement relations can be derived from the expressions (2.8) and (3.9). The final result is expressed as

$$\left\{ \begin{array}{c} \delta\varepsilon_{\xi\xi} \\ \delta\widehat{\varepsilon}_{\xi\eta} \\ \delta\widehat{\varepsilon}_{\xi\zeta} \end{array} \right\} \;=\; \delta\gamma \;+\; \tilde{\ell}^T\,\delta\kappa \tag{3.11}$$

where

$$\delta\gamma \;=\; \mathbf{T}\,\frac{\partial \delta u}{\partial \xi} \;+\; \left\{ \begin{array}{c} 0 \\ -\delta\beta_3 \\ \delta\beta_2 \end{array} \right\} \;, \qquad \delta\kappa \;=\; \frac{\partial\, \delta\beta}{\partial \xi} \;, \qquad \delta\beta \;=\; \mathbf{S}^T\, \delta\alpha \tag{3.12}$$

and is comparable to that of Reissner[31]. In the above expressions, $\delta\gamma$ represents the membrane and two transverse shear strains, $\delta\kappa$ the torsion and two bending strains, and $\delta\beta$ the virtual rotations of the cross-section referred to the convected frame.

In an analogous manner the total stress state is expressed as

$$\left\{ \begin{array}{c} \sigma_{\xi\xi} \\ \sigma_{\xi\eta} \\ \sigma_{\xi\zeta} \end{array} \right\} \;=\; \sigma_\gamma \;+\; \tilde{\ell}^T\, \sigma_\kappa \tag{3.13}$$

to be obtained from a separate stress update procedure. A substitution of (3.11) and (3.13) into (3.10), and a spatial integration over a symmetric cross-sectional area results in the following expression for the internal force

$$\delta F^S \;=\; \int_\xi \{\; \delta\gamma^T\, N_\gamma \;+\; \delta\kappa^T\, M_\kappa \;\}\; d\xi \tag{3.14}$$

where N_γ represent the axial and transverse shear forces per unit length, and M_κ represent the torsional and bending moments per unit length as given by

$$N_\gamma \;=\; \int_A \sigma\; dA \;, \qquad M_\kappa \;=\; \int_A \tilde{\ell}^T\, \sigma\; dA \;. \tag{3.15}$$

To be consistent with the inertia operator derived in (3.4), the above is written as

$$\delta F^S \;=\; \int_\xi \; \{\, \delta u^T \quad \delta\alpha^T \,\} \;\; [\; B \;]^T \left\{ \begin{array}{c} N_\gamma \\ M_\kappa \end{array} \right\} \; d\xi \tag{3.16}$$

which involves a transformation back to the body frame components of the virtual rotations and also an identification of the desired incremental strain-displacement matrix B. To effect the change of the body reference frame of the cross-section orientation in space with respect to the constant convected reference frame, we invoke the following relations:

$$\frac{\partial^a \delta\beta}{\partial \xi} \;=\; S^T\, \frac{\partial^a \delta\alpha}{\partial \xi} \;=\; S^T\, (\; \frac{\partial^b \delta\alpha}{\partial \xi} \;+\; \tilde{\kappa}_S\, \delta\alpha \;) \tag{3.17}$$

$$\tilde{\kappa}_S^T \;=\; \frac{\partial^a \mathbf{S}}{\partial \xi}\, \mathbf{S}^T \tag{3.18}$$

which are completely analogous to those relating changes in the time derivative given in (2.3) and (2.6). The strain operator [B] of (3.16) is then recognized as

$$[\ B\ ] = \begin{bmatrix} \mathbf{T}\frac{\partial^a}{\partial\xi} & \tilde{i}_1\ \mathbf{S}^T \\ \mathbf{0} & \mathbf{S}^T(\ \tilde{\kappa}_S\ +\ \mathbf{I}\ \frac{\partial^b}{\partial\xi}\ ) \end{bmatrix}\ , \qquad \tilde{i}_1\ =\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.19}$$

It remains to provide a procedure for updating σ_γ and σ_κ in order to compute N_γ and M_κ. For this purpose, we employ the following rate-type law that relates the instantaneous rate of stress to the instantaneous rate of deformation:

$$\dot{\sigma}^a_{kl}\ \simeq\ c_{klmp}\ \dot{\varepsilon}^a_{mp} \tag{3.20}$$

where c_{klmp} represents the material response tensor, and $\dot{\sigma}^a_{kl}$ and $\dot{\varepsilon}^a_{mp}$ represent the convected frame stress and strain rates, respectively. This approximate constitutive law can be derived by transforming the Truesdell rate equation[35], which is an objective equation based on inertial components of Cauchy stresses and the velocity gradient tensor, to the convected basis. This equation is then integrated in time as

$$\begin{aligned} \sigma^a_{kl}{}^{n+1} &= \sigma^a_{kl}{}^{n} + \int_{t^n}^{t^{n+1}} c_{klmp}\ \dot{\varepsilon}^a_{mp}\ dt \\ &= \sigma^a_{kl}{}^{n} + c_{klmp}\ \Delta\varepsilon^a_{mp} \end{aligned} \tag{3.21}$$

to define the stress update procedure. The evaluation of the strain increments $\Delta\varepsilon^a_{mp}$, to be defined from the virtual strains (3.12), will be detailed in Section 5.

3.3 Spatial Beam External Force and Traction Operator

The external force operator defined in (3.1) as

$$\delta F^E\ =\ \int_V \delta r_i\ f_i\ dV$$

has the final resultant form

$$\delta F^E\ =\ \int_\xi\ \{\ \delta u^T\ \ \delta\alpha^T\ \}\begin{Bmatrix} f^e \\ f^b \end{Bmatrix}\ d\xi \tag{3.22}$$

where f^e represents the inertial components of a force per unit length acting on the beam neutral axis and f^b represents the body-fixed components of a moment per unit length acting on the beam cross-section. The traction operator defined as

$$\delta F^T\ =\ \int_S \delta r_i\ t_i\ dS \tag{3.23}$$

acts on the exterior surfaces of the beam as natural boundary conditions.

3.4 Finite Element Discretization

The variational form of the partial differential equations representing the spatial dynamics of a continuous beam presented in the preceding sections provide a basis for the finite element method to be used as a spatial discretization procedure[36]. In the present study, we restrict ourselves to the use of linear shape functions to approximate the displacement field along the beam, viz.,

$$u = \sum_{I=1}^{npe} N_I \, u_I \tag{3.24}$$

where N_I denotes the spatial linear shape functions, u_I represents the degrees of freedom at the element nodes, and npe denotes the number of nodes per element. The element inertia operator, from (3.4), is written as

$$\begin{aligned} \delta F^I = \sum_{I=1}^{npe} \sum_{K=1}^{npe} \left\{ \delta u_I^T \, \rho A \, M_{IK}^E \, \frac{d^2 u_K}{dt^2} + \delta \alpha_I^T \, \rho J_I \, M_{IK}^E \, \frac{d^b \omega_K}{dt} \right\} \\ + \sum_{I=1}^{npe} \delta \alpha_I^T \, D_I^E \end{aligned} \tag{3.25}$$

where

$$M_{IK}^E = \int_\xi N_I \, N_K \, d\xi \;, \qquad D^E(\omega)_I = \int_\xi (\tilde{\omega} J \omega)_I \, d\xi$$

represent the element mass matrix and nonlinear angular acceleration vector. The former will be evaluated as a standard lumped mass matrix for the computational efficiency of explicit integration techniques to be described in Section 4, and the latter will be evaluated from an average of the element nodal angular velocities. The element internal force operator, from (3.16), is written as

$$\delta F^S = \sum_{I=1}^{npe} \{ \delta u_I \quad \delta \alpha_I \} \, [\, B_I^E \,]^T \begin{Bmatrix} N_\gamma \\ M_\kappa \end{Bmatrix}^E = \sum_{I=1}^{npe} \{ \delta u_I \quad \delta \alpha_I \} \begin{Bmatrix} S_I^e \\ S_I^b \end{Bmatrix}^E \tag{3.26}$$

where the evaluation of the element strain operator

$$[\, B_I^E \,] = \int_\xi \begin{bmatrix} \mathbf{T} \frac{\partial N_I}{\partial \xi} & \tilde{i}_1 \, N_I \, \mathbf{S}_I^T \\ \mathbf{0} & \mathbf{S}_I^T \left(\tilde{\kappa}_S \, N_I + \frac{\partial N_I}{\partial \xi} \right) \end{bmatrix} d\xi \tag{3.27}$$

and the resultant element stresses N_γ and M_κ, as defined in (3.19) and (3.15) respectively, will be presented in detail in Section 5. The element external force operator, from (3.22), is written as

$$\delta F^E = \sum_{I=1}^{npe} \{ \delta u_I \quad \delta \alpha_I \} \begin{Bmatrix} f_I^e \\ f_I^b \end{Bmatrix} , \qquad f_I^{e,b} = \int_\xi N_I \, f^{e,b} \, d\xi \tag{3.28}$$

and the traction operator is implemented as boundary conditions on the nodes. The equations of motion in terms of nodal degrees of freedom ($\delta u_d, \delta \alpha_d$) for the entire beam are obtained from an assembly of the above element operators. For the unconstrained beam, these nodal virtual displacements and rotations are arbitrary independent variations, and the discrete equations of motion are written as

$$\begin{bmatrix} M_d & 0 \\ 0 & J_d \end{bmatrix} \begin{Bmatrix} \ddot{u}_d \\ \dot{\omega}_d \end{Bmatrix} + \begin{Bmatrix} 0 \\ D_d(\omega) \end{Bmatrix} + \begin{Bmatrix} S_d^e \\ S_d^b \end{Bmatrix} = \begin{Bmatrix} f_d^e \\ f_d^b \end{Bmatrix} \tag{3.29}$$

where M_d, J_d represent the assembled mass and inertia matrices, and $D_d(\omega)$, S_d, f_d represent the assembled nonlinear acceleration, internal force, and external force vectors respectively.

3.5 Extension to Multibody Dynamics

The present formulation of spatial beam dynamics as given by (3.29) can readily be incorporated into a general multibody dynamics methodology. The degrees of freedom of a rigid body, namely the inertially-based translational position of the center of mass and the rotational orientation of the body reference frame, coincide with the degrees of freedom of the nodal coordinates of the present beam components. Thus the equations of motion (3.29) can be specialized to represent a rigid body system by setting the internal force S_d equal to zero.

It remains to augment both the holonomic and nonholonomic constraint conditions modeling the contacts among the various bodies to the equations of motion. For this purpose, the Lagrange multiplier technique is used to couple the algebraic constraint equations with the differential equations of motion of the generalized coordinates by augmenting the virtual work of the unconstrained system (3.2) with the virtual work required to enforce the constraints. Given a set of equations representing holonomic constraint conditions between the displacement coordinates as

$$\Phi_H (u, t) = 0 , \qquad \delta \Phi_H = \frac{\partial \Phi_H}{\partial u} \delta u = \mathbf{B}_H \, \delta u = 0 \tag{3.30}$$

and a set representing nonholonomic constraint conditions between the virtual displacements and rotations as

$$\delta \Phi_N (u, \delta u, R, \delta \alpha) = \mathbf{B}_N \begin{Bmatrix} \delta u \\ \delta \alpha \end{Bmatrix} = 0 , \tag{3.31}$$

the virtual work expression (3.2) of the unconstrained system is modified to account for the constraint via Lagrange's multipliers λ as[37]

$$\delta F^I \;+\; \delta F^S \;+\; \lambda_H \cdot \delta\Phi_H \;+\; \lambda_N \cdot \delta\Phi_N \;=\; \delta F^E \;+\; \delta F^T \quad .$$

The virtual displacements and rotations of the generalized coordinates can now be treated as arbitrary independent variations in the modified virtual work expression. The equations of motion for constrained flexible multibody systems with respect to a set of generalized coordinates (u, ω) denoting both the nodal coordinates of the flexible members and the physical coordinates of the rigid bodies can be expressed as

$$\begin{bmatrix} M & 0 \\ 0 & J \end{bmatrix} \begin{Bmatrix} \ddot{u} \\ \dot{\omega} \end{Bmatrix} \;+\; \mathbf{B}^T \lambda \;=\; \begin{Bmatrix} Q_u \\ Q_\omega \end{Bmatrix} \tag{3.32}$$

where

$$\begin{Bmatrix} Q_u \\ Q_\omega \end{Bmatrix} \;=\; \begin{Bmatrix} f^e - S^e(u,q) \\ f^b - D(\omega) - S^b(u,q) \end{Bmatrix} \;,\quad \mathbf{B} \;=\; \begin{bmatrix} B_H \\ B_N \end{bmatrix} \;,\quad \lambda \;=\; \begin{Bmatrix} \lambda_H \\ \lambda_N \end{Bmatrix} \tag{3.33}$$

in which $D(\omega)$ represents the nonlinear acceleration, S the internal force vector, f the external force vector, and $\mathbf{B}^T \lambda$ the constraint force vector. As an additional number of unknown Lagrange multipliers λ for each constraint condition have been introduced along with the generalized coordinates for each degree of freedom, the above system of equations must be augmented with the constraint equations themselves to achieve a determined system of equations.

4. Time Integration Techniques for Constrained Systems

The present methodology to formulate the equations of motion of an arbitrary assemblage of interconnected flexible beams and rigid bodies is readily adaptable for use with existing multibody dynamics solution techniques. The equations (3.32) model the beam components with degrees of freedom u and ω that embody both the rigid and flexible deformation motions. As such there is no need to solve separately generalized coordinates denoting the flexible motion from a reference set of coordinates denoting the rigid motion. In addition, as the nodal coordinates of the beam components are defined in the same kinematic manner as the physical coordinates of the rigid body components, no distinction need be made between the treatment of the flexible and rigid components of the multibody system other than the calculation of the internal force of the flexible member. Therefore, the salient feature of this type of formulation is that numerical solution procedures for the integration of spatial kinematic systems can be directly applied to the generalized coordinates of both the rigid and flexible components.

A multibody dynamics solution procedure, originally demonstrated on rigid body systems in previous studies[38-41], is adopted for the above flexible multibody system equations

of motion. The key to the procedure is a staggered implementation of the separate generalized coordinate integrator and constraint force solver modules. An improved variation of the explicit central difference algorithm, described in Section 4.1, is used to integrate the translational displacements and the angular velocity of the system. An algorithm based on the Euler parameter representation of finite rotations, described in Section 4.2, is used to update the configuration orientation from the angular velocity. The computations of the Lagrange multipliers are then carried out in a separate routine, described in Section 4.3, which implicitly integrates a stabilized companion differential equation for the constraint forces in time.

4.1 Explicit Generalized Coordinate Integrator

The central difference explicit integration algorithm is written as

$$\begin{aligned} \dot{d}^{n+\frac{1}{2}} &= \dot{d}^{n-\frac{1}{2}} + h\,\ddot{d}^{n} \\ d^{n+1} &= d^{n} + h\,\dot{d}^{n+\frac{1}{2}} \\ \ddot{d}^{n+1} &= M^{-1}\,Q\,(\,d^{n+1}\,,\,\dot{d}^{n+1}\,) \end{aligned} \tag{4.1}$$

where the superscript $n = 1, 2, 3, \cdots$ designates the discrete time station $t^n = n\,h$ and h is the stepsize. Unlike in conventional structural dynamics, a straightforward application of (4.1) on the rotational equations

$$J\,\dot{\omega} + \tilde{\omega}\,J\,\omega = f_{\omega}$$

inherent in the multibody system equations of motion (3.32) leads to computational difficulties. In order to compute $\dot{\omega}^{n+1}$, it is necessary to have ω^{n+1}. However, due to the inherent nature of the algorithm, only $\omega^{n+\frac{1}{2}}$ is available. It was shown[41] that the naive approximation

$$\omega^{n+1} \simeq \omega^{n+\frac{1}{2}} \tag{4.2}$$

results in a computationally unstable integration of the angular velocity ω. To correct this within the context of explicit computational sequences, an interlaced application of the central difference algorithm such that the displacements and velocities are advanced one-half time step at a time was proposed[40,41]. The algorithm advances the solution to the time station $t^{n+\frac{1}{2}}$ given the solutions of the two preceding time stations $t^{n-\frac{1}{2}}$ and t^n as follows:

$$
\begin{aligned}
(a) \qquad & u^{n+\frac{1}{2}} = u^{n-\frac{1}{2}} + h\,\dot{u}^{n} \\
(b) \qquad & \dot{u}^{n+\frac{1}{2}} = \dot{u}^{n-\frac{1}{2}} + h\,\ddot{u}^{n} \\
(c) \qquad & \omega^{n+\frac{1}{2}} = \omega^{n-\frac{1}{2}} + h\,\dot{\omega}^{n} \\
(d) \qquad & q^{n+\frac{1}{2}} = q\,(\,\omega^{n+\frac{1}{2}}\,) \\
(e) \qquad & S^{n+\frac{1}{2}} = S\,(\,u^{n+\frac{1}{2}}, q^{n+\frac{1}{2}}\,) \\
(f) \qquad & D^{n+\frac{1}{2}} = D\,(\,\omega^{n+\frac{1}{2}}\,) \;,\quad f^{n+\frac{1}{2}} = f\,(\,t^{n+\frac{1}{2}}\,) \\
(g) \qquad & Q^{n+\frac{1}{2}} = Q\,(\,f^{n+\frac{1}{2}}, S^{n+\frac{1}{2}}, D^{n+\frac{1}{2}}\,) \\
(h) \qquad & \lambda^{n+\frac{1}{2}} = \lambda\,(\,\lambda^{n}, Q^{n+\frac{1}{2}}\,) \\
(i) \qquad & \ddot{u}^{n+\frac{1}{2}} = M^{-1}\,(\,Q_u^{n+\frac{1}{2}} - B_u^T\,\lambda^{n+\frac{1}{2}}\,) \\
(j) \qquad & \dot{\omega}^{n+\frac{1}{2}} = J^{-1}\,(\,Q_\omega^{n+\frac{1}{2}} - B_\omega^T\,\lambda^{n+\frac{1}{2}}\,)
\end{aligned}
$$

The evaluation of the generalized rotational parameters q to be obtained from the angular velocity, as represented by step (d), will be detailed in Section 4.2. The evaluation of the internal force S from the current configuration coordinates u and q, as represented by step (e), will be detailed in Section 5. The evaluation of the Lagrange multipliers λ, as represented by step (h), will be detailed in Section 4.3. The algorithm proceeds to the next half time station t^{n+1}, now given the solutions at time stations t^n and $t^{n+\frac{1}{2}}$, and thus the force and acceleration terms are evaluated twice each time step. The algorithm is initiated for time $t^{\frac{1}{2}}$ given initial conditions for time t^0 in the following manner:

$$
\begin{aligned}
(k) \qquad & \dot{u}^{\frac{1}{2}} = \dot{u}^{0} + \frac{h}{2}\ddot{u}^{0} \\
(l) \qquad & \omega^{\frac{1}{2}} = \omega^{0} + \frac{h}{2}\,\dot{\omega}^{0} \\
(m) \qquad & u^{1} = u^{0} + h\,\dot{u}^{\frac{1}{2}} \\
(n) \qquad & u^{\frac{1}{2}} = \frac{1}{2}\,(\,u^{0} + u^{1}\,)
\end{aligned}
$$

from which steps (d) through (j) can be performed.

One last remark will be made on the angular velocity integration. The equations of motion were derived using body frame angular velocity components. The integration of these quantities shown in step (c) is not formally correct as the components at different time steps are defined with respect to different body-fixed frames. This concern can be eliminated by applying the central difference update to the inertial components of the angular velocity. Step (d) will then consist of an appropriate function of inertial angular velocity components. The integrated inertial angular velocities must be transformed to the moving reference frame before evaluating steps (f) and (j) since the equations of motion are written with respect to the body frame angular velocity description. The angular acceleration evaluated in step (j) must then be transformed back to inertial reference frame before being integrated again in step (c).

4.2 Rotational Parameter Integration

The two-stage explicit integrator was applied to the translational displacement and velocity coordinates and the angular velocity coordinates. As the rotational orientation parameters are not directly integrable from the angular velocity vector, a procedure must be developed to update the configuration orientation given the angular velocity. Any finite rotation can be uniquely expressed by a rotation angle θ and an appropriate rotation axis $\mathbf{n}$ [42]. Two rotational parameterizations based on this description are the rotational vector (Θ) and the Euler parameters ($q_0, \mathbf{q}$) defined respectively as

$$\Theta \;=\; \theta\,\mathbf{n} \quad , \qquad \left\{ \begin{matrix} q_0 \\ \mathbf{q} \end{matrix} \right\} \;=\; \left\{ \begin{matrix} \cos\frac{\theta}{2} \\ \mathbf{n}\,\sin\frac{\theta}{2} \end{matrix} \right\} \tag{4.3}$$

The three parameters of the rotational vector are independent, while the four Euler parameters are subject to the constraints

$$q_0^2 \;+\; \mathbf{q}^T\mathbf{q} \;=\; 1 \tag{4.4}$$

The rotation matrix is represented as a function of the Euler parameters as

$$\mathbf{R} \;=\; \begin{bmatrix} 2(q_0^2+q_1^2)-1 & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & 2(q_0^2+q_2^2)-1 & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & 2(q_0^2+q_3^2)-1 \end{bmatrix} \tag{4.5}$$

The body frame components of the angular velocity tensor defined in (2.6) as

$$\tilde{\omega}_b^T \;=\; \dot{\mathbf{R}}\mathbf{R}^T \;=\; \begin{bmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & -\omega_1 & 0 \end{bmatrix}, \quad \omega_b \;=\; \left\{ \begin{matrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{matrix} \right\}_b$$

has the Euler parameter representation[42]

$$\left\{ \begin{matrix} 0 \\ \omega_b \end{matrix} \right\} \;=\; 2 \begin{bmatrix} q_0 & \mathbf{q}^T \\ -\mathbf{q} & q_0\mathbf{I}-\tilde{\mathbf{q}} \end{bmatrix} \left\{ \begin{matrix} \dot{q}_0 \\ \dot{\mathbf{q}} \end{matrix} \right\} \tag{4.6}$$

A similar expression for the inertial components of the angular velocity tensor

$$\tilde{\omega}_e^T \;=\; \mathbf{R}^T\,\tilde{\omega}_b^T\mathbf{R} \;=\; \mathbf{R}^T\,\dot{\mathbf{R}} \tag{4.7}$$

can be derived as

$$\left\{ \begin{matrix} 0 \\ \omega_e \end{matrix} \right\} \;=\; 2 \begin{bmatrix} q_0 & \mathbf{q}^T \\ -\mathbf{q} & q_0\mathbf{I}+\tilde{\mathbf{q}} \end{bmatrix} \left\{ \begin{matrix} \dot{q}_0 \\ \dot{\mathbf{q}} \end{matrix} \right\} \tag{4.8}$$

The above definitions can be inverted to yield the expressions

$$\left\{ \begin{matrix} \dot{q}_0 \\ \dot{\mathbf{q}} \end{matrix} \right\} \;=\; \frac{1}{2} \begin{bmatrix} 0 & -\omega_b^T \\ \omega_b & \tilde{\omega}_b^T \end{bmatrix} \left\{ \begin{matrix} q_0 \\ \mathbf{q} \end{matrix} \right\} \;=\; \mathbf{A}_b\,(\,\omega_b\,) \left\{ \begin{matrix} q_0 \\ \mathbf{q} \end{matrix} \right\} \tag{4.9}$$

for the body frame components and

$$\left\{ \begin{array}{c} \dot{q}_0 \\ \dot{\mathbf{q}} \end{array} \right\} = \frac{1}{2} \left[\begin{array}{cc} 0 & -\omega_e^T \\ \omega_e & \tilde{\omega}_e \end{array} \right] \left\{ \begin{array}{c} q_0 \\ \mathbf{q} \end{array} \right\} = \mathbf{A_e}\,(\,\omega_e\,)\, \left\{ \begin{array}{c} q_0 \\ \mathbf{q} \end{array} \right\} \tag{4.10}$$

for the inertial frame components. A general representation

$$\dot{q} = \mathbf{A}(\omega)\, q \;, \qquad q = \left\{ \begin{array}{c} q_0 \\ \mathbf{q} \end{array} \right\} \tag{4.11}$$

will be used to denote (4.9) or (4.10) given the angular velocity description. These inverse expressions are derived from (4.6) and (4.8) by incorporating the derivative of the constraint equation (4.4)

$$\dot{q}_0\, q_0 + \dot{\mathbf{q}}^T \mathbf{q} = 0 \quad . \tag{4.12}$$

The configuration orientation is obtained from a numerical time discretization of the above Euler parameter - angular velocity representations. Among several possibilities, the approximation that satisfies the constraint condition (4.12) in the discrete sense is the following trapezoidal formula

$$\frac{1}{h}\,(\,q^{n+1} - q^n\,) = \mathbf{A}(\omega^{n+\frac{1}{2}})\, \frac{1}{2}\,(\,q^{n+1} + q^n\,) \quad . \tag{4.13}$$

Due to the structure of $\mathbf{A}$, the solution matrix can be analytically inverted such that the discrete orientation update

$$q^{n+1} = \frac{1}{D}\,[\,I + \frac{h}{2}\,\mathbf{A}(\omega^{n+\frac{1}{2}})\,]\,[\,I + \frac{h}{2}\,\mathbf{A}(\omega^{n+\frac{1}{2}})\,]\, q^n \tag{4.14}$$

where

$$D = 1 + \frac{h^2}{4}\,(\,w_1^2 + w_2^2 + w_3^2\,)$$

results. The final result is normalized to satisfy the constraint (4.4). The above equation is valid for either the body or inertial frame decomposition of the angular velocity as long as the corresponding form of $\mathbf{A}$ from (4.9) or (4.10) is used. The resulting update (4.14) involves only explicit computations and is readily incorporated into the two-stage explicit integration procedure.

4.3 Constraint Force Solution Procedure

A partitioned solution procedure has been employed to solve the generalized coordinates separately from the Lagrange multipliers. To effect a partitioned solution of the constraints, a stabilized companion differential equation for the constraint forces is formed by adopting the penalty procedure[38,39]. The penalty procedure uses the equations

$$\lambda_H = \frac{1}{\epsilon}\,\Phi_H \;, \qquad \dot{\lambda}_N = \frac{1}{\epsilon}\,\dot{\Phi}_N \;, \qquad \epsilon \to 0 \tag{4.15}$$

as the basic constraint equations instead of (3.30) and (3.31) for the holonomic and the nonholonomic constraint conditions respectively. The penalty equations can be written in the general form, from (3.30) and (3.31), as

$$\dot{\lambda} \;=\; \frac{1}{\epsilon}\, B\, \dot{z} \;, \qquad \dot{z} \;=\; \left\{ \begin{matrix} \dot{u} \\ \omega \end{matrix} \right\} \tag{4.16}$$

The numerical solution to the above companion differential equation is obtained as follows. The constrained equations of motion (3.32) are integrated once from (3.20) using the implicit integration rule

$$\dot{z}^{n+\frac{1}{2}} \;=\; \dot{z}^{n} \;+\; \delta\, \ddot{z}^{n+\frac{1}{2}} \;, \quad \delta \;=\; \frac{h}{2}$$

as

$$\dot{z}^{n+\frac{1}{2}} \;=\; \delta\, M^{-1}\, (\, Q^{n+\frac{1}{2}} \;-\; B^{T}\, \lambda^{n+\frac{1}{2}}\,) \;+\; \dot{z}^{n} \tag{4.17}$$

This expression is substituted into (4.16) to obtain the stabilized differential equation for the Lagrange multipliers

$$\epsilon \dot{\lambda}^{n+\frac{1}{2}} \;+\; \delta B\, M^{-1} B^{T}\, \lambda^{n+\frac{1}{2}} \;=\; \delta B\, M^{-1}\, Q^{n+\frac{1}{2}} \;+\; B\, \dot{z}^{n} \;. \tag{4.18}$$

The same integration rule is applied to this equation to result in the discrete update

$$(\, \epsilon\, I \;+\; \delta^{2}\, B\, M^{-1} B^{T}\,)\, \lambda^{n+\frac{1}{2}} \;=\; \epsilon \lambda^{n} \;+\; r_{\lambda}^{n+\frac{1}{2}} \tag{4.19}$$

$$r_{\lambda}^{n+\frac{1}{2}} \;=\; \delta^{2}\, B\, M^{-1} Q^{n+\frac{1}{2}} \;+\; \delta\, B\, \dot{z}^{n} \;. \tag{4.20}$$

The same procedure can also be derived with different integration rules. The update of the Lagrange multipliers, performed for each half time step, is easily adapted into the two-stage explicit integration procedure.

5. Internal Force Computations

The algorithmic treatment of the nonlinear stiffness operator is addressed in this section. The explicit generalized coordinate integrator of the previous section requires an evaluation of the internal force at a current time step t^{n+1} from the coordinates of the beam configuration at that time. The internal force is first evaluated on the element level for all the finite elements comprising the flexible component from (3.26) as

$$\left\{ \begin{matrix} S_{I}^{e} \\ \\ S_{I}^{b} \end{matrix} \right\}^{E} \;=\; [\, B_{I}^{E}\,]^{T} \left\{ \begin{matrix} N_{\gamma} \\ M_{\kappa} \end{matrix} \right\}^{E} \tag{5.1}$$

after which these individual element computations are assembled to form the internal force of the discrete beam. The necessary computations to be described are the evaluations of

the discrete strain operator $[\ B^E_I\ ]$ defined in (3.23) and the resultant stresses N_γ and M_κ respectively.

The Timoshenko beam formulation in which the translational degrees of freedom are independent from the rotational degrees of freedom requires an approximation within the element such that these variables will be continuous across the element boundaries. Thus a two node finite element representing a linear interpolation of the translational and rotational variables is a sufficient discretization of the beam. To avoid the locking phenomenon, the interpolation of the rotational degrees of freedom associated with the transverse shear strain is underintegrated. After incorporating these concepts into (3.27), the resulting expression for the discrete strain operator is given by

$$[\ B^E_I\ ] \;=\; \begin{bmatrix} -\frac{1}{\ell}\mathbf{T} & \frac{1}{\ell}\mathbf{T} & \frac{1}{2}\tilde{i}_1\ \mathbf{S}_1^T & \frac{1}{2}\tilde{i}_1\ \mathbf{S}_2^T \\ \\ \mathbf{0} & \mathbf{0} & \mathbf{S}_1^T\ (\ \frac{1}{2}\tilde{\kappa}_S\ -\ \frac{1}{\ell}\mathbf{I}\ ) & \mathbf{S}_2^T\ (\ \frac{1}{2}\tilde{\kappa}_S\ +\ \frac{1}{\ell}\mathbf{I}\ ) \end{bmatrix} \tag{5.2}$$

which acts on the virtual displacements and rotations

$$\{\ \delta u_1 \quad \delta u_2 \quad \delta\alpha_1 \quad \delta\alpha_2\ \}^T$$

where the subscripts refer to the element node number. The convected frame $\mathbf{T}$ matrix, body frame curvature tensor $\tilde{\kappa}_S$, and element neutral-axis length ℓ are constant quantities over the element domain, while the relative cross-section deformation $\mathbf{S}$ matrices are nodal quantities. The computation of these terms from the nodal displacement and rotation coordinates of the current configuration are detailed in Section 5.1.

A stress update procedure of the form

$$\begin{Bmatrix} N_\gamma \\ M_\kappa \end{Bmatrix}^{n+1} \;=\; \begin{Bmatrix} N_\gamma \\ M_\kappa \end{Bmatrix}^{n} \;+\; \Delta \begin{Bmatrix} N_\gamma \\ M_\kappa \end{Bmatrix} \tag{5.3}$$

is used to derive the resultant stresses of the current configuration at time t^{n+1} from the resultant stresses of the past configuration at time t^n. The simple additive form of the procedure, which was derived from the numerical integration of a rate-type constitutive law, is due to the use of a convected frame stress and conjugate strain decomposition. The resultant stress increments ΔN_γ and ΔM_κ are obtained via

$$\Delta\ N_\gamma \;=\; \begin{bmatrix} EA & 0 & 0 \\ 0 & GA & 0 \\ 0 & 0 & GA \end{bmatrix} \Delta\gamma \quad , \quad \Delta M_\kappa \;=\; \begin{bmatrix} GJ & 0 & 0 \\ 0 & EI_2 & 0 \\ 0 & 0 & EI_3 \end{bmatrix} \Delta\kappa \quad . \tag{5.4}$$

A set of strain increments $\Delta\gamma$ and $\Delta\kappa$, which denote the change from time t^n to t^{n+1}, are defined as a finite analogy to the infinitesimal virtual strains $\delta\gamma$ and $\delta\kappa$ derived in Section 2. A specific computational procedure designed for use with this incremental interpretation

of the continuum-based formulation such that the computed finite strain increments are invariant to arbitrary rigid body motions is discussed in Section 5.2.

5.1 Computation of the Strain Operator

The reference frames introduced in the formulation, namely the body frame **b** attached to the cross-section and the convected frame **a** tangent to the deformed neutral axis, are computed as follows. The Euler parameters representing the orientation of the beam cross-section at the finite element nodes are output from the generalized coordinate integrator at each time step. The rotation matrices $\mathbf{R}_i$, representing the **b** reference frame at each element node, are thus computed directly from the Euler parameter representation of a rotation matrix (4.5). This matrix contains rotational information of both that due to the rigid motion of the convected reference frame and the transverse shear and torsional deformations of the cross-section relative to the convected frame.

The neutral axis of the finite element is defined as the straight line connecting the two element nodes, the tangent of which is trivial, and is directly calculated from the translational displacements output from the generalized coordinate integrator. Given this tangent $\mathbf{a}_1$, the $\mathbf{a}_2$ vector is defined as the cross product of $\mathbf{a}_1$ with the $\mathbf{b}_3$ axis of $\mathbf{R}_1$, and the remaining axis $\mathbf{a}_3$ defined to complete the right-hand coordinate system. The computed axis $\{ \mathbf{a}_1 , \mathbf{a}_2 , \mathbf{a}_3 \}$, as shown in Figure 2, define the rows of the $\mathbf{T}$ matrix. The rotation matrices $\mathbf{S}_i$, defined at each element node as the relative difference between the element convected frame and the nodal body frames, are thus

$$\mathbf{S}_i \;=\; \mathbf{R}_i\,\mathbf{T}^T \;, \qquad i \;=\; 1,2 \;\; . \tag{5.5}$$

The procedure is an approximation applicable for moderate strains such that the $\mathbf{S}_i$ matrices contain information solely due to transverse shear and torsional deformations[43]. The rotation matrices of the discrete strain operator (5.2) have thus been defined.

The body frame components of the curvature tensor $\tilde{\kappa}_S^T$ defined in (3.18) as

$$\tilde{\kappa}_S^T \;=\; \frac{\partial^a \mathbf{S}}{\partial \xi}\,\mathbf{S}^T \;=\; \begin{bmatrix} 0 & \kappa_3 & -\kappa_2 \\ -\kappa_3 & 0 & \kappa_1 \\ \kappa_2 & -\kappa_1 & 0 \end{bmatrix} \;, \quad \kappa \;=\; \begin{Bmatrix} \kappa_1 \\ \kappa_2 \\ \kappa_3 \end{Bmatrix}$$

are equivalent to

$$\tilde{\kappa}_S^T \;=\; \frac{\partial^e \mathbf{R}}{\partial \xi}\,\mathbf{R}^T \tag{5.6}$$

as the convected frame $\mathbf{T}$ matrix is defined to be constant along the element domain where the differentiation is performed. This definition is completely analogous to the angular velocity tensor defined in (2.6) and motivates the use of an Euler parameter representation of the curvature completely analogous to the Euler parameter representation of angular

velocity (4.6) as a basis for the computation of the element curvature from the nodal rotational variables. The Euler parameter - curvature representation is

$$\begin{Bmatrix} 0 \\ \kappa \end{Bmatrix} = 2 \begin{bmatrix} q_0 & \mathbf{q}^T \\ -\mathbf{q} & q_0\mathbf{I} - \tilde{\mathbf{q}} \end{bmatrix} \begin{Bmatrix} \frac{\partial q_0}{\partial \xi} \\ \frac{\partial \mathbf{q}}{\partial \xi} \end{Bmatrix} = \mathbf{E}\,(\,q_a\,)\,\frac{\partial q}{\partial \xi} \tag{5.7}$$

subject to the constraints

$$q_0{}^2 + \mathbf{q}^T\mathbf{q} = 1\ , \qquad \frac{\partial q_0}{\partial \xi} q_0 + \frac{\partial \mathbf{q}}{\partial \xi}^T \mathbf{q} = 0 \quad . \tag{5.8}$$

An approximation to be used in (5.7) that satisfies the constraint conditions in the discrete sense

$$\frac{\partial q}{\partial \xi} = \frac{1}{\ell}\,(\,q_2 - q_1\,)\ , \qquad q_a = \frac{\frac{1}{2}\,(\,q_1 + q_2\,)}{\|\frac{1}{2}\,(\,q_1 + q_2\,)\,\|} \tag{5.9}$$

is evaluated using the Euler parameters of the element nodes output from the generalized coordinate integrator. It will be shown that this discrete computation is invariant to rigid rotations contained in the total nodal Euler parameters.

The simple normalized average of the nodal Euler parameters has a physical interpretation. The Euler parameters q_a correspond to an average orientation, in a geometric sense, of the two nodal cross-section orientations. This is demonstrated from the following example characterizing a state of constant curvature of a finite element shown in Figure 3. The orientation of the convected element frame is characterized by a rotation of an angle ϕ about an axis $\mathbf{n}_a$ from the inertial reference frame, and the relative nodal cross-section orientations are characterized by a rotation from the convected frame of angles $-\tau$ and τ about axis $\mathbf{n}_b$ for nodes 1 and 2 respectively. The Euler parameters designating the total cross section orientation of the two nodes due to these combined effects can be expressed as

$$\begin{aligned} q_1 &= \begin{Bmatrix} \cos\frac{\phi}{2}\cos\frac{\tau}{2} + \mathbf{n}_a\cdot\mathbf{n}_b\,\sin\frac{\phi}{2}\sin\frac{\tau}{2} \\ -\cos\frac{\phi}{2}\sin\frac{\tau}{2}\,\mathbf{n}_b + \cos\frac{\tau}{2}\sin\frac{\phi}{2}\,\mathbf{n}_a - \sin\frac{\tau}{2}\sin\frac{\phi}{2}\,\mathbf{n}_a\times\mathbf{n}_b \end{Bmatrix} \\ q_2 &= \begin{Bmatrix} \cos\frac{\phi}{2}\cos\frac{\tau}{2} - \mathbf{n}_a\cdot\mathbf{n}_b\,\sin\frac{\phi}{2}\sin\frac{\tau}{2} \\ \cos\frac{\phi}{2}\sin\frac{\tau}{2}\,\mathbf{n}_b + \cos\frac{\tau}{2}\sin\frac{\phi}{2}\,\mathbf{n}_a + \sin\frac{\tau}{2}\sin\frac{\phi}{2}\,\mathbf{n}_a\times\mathbf{n}_b \end{Bmatrix} \end{aligned} \tag{5.10}$$

which is obtained by applying the quaternion product rule[44] to the Euler parameter definitions

$$q_{r_1} = \begin{Bmatrix} \cos\frac{\tau}{2} \\ -\sin\frac{\tau}{2}\,\mathbf{n}_b \end{Bmatrix}\ , \quad q_{r_2} = \begin{Bmatrix} \cos\frac{\tau}{2} \\ \sin\frac{\tau}{2}\,\mathbf{n}_b \end{Bmatrix}\ , \quad q_a = \begin{Bmatrix} \cos\frac{\phi}{2} \\ \sin\frac{\phi}{2}\,\mathbf{n}_a \end{Bmatrix}$$

of the relative nodal orientations and the convected orientation respectively. The average of the two nodal Euler parameters (5.10) is

$$\frac{1}{2}\,(\,q_1 + q_2\,) = \begin{Bmatrix} \cos\frac{\phi}{2}\cos\frac{\tau}{2} \\ \cos\frac{\tau}{2}\sin\frac{\phi}{2}\,\mathbf{n}_a \end{Bmatrix}\ ,$$

the norm of which is $\cos\frac{\tau}{2}$. When normalized, the above average is identical to the average orientation of the two nodes given by q_a. It can be shown that for this example the discretization (5.9) when substituted into (5.7) gives the finite element curvature computation

$$\kappa \;=\; \frac{4}{\ell}\,\sin\frac{\tau}{2}\,\mathbf{n}_b$$

which approximates the true curvature strain $\frac{2}{\ell}\tau\mathbf{n}_b$. The computation retains only the rotation parameters τ originally defined relative to the rigid body orientation, and is thus invariant to the rigid body motions. For instances when the validity of the approximation is challenged, an incremental curvature computation can be made as discussed in the next section, from which the total curvature is obtained from an appropriate update procedure.

5.2 Computation of the Strain Increments

The strain increments are defined from the virtual strains (3.12) by replacing the variational operator δ with an incremental operator $\hat{\Delta}$ as

$$\hat{\Delta}\gamma \;=\; \mathbf{T}\,\frac{\partial\hat{\Delta}u}{\partial\xi} \;+\; \begin{Bmatrix} 0 \\ -\hat{\Delta}\beta_3 \\ \hat{\Delta}\beta_2 \end{Bmatrix}\;, \qquad \hat{\Delta}\kappa \;=\; \frac{\partial\,\hat{\Delta}\beta}{\partial\xi}$$

such that $\hat{\Delta}u$ and $\hat{\Delta}\beta$ are finite analogs of the infinitesimal displacements and rotations δu and $\delta\beta$. For computation purposes, it becomes necessary to decompose the convected frame components of the virtual rotations of the of the cross-section $\delta\beta$ into a rotation due to rigid body motion $\delta\varphi$ and that due to deformations $\delta\tau$ as

$$\delta\beta \;=\; \delta\tau_a \;+\; \delta\varphi \quad . \tag{5.11}$$

This relation is derived by substituting the following definitions

$$\delta\tilde{\beta}^T \;=\; \mathbf{S}^T\,\delta\tilde{\alpha}^T\,\mathbf{S}\;, \qquad \delta\tilde{\alpha}^T \;=\; \delta\mathbf{R}\mathbf{R}^T$$

$$\delta\tilde{\varphi}^T \;=\; \delta\mathbf{T}\mathbf{T}^T\;, \qquad \delta\tilde{\tau}^T \;=\; \delta\mathbf{S}\mathbf{S}^T\;, \qquad \delta\tilde{\tau}_a^T \;=\; \mathbf{S}^T\delta\tilde{\tau}^T\,\mathbf{S}$$

into the identity

$$\mathbf{R} \;=\; \mathbf{S}\,\mathbf{T}\;, \qquad \delta\mathbf{R} \;=\; \delta\mathbf{S}\,\mathbf{T} \;+\; \mathbf{S}\,\delta\mathbf{T}\quad .$$

It is noted that $\delta\alpha$, $\delta\varphi$, and $\delta\tau$ represent moving frame or spatial components referred to the defining reference frame, whereas $\delta\beta$ and $\delta\tau_a$ represent material components referred back to the convected frame. From these definitions, the incremental strain $\hat{\Delta}\gamma$ is given by

$$\hat{\Delta}\gamma \;=\; \mathbf{T}\,\frac{\partial\hat{\Delta}u}{\partial\xi} \;+\; \begin{Bmatrix} 0 \\ -\hat{\Delta}\varphi_3 \\ \hat{\Delta}\varphi_2 \end{Bmatrix} \;+\; \begin{Bmatrix} 0 \\ -\hat{\Delta}\tau_{a_3} \\ \hat{\Delta}\tau_{a_2} \end{Bmatrix} \tag{5.12}$$

representing the membrane strain and transverse shear strain increments. Likewise, the incremental curvature representing the torsion and bending strains is given by

$$\hat{\Delta}\kappa \;=\; \frac{\partial\, \hat{\Delta}\tau_a}{\partial \xi} \tag{5.13}$$

as the incremental rotations $\Delta\varphi$ defined from the $\mathbf{T}$ matrix are constant over the element length.

Essential for the use of these incremental strains is a proper definition and subsequent computation of the finite displacement and finite rotation increments. The incremental translations are defined by

$$\hat{\Delta}u \;\equiv\; u^{n+1} \;-\; u^n \tag{5.14}$$

as the displacements are true vector quantities. The incremental rotations are defined as follows. Rotations are updated by the product of orthogonal matrices via either[24]

$$\begin{aligned} \mathbf{R}^{n+1} \;&=\; \mathbf{R}_{(l)}\,\mathbf{R}^n \;=\; e^{\tilde{\theta}^T}\;\mathbf{R}^n \\ &=\; \mathbf{R}^n\,\mathbf{R}_{(r)} \;=\; \mathbf{R}^n\, e^{\tilde{\Theta}^T} \end{aligned} \tag{5.15}$$

using the rotational vectors θ or Θ based on the spatial or material reference frames respectively. It can be seen from the linearizations of the left and right rotational updates[24]

$$\begin{aligned} \mathbf{R}^{n+1} \;&\simeq\; \mathbf{R}^n \;+\; \delta\mathbf{R} \\ \delta\mathbf{R} \;&=\; \tilde{\theta}^T\,\mathbf{R}^n \;=\; \mathbf{R}^n\,\tilde{\Theta}^T \end{aligned}$$

that the virtual rotations

$$\delta\tilde{\varphi}^T \;=\; \delta\mathbf{T}\,\mathbf{T}^T\;, \qquad \delta\tilde{\tau}_a^T \;=\; \mathbf{S}^T\,\delta\mathbf{S}$$

correspond to spatial and material rotation updates

$$\mathbf{T}^{n+1} \;=\; \Delta\mathbf{T}\,\mathbf{T}^n\;, \qquad \mathbf{S}^{n+1} \;=\; \mathbf{S}^n\Delta\mathbf{S} \tag{5.16}$$

respectively. Thus $\hat{\Delta}\varphi$ and $\hat{\Delta}\tau_a$ are defined as the rotational vectors parameterizing the matrices $\Delta\mathbf{T}$ and $\Delta\mathbf{S}$ respectively. Two different approximate methods which then extract this pseudovector from the given rotation matrix are used to obtain the incremental rotations. The particular approximation methods are chosen such that objective computations of the incremental strains (5.12) and (5.13) are achieved.

To this end, the first two terms of (5.12)

$$\Delta\gamma_1 \;=\; \mathbf{T}\,\frac{\partial\hat{\Delta}u}{\partial\xi} \;+\; \begin{Bmatrix} 0 \\ -\hat{\Delta}\varphi_3 \\ \hat{\Delta}\varphi_2 \end{Bmatrix} \tag{5.17}$$

must be computed such that the $\hat{\Delta}\varphi$ rotation increment compensates for the rigid rotation contained in the displacement increment $\hat{\Delta}u$ defined in (5.14). To accomplish this, $\hat{\Delta}\varphi$ is computed by

$$\hat{\Delta}\tilde{\varphi}^T \;=\; \Delta\mathbf{T}^{n+\frac{1}{2}} \;-\; \Delta\mathbf{T}^{n+\frac{1}{2}\,T} \tag{5.18}$$

where $\Delta\mathbf{T}^{n+\frac{1}{2}}$ is defined from

$$\mathbf{T}^{n+\frac{1}{2}} \;\equiv\; \exp\,\left(\;\frac{1}{2}\Delta\tilde{\varphi}^T\;\right)\,\mathbf{T}^n \;\equiv\; \Delta\mathbf{T}^{n+\frac{1}{2}}\,\mathbf{T}^n \quad . \tag{5.19}$$

The computation was derived from the linear approximation

$$\mathbf{T}^{n+1} \;\simeq\; \left(\; I \,+\, \Delta\tilde{\varphi}^T\;\right)\,\mathbf{T}^n$$

rewritten as

$$\hat{\Delta}\tilde{\varphi}^T \;=\; \left(\;\mathbf{T}^{n+1} \,-\, \mathbf{T}^n\;\right)\,\mathbf{T}^{n+\frac{1}{2}} \tag{5.20}$$

and introducing (5.19) to achieve a skew symmetric matrix. In order to preserve rigid motions, the matrix $\mathbf{T}$ in the first term of (5.17) must be evaluated as $\mathbf{T}^{n+\frac{1}{2}}$. This is shown as follows from an example of the rigid rotation of an element in which $\Delta\gamma_1 \equiv 0$. From (5.20), it is seen that the rotational term of (5.17) becomes

$$\left\{\begin{array}{c} 0 \\ -\hat{\Delta}\varphi_3 \\ \hat{\Delta}\varphi_2 \end{array}\right\} \;=\; -\,\mathbf{T}^{n+\frac{1}{2}}\,\left(\,{t_\xi}^{n+1} \,-\, {t_\xi}^n\,\right)\;, \qquad t_\xi \;=\; \left\{\begin{array}{c} T_{11} \\ T_{12} \\ T_{13} \end{array}\right\} \tag{5.21}$$

The finite element evaluation of the displacement term of (5.17) is given by

$$\mathbf{T}\frac{\partial\hat{\Delta}u}{\partial\xi} \;=\; \mathbf{T}\;\frac{1}{\ell_e}\;\left(\;\hat{\Delta}u_2 \,-\, \hat{\Delta}u_1\;\right) \tag{5.22}$$

for the two-noded beam element of length ℓ_e. For the rigid rotation of the second node about the first node, the incremental translational displacements are simply

$$\hat{\Delta}u_1 \;=\; 0\;, \qquad \hat{\Delta}u_2 \;=\; \ell_e\,\left(\,{t_\xi}^{n+1} - {t_\xi}^n\,\right)\;, \qquad \frac{\partial\hat{\Delta}u}{\partial\xi} \;=\; t_\xi^{n+1} \,-\, t_\xi^n \quad ,$$

as the direction cosines of the rotation are contained in the first row of the $\mathbf{T}$ matrix. Thus for (5.17) to be identically equal to zero, it is necessary to evaluate (5.22) using $\mathbf{T}^{n+\frac{1}{2}}$. To obtain the true stretch with respect to the neutral-axis reference frame at the current configuration, we simply rotate the mid-configuration computation up to the current configuration as

$$\Delta\mathbf{T}^{n+\frac{1}{2}}\;\left[\;\mathbf{T}^{n+\frac{1}{2}}\;\frac{\partial\hat{\Delta}u}{\partial\xi} \;+\; \left\{\begin{array}{c} 0 \\ -\hat{\Delta}\varphi_3 \\ \hat{\Delta}\varphi_2 \end{array}\right\}\;\right] \tag{5.23}$$

As in the preceding analysis, the incremental displacements for an arbitrary rotation and stretch are given by

$$\hat{\Delta}u_1 \;=\; 0 \;, \qquad \hat{\Delta}u_2 \;=\; \left(\left(\ell_e \,+\, d\right) t_\xi^{\,n+1} \;-\; \ell_e \, t_\xi^{\,n} \right)$$

where d represents a stretch relative to the original element length ℓ_e. The rotational expression (5.21) remains valid, and the bracketed term in (5.23) becomes

$$\left(\frac{d}{\ell_e \,+\, d} \right) \mathbf{T}^{n+\frac{1}{2}} \, t_\xi^{\,n+1} \quad .$$

Premultiplication of the above by $\Delta\mathbf{T}^{n+\frac{1}{2}}$ results in the final computation

$$\left(\frac{d}{\ell_e \,+\, d} \right) \mathbf{T}^{n+1} \, t_\xi^{\,n+1} \;=\; \left(\frac{d}{\ell_e \,+\, d} \right) \left\{ \begin{matrix} 1 \\ 0 \\ 0 \end{matrix} \right\}$$

containing solely a measure of stretch regardless of the magnitude of the rigid rotation.

The incremental rotations $\hat{\Delta}\tau_a$ used to compute the remaining terms,

$$\hat{\Delta}\gamma_2 \;=\; \left\{ \begin{matrix} 0 \\ -\hat{\Delta}\tau_{a_3} \\ \hat{\Delta}\tau_{a_2} \end{matrix} \right\} \qquad \hat{\Delta}\kappa \;=\; \frac{\partial\, \hat{\Delta}\tau_a}{\partial \xi} \tag{5.24}$$

representing transverse shear and curvature strains respectively, are computed independently from $\hat{\Delta}\varphi$ as follows. The rotation increments $\hat{\Delta}\tau_a$ are obtained from the matrix $\Delta\mathbf{S}$ defined in (5.16) denoting the relative orientation between the current deformation matrix $\mathbf{S}^{n+1}$ and the past deformation matrix $\mathbf{S}^n$ rigidly rotated to the current convected frame. Another method to extract a rotation pseudovector from a given orthogonal rotation matrix given by[43]

$$\hat{\Delta}\tilde{\tau}_{a\,i}^{T} \;=\; \frac{2\,(\,\Delta\mathbf{S}_i \;-\; \Delta\mathbf{S}_i^T\,)}{1 \;+\; tr\;\Delta\mathbf{S}_i} \;, \qquad i = 1,2 \tag{5.25}$$

is used to define $\hat{\Delta}\tau_a$ at each element node. The above method yields a simpler and more accurate computation of a rotation vector than (5.18). Whereas (5.18) was necessary to compute $\hat{\Delta}\varphi$ such that the rigid rotations within (5.17) are preserved, (5.25) is used within (5.24) as this computation is made from matrices which by construction contain information solely due to deformation. Given the nodal rotation increments, the locking-free form of the elemental shear strain is obtained from the nodal average as

$$\hat{\Delta}\gamma_2 \;=\; \frac{1}{2} \left\{ \begin{matrix} 0 \\ -\hat{\Delta}\tau_{a_3} \\ \hat{\Delta}\tau_{a_2} \end{matrix} \right\}_1 \;+\; \frac{1}{2} \left\{ \begin{matrix} 0 \\ -\hat{\Delta}\tau_{a_3} \\ \hat{\Delta}\tau_{a_2} \end{matrix} \right\}_2 \quad ,$$

and the elemental curvature is computed from the finite element approximation

$$\hat{\Delta}\kappa \;=\; \frac{1}{\ell}\;(\hat{\Delta}\tau_{a_2} \;-\; \hat{\Delta}\tau_{a_1}\;)\quad .$$

This completes the computational procedures for the incremental strains. The detailed strain computations of (5.12) and (5.13) are used in (5.4) to determine the stress increments, from which the current stress state is obtained from the update procedure (5.3).

6. Numerical Examples

The computational techniques, namely the staggered multibody dynamics solution procedure combining the generalized coordinate integrator and the constraint force solver discussed in Section 4 and the finite element computations of the beam internal force discussed in Section 5, have been implemented into a Fortran 77 software package. The result is a robust method which solves the present formulation of the equations of motion of an arbitrary assemblage of flexible beams and rigid bodies. In order to demonstrate the current software capabilities, the following examples highlighting the flexible motion of the beam component are presented.

The first example is included to verify the geometric stiffening phenomena exhibited by a rotating beam[6,18,21,28]. The beam is pinned at the left end; the other end remains free. The following material and geometric properties were used:

$$EA = 2.8 \times 10^7\ lb, \quad GA = 1.0 \times 10^7\ lb, \quad EI = 1.4 \times 10^4\ lb\ in^2$$

$$\rho A = 1.2\ lbm/in, \quad \rho I = 6.0 \times 10^{-4}\ lbm\ in, \quad l = 10\ in.$$

A prescribed angular rotation about the $\mathbf{e}_3$ axis of

$$\theta(t) = \begin{cases} \dfrac{6}{15}\left[\dfrac{t^2}{2} + \dfrac{15^2}{2\pi}\left(\cos\dfrac{2\pi t}{15} - 1\right)\right]\ rad & 0 \le t \le 15\ sec \\ (6t - 45)\ rad & t > 15\ sec \end{cases}$$

is applied at the pinned end. The time history of the tip deflection relative to a reference frame coinciding with the prescribed angular position and the time history of several configurations of the beam are given in Figure 4. As alluded to in the introduction, an overall steady rotation of the beam gives rise to a centrifugal force which is responsible for a change in the bending stiffness that cannot be predicted using linear deformation theories. After initial increasing tip deflections, the beam begins to stiffen as the angular velocity increases due to the centrifugal inertia force. As the angular velocity reaches a constant state, the beam then reaches a steady state phase of small vibrations. This example shows the capability of the nonlinear strain formulation to automatically account for the geometric stiffening effect. The results are comparable to those presented by Simo

and Vu-Quoc[28]. To reproduce these results with alternative methods as the substructuring technique[21], a convergence analysis based on the selection of mode shapes must be performed.

The next examples exhibit the combined large deformation and large rotation capabilities of the present formulation. In the first instance, the beam is pinned as above and is subjected to given initial velocity impulses exciting various deformation mode shapes under planar motion. The following material and geometric properties were used in order to witness finite deformations:

$$EA = 4.0 \times 10^7 \; lb, \quad GA = 2.0 \times 10^7 \; lb, \quad EI = 1.3 \times 10^7 \; lb \; in^2$$

$$\rho A = .98 \; lbm/in, \quad \rho I = 3.3 \times 10^{-2} \; lbm \; in, \quad l = 200 \; in.$$

The initial velocity profiles with the resulting time histories of several deformed configurations are given in Figures 5, 6, and 7. It is noted the versatility of the formulation in its ability to capture the response to a variety of situations in which different fundamental modes of the beam are excited. The approach avoids the difficulty of tailoring the selection of modes shapes of the flexible components to the given problem at hand. The repeatability of the deformation shapes through time is due to the invariance of the internal force computations to the overall rigid motion. This property of computational objectivity is further illustrated in Figure 8 which shows the time history of the strain, kinetic, and total energy over four revolutions for the first bending mode example. The nature of the time integration and internal force algorithms are such that the conservation of energy is retained computationally, as seen by the fact that the total energy remains constant over all the revolutions. Similar results, not presented within, are obtained for the other deformation examples.

To present the applicability of the flexible beam component within the multibody dynamics framework, the final example of a spatial double pendulum is given. The double pendulum is modeled with two beams; a spherical joint connects the last node of the first beam to the first node of the second beam and also pins the first node of the first beam. It is noted that the joint connection can easily be accounted for from a finite element assemblage which leaves the rotational degrees of freedom free at the hinge location. The method was used to verify the results obtained using the Lagrange multiplier solver on the augmented equations described in Section 3.5. In the first case, the pendulum is subjected to a gravity field in the vertical z-direction and an initial velocity impulse in the horizontal x-y plane such that soley rigid motion is excited. The problem is run for four cases of increasing beam flexibility as follows:

1. $EA = 1.0 \times 10^4 \; lb \qquad GA = 0.5 \times 10^4 \; lb$
2. $EA = 1.0 \times 10^3 \; lb \qquad GA = 0.5 \times 10^3 \; lb$
3. $EA = 2.0 \times 10^2 \; lb \qquad GA = 1.0 \times 10^2 \; lb$
4. $EA = 1.0 \times 10^2 \; lb \qquad GA = 0.5 \times 10^2 \; lb$

with the remaining parameters

$$\rho A = 1\ lbm/in \qquad \rho I = .833 \times 10^{-3}\ lbm\ in \qquad l = 1\ in$$

held constant. The initial velocity impulse, and the spatial trajectories of the mass center of the second beam as projected on the x-y and x-z planes is given in Figure 9. The trajectory of the first case coincides exactly with a rigid body solution to the problem, and the slight deviation of the trajectories due to the increasing flexibility can be seen. The energy time histories for the problem, given in Figure 10, verify the computational objectivity of the algorithm as again energy is identically conserved. Again, the invariance of the internal force calculations in the three dimensional environment is witnessed by the negligible strain energy contribution for all of the flexible cases. The time integration of the spatial kinematics preserves the balance between the kinetic and potential energies of the problem. Next, the flexible double pendulum is given an initial velocity impulse to excite deformation motion as well as the rigid motion. For this case the parameters used were

$$EA = 1.8 \times 10^8\ lb, \quad GA = 0.9 \times 10^8\ lb, \quad EI = 1.4 \times 10^8\ \ lb\ in^2$$

$$\rho A = .98 lbm/in, \quad \rho I = 0.67\ lbm\ in, \quad l = 120\ \ in.$$

The initial velocity profile, the resulting time histories of several deformed configurations and energy time history are given in Figure 11, exhibiting the large spatial rotation and deformation capabilities of the formulation. The energy conservation is retained for the computations of spatial deformations.

Further examples of large scale multibody systems are in process, and these results are to be presented in the near future.

7. Concluding Remarks

A flexible beam finite element that is readily incorporated into multibody dynamics applications has been presented. The beam formulation is based on fully nonlinear strain measures which remain invariant to rigid body motions. The model retains a Cauchy stress and physical strain description, and as such it can be easily interfaced with real-time slewing control applications as the measured strains can directly be used as a feedback signal without requiring sophisticated transformations. In addition, the formulation uses an inertial reference for the beam dynamics such that the degrees of freedom of the flexible component are defined in the same sense as the rigid components by including without distinction both the rigid and flexible deformation motions. The consequence is adaptability into multibody dynamics methodologies as numerical solution procedures for the integration of spatial kinematic systems can directly be applied to the generalized coordinates of both the rigid and flexible components. The success of the approach relies on an accurate computation of the nonlinear internal force term. For this reason, the calculation of finite strain increments has been presented which are invariant to arbitrary rigid motions of the beam. The proposed methodology is suitable to treat the dynamics of flexible beams which

undergo a variety of structural deformations in addition to the large overall motions. The same approach can be used in formulating other types of structural components.

Acknowledgements

The work reported herein was supported by NASA/Langley Research Center under Grant NAG-1-756. The authors wish to thank Dr. Jerry Housner for his interest and support during the course of the present work.

8. References

1. Ashley, H., "Observation on the dynamic behavior of flexible bodies in orbit," *AIAA J.* **5** (1967), 460–469
2. Canavin, J.R. and Likins, P.W., "Floating reference frames for flexible spacecraft," *J. of Spacecraft* (1977), 724-732.
3. De Veubeke, B.F., "The dynamics of flexible bodies," *Int J. Engng. Sci.* **14** (1976), 895–913.
4. Grotte, P.B., J.C. McMunn, and R. Gluck, "Equations of motion of flexible spacecraft," *J. of Spacecraft and Rockets* **8** (1971), 561–567.
5. McDonough, T.B., "Formulation of the global equations of motion of a deformable body," *AIAA J.* **14** (1976), 656-660.
6. Kane, T. and Levinson, D., "Simulation of large motions of nonuniform beams in orbit: Part II - The unrestrained beam," *J. Astronautical Sciences* **29**, (1981), 245–275.
7. Bodley, C., Devers, A., Park, A., and Frisch, H., "A digital computer program for dynamic interaction simulation of controls and structures (DISCOS)," NASA Technical Paper 1219, May 1978.
8. Song, J.O. and Haug, E.J., "Dynamic analysis of planar flexible mechanisms," *Comp. Meth. Appl. Mech. Engrg.* **24** (1980), 359–381.
9. Cavin, R.K. and Dusto, A.R., "Hamilton's principle: finite-element methods and flexible body dynamics," *AIAA J.* **15** (1977) 1684–1690.
10. Agrawal, O.P. and Shabana, A.A., "Dynamic analysis of multibody systems using component modes," *Computers & Structures* **21** (1985), 1303–1312.
11. Agrawal, O.P. and Shabana, A.A., "Application of deformable-body mean axis to flexible multibody system dynamics, " *Comp. Meth. Appl. Mech. Engrg.* **56** (1986), 217–245.
12. Shabana, A.A. and Wehage, R.A., "A coordinate reduction technique for dynamic analysis of spatial substructures with large angular rotations," *J. Struct. Mech.* **11** (1983), 401–431.

13. Yoo, W.S. and Haug, E.J., "Dynamics of articulated structures, Part I: Theory and Part II: Computer implementation and applications," *J. of Structure Mechanics* **14** (1986), 105–126 and 177-189.

14. Yoo, W.S. and Haug, E.J., "Dynamics of flexible mechanical systems using vibration and static correction modes," *J. Mechanisms, Transmissions, and Automation in Design* **108** (1986) 315–322.

15. Belytschko, T. and Hsieh, B.J., "Nonlinear transient finite element analysis with convected coordinates," *Int. J. Num. Meth. Eng.* **7** (1973), 255–271.

16. Belytschko, T., Schwer, L., and Klein, M.J., "Large displacement, transient analysis of space frames," *Int. J. Num. Meth. Eng.* **11** (1977), 65–84.

17. Housner, J., "Convected transient analysis for large space structures maneuver and deployment," Proc. the 25th Structures, Structural Dynamics and Materials Conference, AIAA Paper No. 84-1023, (1984) 616–629.

18. Housner, J.M., Wu, S.C., and Chang, C.W., "A finite element method for time varying geometry in multibody structures," Proc. the 29th Structures, Structural Dynamics and Materials Conference, April 1988, AIAA Paper No. 88-2234.

19. Laskin, R.A., Likins, P.W., and Longman, R.W., " Dynamical Equations of a Free-Free Beam Subject to Large Overall Motions," *J. Astronautical Sciences* **31** (1983), 507–528.

20. Kane, T.R., Ryan, R.R., and Banerjee, A.K., "Dynamics of a cantilever beam attached to a moving base," *J. Guidance, Control, and Dynamics* **10** (1987) 139–151.

21. Wu, S.C. and Haug, E.J., "Geometric nonlinear substructuring for dynamics of flexible mechanical systems," *Int. J. Num. Meth. Engrg.* **26** (1988) 2211–2226.

22. Bakr, E.M. and Shabana, A.A., "Geometrically nonlinear analysis of multibody systems," *Computers & Structures* **23** (1986) 739–751.

23. Christensen, E.R. and Lee, S.W., "Nonlinear finite element modeling of the dynamics of unrestrained flexible structures," *Computers & Structures* **23** (1986) 819–829.

24. Cardona, A. and Geradin, M., "A beam finite element nonlinear theory with finite rotations," *Int. J. Num. Meth. Eng.* **26** (1988), 2403-2438.

25. Iura, M. and Atluri, S.N., "On a consistent theory, and variational formulation of finitely stretched and rotated 3-D space-curved beams," *Computational Mechanics* (4) (1989), 73–88.

26. Simo, J.C., "A finite strain beam formulation. Part I: The three dimensional dynamic problem," *Comp. Meth. Appl. Mech. Engrg.* **49** (1985), 55–70.

27. Simo, J.C. and Vu-Quoc, L., "A three-dimensional finite strain rod model. Part II: Computational aspects," *Comp. Meth. Appl. Mech. Engrg.* **58** (1986), 79–116.

28. Simo, J.C. and Vu-Quoc, L., "Finite-strain rods undergoing large motions," *Comp. Meth. Appl. Mech. Engrg.* **66** (1988), 125-161.

29. Antman, S.S., "Kirchhoff problem for nonlinearly elastic rods," *Quat. J. Appl. Math XXXII* **3** (1974), 221–240.

30. Reissner, E., "On a one–dimensional large–displacement finite–strain beam theory," *Studies in Applied Mathematics* **52** (1973), 87–95.

31. Reissner, E., "On finite deformations of space–curved beams," *ZAMP* **132** (1981), 734–744.

32. Wempner, G., *Mechanics of Solids with Applications to Thin Bodies*, Sijthoff & Noordhoff, The Netherlands (1981).

33. Goldstein, H., *Classical Mechanics, 2nd ed.*, Addison-Wesley (1980).

34. Malvern, L.E., *Introduction to the Mechanics of a Continuous Medium* , Prentice-Hall, Inc., Englewood Cliffs, N. J., (1969).

35. Eringen, A.C., *Mechanics of Continua*, Robert E. Krieger Publishing Co., Huntington, N.Y., (1980).

36. Zienkiewicz, O.C., 1977, *The Finite Element Method, 3rd ed.*, McGraw Hill Book Company, Ltd, London.

37. Lanczos, L., *The Variational Principles of Mechanics, 4th ed.*, University of Toronto Press, 1970.

38. Park, K.C., and Chiou, J.C., "Evaluation of constraint stabilization procedures for multibody dynamical systems," *Proc. the 28th Structures, Structural Dynamics and Materials Conference*, Part 2A, Monterey, CA, AIAA Paper No. 87-0927 (1987), 769–773.

39. Park, K.C., and Chiou, J.C., "Stabilization of computational procedures for constrained dynamical systems," *Journal of Guidance, Control and Dynamics* **11** (1988), 365–370.

40. Park, K.C., Chiou, J.C., and Downer, J.D., "A computational procedure for large rotational motions in multibody dynamics," *Proc. the 29th Structures, Structural Dynamics and Materials Conference*, Part 3, Williamsburg, VA, AIAA Paper No. 88-2416 (1988), 1593–1601 (also to appear in J. Guidance, Control and Dynamics).

41. Park, K.C., Chiou, J.C., and Downer, J.D., "An explicit-implicit staggered procedure for multibody dynamics analysis, Part I: Algorithmic aspects," Report No. CU-CSSC-88-08, Center for Space Structures and Controls, University of Colorado (1988).

42. Wittenburg, J., *Dynamics of Systems of Rigid Bodies*, B.G. Teubner, Stuttgart, 1977.

43. Rankin, C.C. and Brogran, F.A., "An element independent corotational procedure for the treatment of large rotations," *J. of Pressure Vessel Technology* **108** (1986) 165–174.

44. Geradin, M., Robert, G., and Buchet, P.,"Kinematic and dynamic analysis of mechanisms: A finite element approach based on Euler parameters," in: *Finite Element Methods for Nonlinear Problems*(P. Bergan, ed.), Berlin, Heidelberg, New York, Springer, 1986.

FIGURES

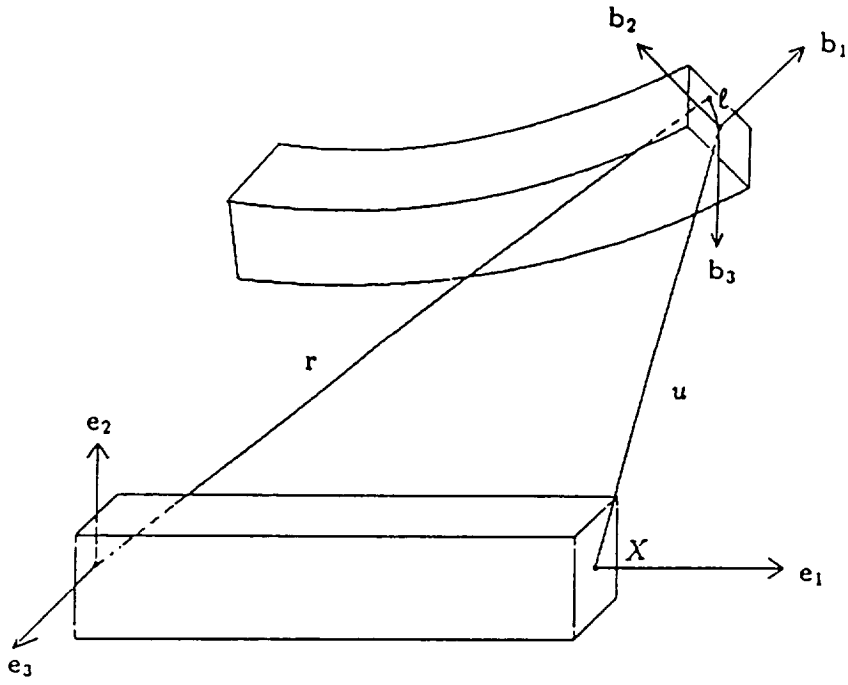


Figure 1. Beam Kinematics

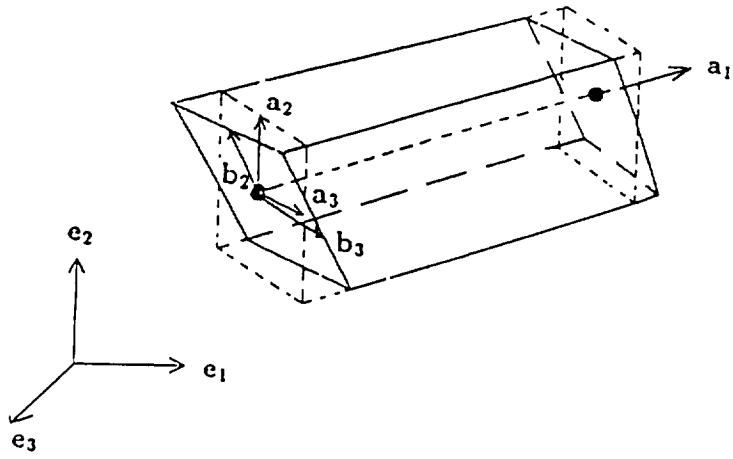


Figure 2. Convected Reference Frame

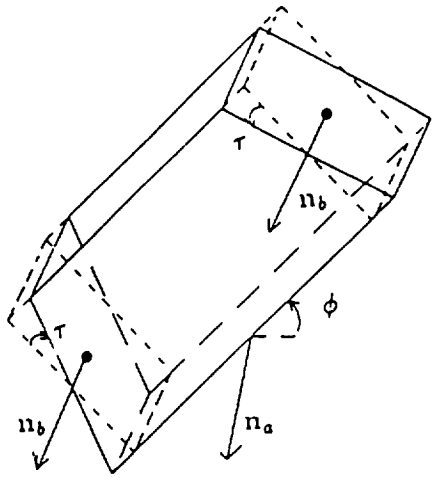
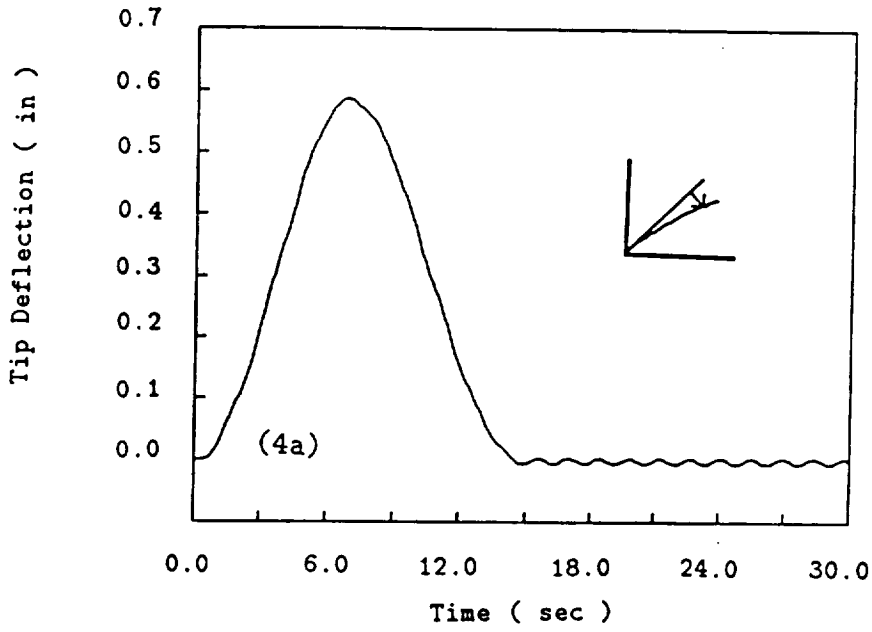


Figure 3. Pure Bending of Beam Element



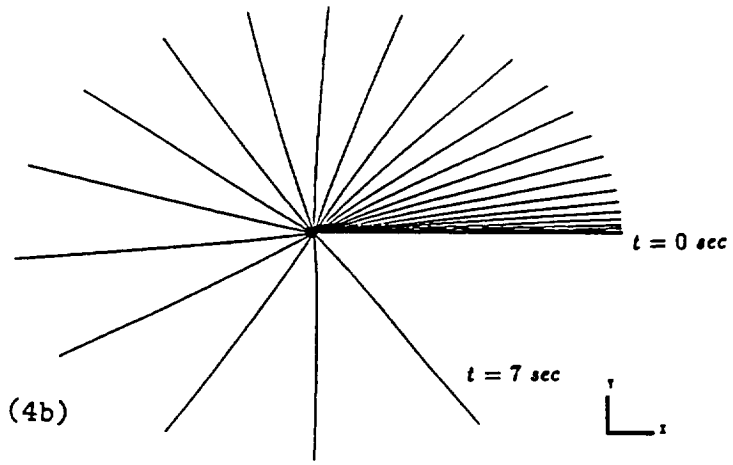
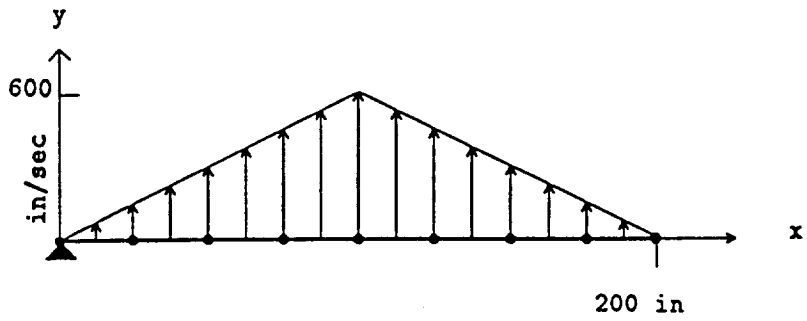
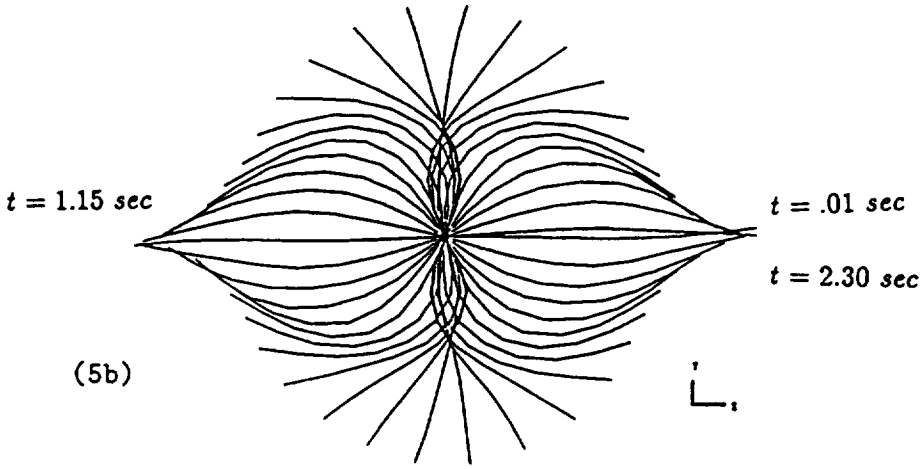


Figure 4. Geometric Stiffening (5 Elements):
(a) Tip Deflection vs. Time
(b) Displacement History

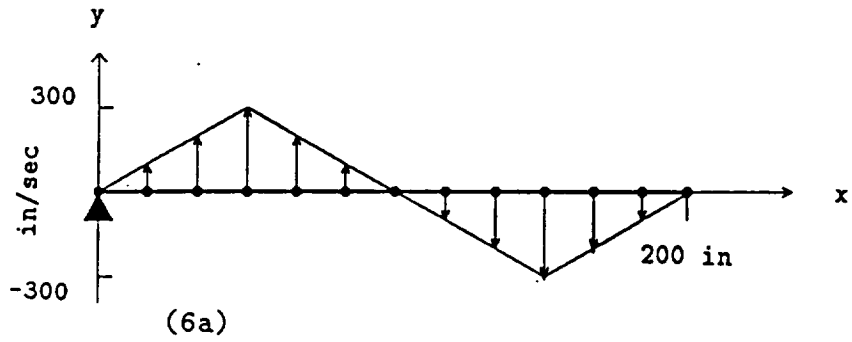


(5a)

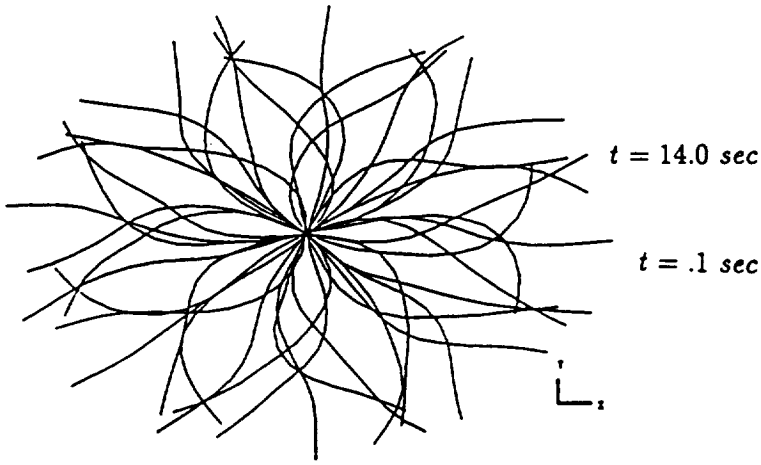


(5b)

Figure 5. First Bending Mode (8 Elements):
(a) Initial Beam Position vs. Initial Velocity Profile
(b) Displacement History

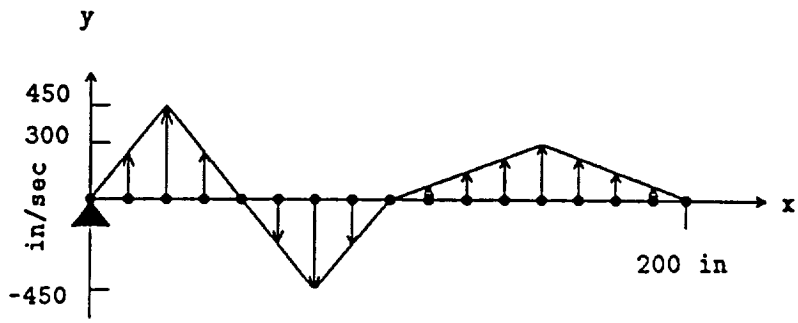


(6a)

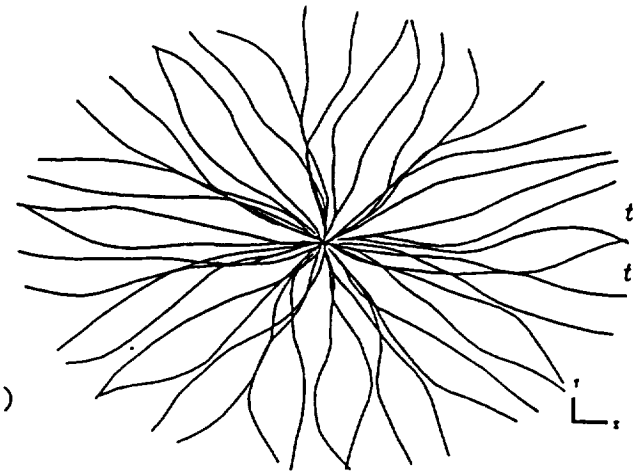


(6b)

Figure 6. Second Bending Mode (12 Elements):
(a) Initial Beam Position vs. Initial Velocity Profile
(b) Displacement History



(7a)



(7b)

Figure 7. Combination Bending Mode (16 Elements):
(a) Initial Beam Position vs. Initial Velocity Profile
(b) Displacement History

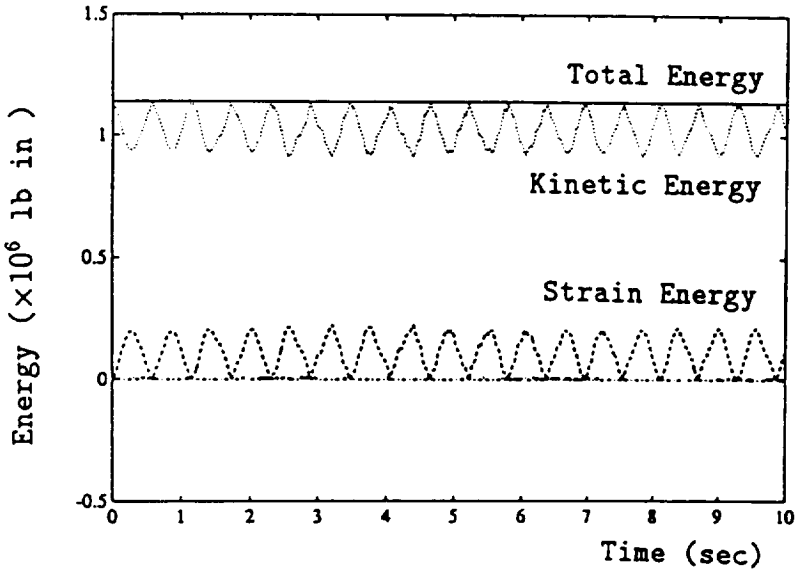


Figure 8. First Bending Mode: Energy Conservation

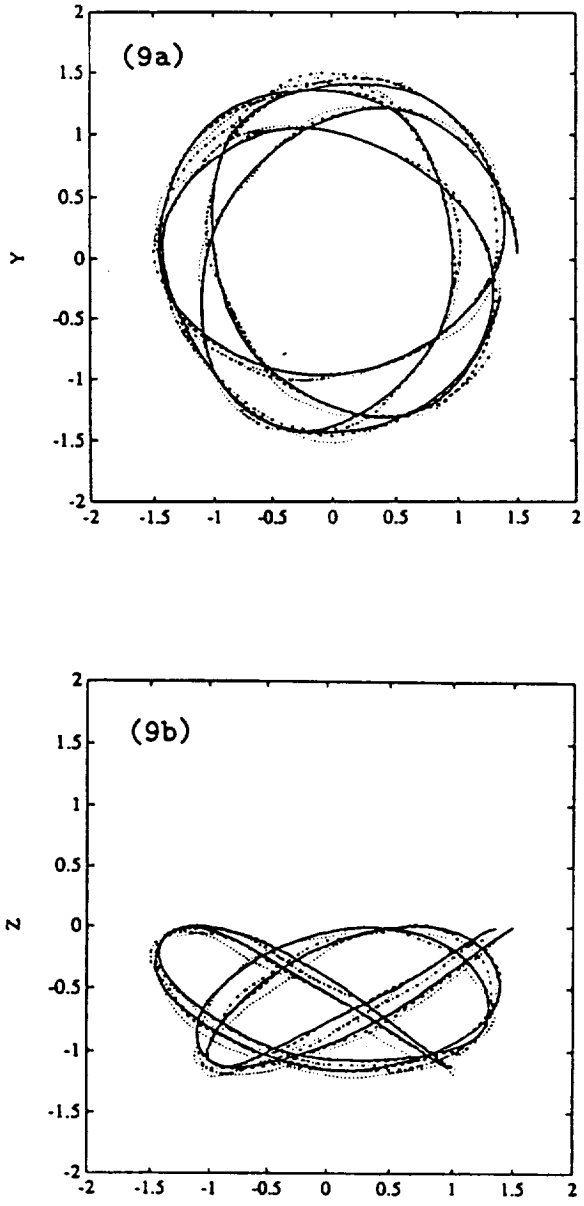


Figure 9. Spatial Double Pendulum (16 Elements):
(a) Second Beam Trajectory: X-Y Plane
(b) Second Beam Trajectory: X-Z Plane

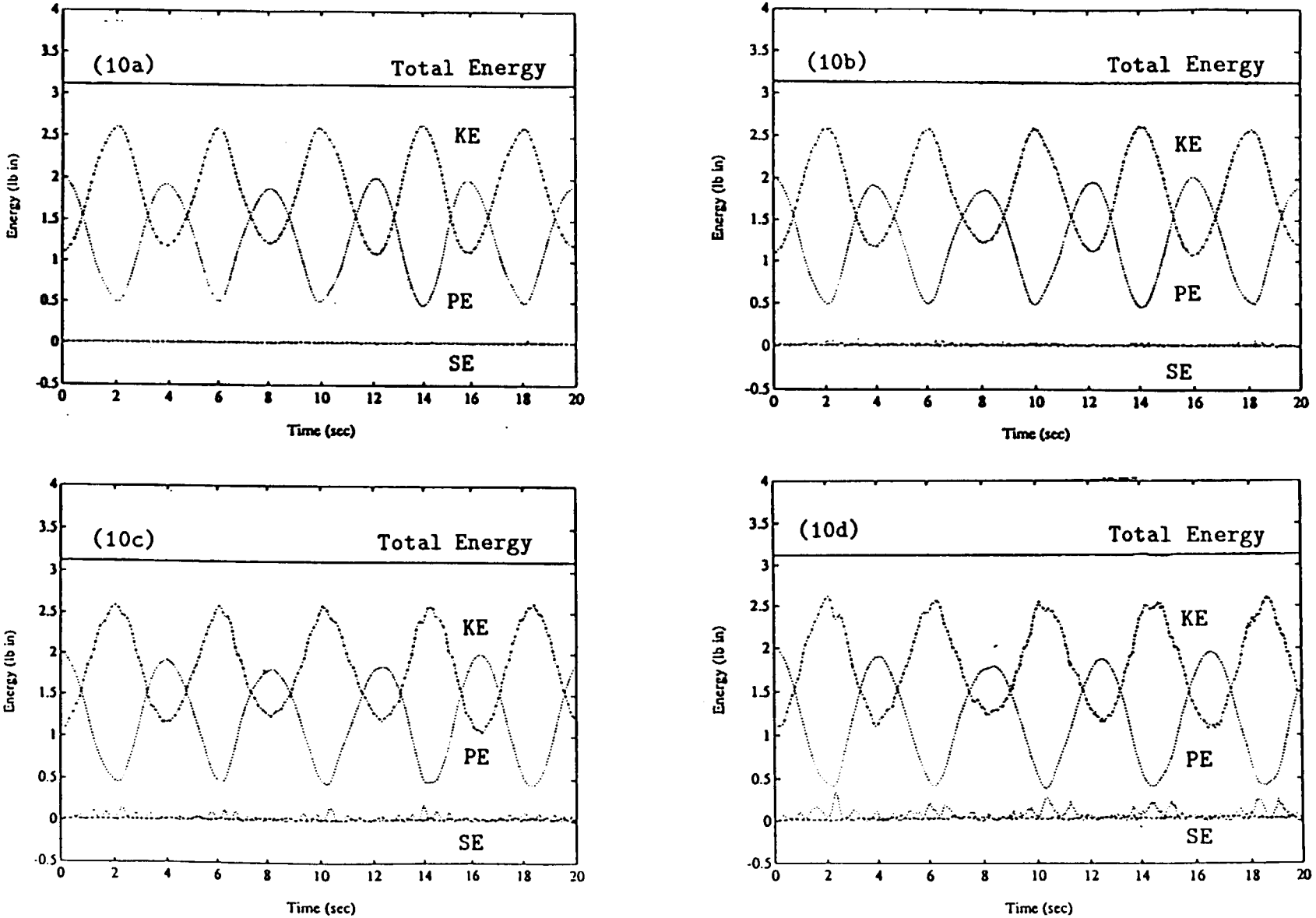


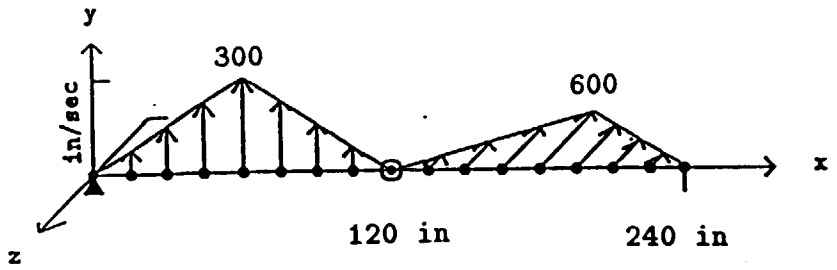
Figure 10. Spatial Double Pendulum: Energy Conservation

(a) $EA = 1.0 \times 10^4$ lb $GA = 0.5 \times 10^4$ lb

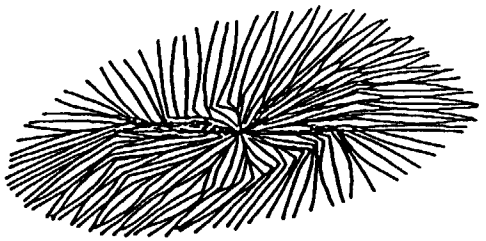
(b) $EA = 1.0 \times 10^3$ lb $GA = 0.5 \times 10^3$ lb

(c) $EA = 2.0 \times 10^2$ lb $GA = 1.0 \times 10^2$ lb

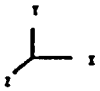
(d) [illegible]



(11a)



(11b)



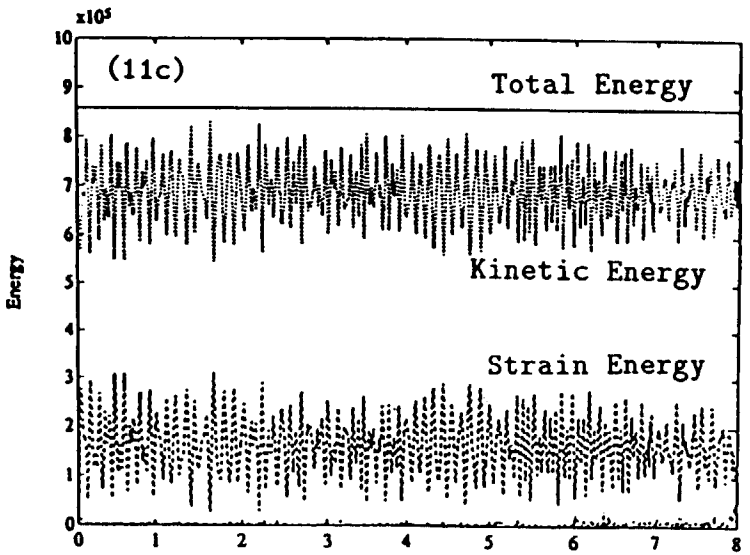


Figure 11. Spatial Double Pendulum (16 Elements):
(a) Initial Beam Position vs. Initial Velocity Profile
(b) Displacement History
(c) Energy Conservation

A Natural Partitioning Scheme for Parallel Simulation of Multibody Systems

J. C. Chiou, K. C. Park, and C. Farhat

Center for Space Structures and Controls and
Department of Aerospace Engineering Sciences
University of Colorado, Campus Box 429
Boulder, Co. 80309

Abstract

A parallel partitioning scheme based on physical-coordinate variables is presented to systematically eliminate system constraint forces and yield the equations of motion of multibody dynamics systems in terms of their independent coordinates. Key features of the present scheme include an explicit determination of the independent coordinates, a parallel construction of the null space matrix of the constraint Jacobian matrix, an easy incorporation of the previously developed two-stage staggered solution procedure, and a Schur complement based parallel preconditioned conjugate gradient numerical algorithm.

1. Introduction

In the past decade, several stand alone general-purpose multibody simulation codes [1-11] have achieved progressive development for their capability to apply to multidisciplinary engineering problems to improve either control system design and verification or system design and dynamics analysis. As a result, these computer codes have been successfully applied to a number of multibody dynamics (MBD) problems such as robot arm maneuvers, spacecrafts and ground vehicle dynamics. However, when systems become very complex, computational efficiency becomes a dominant concern during the preliminary design stage that require many analysis iterations. This has motivated us to make an effective use of parallel computational technology in order to speed up the dynamics analysis of MBD systems, thus ultimately achieving real-time simulation for large-scale problems. The issues of exploiting the parallelism that are inherent in MBD systems include a versatile data structure for describing system topology, an automatic procedure to generate system equations of motion, a streamlined incorporation or elimination of system constraints, a robust time integration algorithm, and an easy interpretation of the simulation results.

In general, the equations of motion for MBD systems can be generated by employing a set of generalized coordinates to define the state of the system [6-8]. Note that, the motions of each body in the system can initially be assumed to be independent of one another. Kinematic relationships between bodies in the system are then imposed, which result in the corresponding constraint conditions. If one augments the constraint equations to the governing equations of motion by introducing the Lagrange multipliers, the resulting equations of motion are characterized as differential-algebraic equations (DAE).

Since a closed-form solution of DAE is in general not attainable except for highly simplified problems, two different approaches have been developed for the solution of DAE. The first approach adopts so-called constraint stabilization methods [12,13,17-19,23] which integrate and solve DAE while attempting to satisfy the constraint equations. From computational point of view, this approach utilizes a large number of equations yet preserves the sparsity of the solution matrix and simple expression for the kinematic relationships. The second approach eliminates system dependent coordinates which is equivalent to eliminating the Lagrange multipliers from DAE so that a set of second order differential equations can be obtained. Schemes [7,10,20-22] leading to such approach include the generalized coordinate partitioning (GCP) scheme, the singular vales decomposition (SVD) scheme and the null space (NS) scheme. In contrast to the first approach, the second approach enjoys a minimal set of equations of motion but suffers from dense solution matrices and highly nonlinear kinematic descriptions.

Numerical experience indicates that constraint stabilization methods are generally preferred for closed kinematic loops whereas constraint elimination methods are better suited for open kinematic links. The objective of the paper is to present a parallel constraint elimination algorithm by constructing the null space of the constraint Jacobian matrix, and employ a parallel preconditioned conjugate gradient numerical algorithm to solve for the equations of motion that are given in Schur complement form.

To address the present natural partitioning scheme, the paper is organized as follows. Section 2 presents the equations of motion that have been derived in DAE form. Section 3 describes the natural partitioning scheme in detail with several example problems. Section 4 applies a parallel computational algorithm to the second order differential equations.

Section 5 describes a parallel preconditioned conjugate gradient scheme that is used to find the solution for the independent accelerations. Section 6 reports on some preliminary results that were obtained using the natural partitioning scheme and the staggered solution procedure that has been previously developed [15,16].

2. Equations of Motion for Multibody Systems

The equations of motion for a MBD system can be derived and expressed in various forms depending upon the type of coordinates one has chosen to describe the configuration of the bodies in the system. In the present derivation, a spatial position vector with respect to an inertial reference frame is described by using Cartesian coordinate. A body-fixed coordinate is then attached to the center of mass of each body. The position of a body is then defined from the origin of the inertial reference frame to the origin of the body-fixed frame, and the position of a particle at the body is defined from the origin of the body-fixed frame to the particle. A velocity vector $\dot{\mathbf{u}}$ contains the translational velocity $\dot{\mathbf{r}}$ which is defined by the inertial frame and angular velocity $\boldsymbol{\omega}$ which is defined by the body-fixed frame. When d'Alembert's principle of virtual work is applied to the entire system plus the constraint equations via Lagrange multipliers, the equations of motion for a multibody dynamics system with $\mathbf{n}$ physical coordinates and $\mathbf{m}$ constraints can be expressed in the following DAE form :

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{B}^T\boldsymbol{\lambda} = \mathbf{F} \tag{2.1}$$

with holonomic constraints

$$\boldsymbol{\Phi}(\mathbf{u}) = 0 \tag{2.2}$$

The first and second time differentiation of (2.2) yield

$$\dot{\boldsymbol{\Phi}}(\mathbf{u}) = \mathbf{B}\dot{\mathbf{u}} = 0 \tag{2.3}$$

and

$$\ddot{\boldsymbol{\Phi}}(\mathbf{u}) = \mathbf{B}\ddot{\mathbf{u}} + \dot{\mathbf{B}}\dot{\mathbf{u}} = 0 \tag{2.4}$$

where $\mathbf{M}$ is the $\mathbf{n} \times \mathbf{n}$ constant mass matrix, $\mathbf{B} = \boldsymbol{\Phi}_{\mathbf{u}}$ is the $\mathbf{m} \times \mathbf{n}$ constraint Jacobian matrix, $\boldsymbol{\lambda}$ is the $\mathbf{m}$ corresponding constraint forces, $\mathbf{F}$ is the $\mathbf{n}$ generalized forces that include external forces and inertia forces due to centrifugal acceleration, and $\ddot{\mathbf{u}}$ consists of the translational and rotational accelerations.

Note that, for each body $\dot{\mathbf{u}}$ consists of three translational velocity components expressed in the inertial frame and three angular velocity components expressed in the body-fixed frame. In other words, they are physical coordinates which are a particular set of generalized coordinates. In addition, due to the present representation of translational motion (referred to an inertial frame) and rotational motion (expressed in the convected frame), the task for identifying the dependent and independent coordinates for the system constraint equations becomes straightforward, thus leading to the development of the present natural partitioning scheme.

3. A Natural Partitioning Scheme

In this section, the Lagrange multipliers are eliminated from (2.1) and a set of second order differential equations are derived in terms of system independent coordinates. To determine the system independent coordinates, a natural partitioning scheme is proposed to efficiently construct the null space of the constraint Jacobian matrix. A parallel methodology is demonstrated if system topologies consist of a number of tree structures. For a system that contains closed-loops, a cut-joint technique is used so that the present scheme can be equally applied.

3.1 Constraint Elimination Method For DAE

In constraint elimination, the main task is to find a projection matrix $\mathbf{A}$ such that, when its transposed is post-multiplied by $\mathbf{B}^T\boldsymbol{\lambda}$, we have

$$\mathbf{A}^T\mathbf{B}^T\boldsymbol{\lambda} = 0 \tag{3.1.1}$$

This projection matrix can be obtained by expressing the physical velocity $\dot{\mathbf{u}}$ in terms of the independent velocities $\dot{\mathbf{u}}^i$ as

$$\dot{\mathbf{u}} = \mathbf{A}\dot{\mathbf{u}}^i \tag{3.1.2}$$

Time differentiation of (3.1.2) gives

$$\ddot{\mathbf{u}} = \mathbf{A}\ddot{\mathbf{u}}^i + \dot{\mathbf{A}}\dot{\mathbf{u}}^i \tag{3.1.3}$$

Substituting (3.1.2) into (2.3) yields

$$\mathbf{B}\dot{\mathbf{u}} = \mathbf{B}\mathbf{A}\dot{\mathbf{u}}^i = 0 \tag{3.1.4}$$

Since $\dot{\mathbf{u}}^i$ is a set of independent velocities and in general $\dot{\mathbf{u}}^i \neq 0$, (3.1.4) implies

$$\mathbf{B}\mathbf{A} = 0 \;\; ; \;\; \mathbf{A}^T\mathbf{B}^T = 0 \tag{3.1.5}$$

where $\mathbf{A}$ is called the null space of the constraint Jacobian matrix $\mathbf{B}$. Once $\mathbf{A}$ is constructed, pre-multiplication of (2.1) by $\mathbf{A}^T$ yields

$$\mathbf{A}^T\mathbf{M}\ddot{\mathbf{u}} + \mathbf{A}^T\mathbf{B}^T\boldsymbol{\lambda} = \mathbf{A}^T\mathbf{F} \tag{3.1.6}$$

By (3.1.1), the second term on the left hand side of (3.1.6) is equal to zero, hence the above equation reduces to

$$\mathbf{A}^T\mathbf{M}\ddot{\mathbf{u}} = \mathbf{A}^T\mathbf{F} \tag{3.1.7}$$

Substituting (3.1.3) into (3.1.7) yields the desired equations of motion in terms of their independent velocities $\dot{\mathbf{u}}^i$ as

$$\mathbf{A}^T\mathbf{M}\mathbf{A}\ddot{\mathbf{u}}^i = \mathbf{A}^T\mathbf{F} - \mathbf{A}^T\mathbf{M}\dot{\mathbf{A}}\dot{\mathbf{u}}^i \tag{3.1.8}$$

Once the right hand side of (3.1.8) is obtained, the system equations can be written in the following form :

$$\mathbf{M}^*\ddot{\mathbf{u}}^i = \mathbf{b} \tag{3.1.9}$$

where

$$\mathbf{M}^* = \mathbf{A}^T\mathbf{M}\mathbf{A} \tag{3.1.10}$$

$$\mathbf{b} = \mathbf{A}^T\mathbf{F} - \mathbf{A}^T\mathbf{M}\dot{\mathbf{A}}\dot{\mathbf{u}}^i \tag{3.1.11}$$

3.2 A Natural Partitioning Scheme For Open-Loop MBD Systems

To demonstrate the present natural partitioning scheme for open loop systems, a three-dimensional triple-pendulum problem (Fig. 1) is chosen. The constraint equations for this problem can be written as

$$\begin{bmatrix} [B_{11}] & 0 & 0 \\ [B_{21}] & [B_{22}] & 0 \\ 0 & [B_{32}] & [B_{33}] \end{bmatrix} \begin{Bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \end{Bmatrix} =$$

$$\begin{bmatrix} [-I] & [R_{s11}] & 0 & 0 & 0 & 0 \\ [I] & [R_{s21}] & [-I] & [R_{s22}] & 0 & 0 \\ 0 & 0 & [I] & [R_{s32}] & [-I] & [R_{s33}] \end{bmatrix} \begin{Bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \end{Bmatrix} = 0 \tag{3.2.1}$$

where the bodies in this pendulum problem are connected by three spherical joints and R_s are function of rotational operators and position vectors from the center of mass of each body to the position of their connecting joints. To obtain the necessary projection matrix $\mathbf{A}$, we start with the first row of (3.2.1) :

$$B_{11}\dot{u}_1 = [\ -I\ ,\ R_{s11}\ ]\ \dot{u}_1 = 0 \tag{3.2.2}$$

that can be partitioned into

$$[B_{11}^d | B_{11}^i] \begin{Bmatrix} \dot{u}_1^d \\ \dot{u}_1^i \end{Bmatrix} = 0 \tag{3.2.3}$$

or

$$B_{11}^d\dot{u}_1^d + B_{11}^i\dot{u}_1^i = 0 \tag{3.2.4}$$

where $B_{11}^d = -I$, $B_{11}^i = R_{s11}$, and d represents the dependent coordinates and i represents the independent coordinates. Since $|B_{11}^d| \neq 0$, the dependent velocity components of first body can be calculated as

$$\dot{u}_1^d = -{B_{11}^d}^{-1} B_{11}^i\dot{u}_1^i = P_1\dot{u}_1^i \tag{3.2.5}$$

where $P_1 = -{B_{11}^d}^{-1} B_{11}^i = R_{s11}$. The velocity vector of first body $\dot{u}_1$ can be written in terms of independent velocities $\dot{u}_1^i$ as

$$\dot{u}_1 = \begin{Bmatrix} \dot{u}_1^d \\ \dot{u}_1^i \end{Bmatrix} = \begin{pmatrix} P_1 \\ I \end{pmatrix} \dot{u}_1^i = Q_1\dot{u}_1^i \tag{3.2.6}$$

where $Q_1 = \begin{pmatrix} P_1 \\ I \end{pmatrix}$. Likewise, B_{22} of the second row of (3.2.1) can be partitioned into

$$B_{21}\dot{u}_1 + B_{22}^d\dot{u}_2^d + B_{22}^i\dot{u}_2^i = 0 \tag{3.2.7}$$

or

$$\dot{u}_2^d = -{B_{22}^d}^{-1}(B_{21}\dot{u}_1 + B_{22}^i\dot{u}_2^i) \tag{3.2.8}$$

for $|B_{22}^d| \neq 0$. Substituting (3.2.6) into (3.2.8) yields

$$\dot{u}_2^d = -{B_{22}^d}^{-1}(B_{21}Q_1\dot{u}_1^i + B_{22}^i\dot{u}_2^i) = R_1\dot{u}_1^i + R_2\dot{u}_2^i \tag{3.2.9}$$

where $R_1 = -{B_{22}^d}^{-1}B_{21}Q_1 = B_{21}Q_1$, and $R_2 = -{B_{22}^d}^{-1}B_{22}^i = B_{22}^i$. The velocity vector of second body, $\dot{u}_2$, can be expressed in terms of the independent velocities, $\dot{u}_1^i$ and $\dot{u}_2^i$, as

$$\dot{u}_2 = \begin{Bmatrix} \dot{u}_2^d \\ \dot{u}_2^i \end{Bmatrix} = \begin{bmatrix} R_1 & R_2 \\ 0 & I \end{bmatrix} \begin{Bmatrix} \dot{u}_1^i \\ \dot{u}_2^i \end{Bmatrix} = [S_1|S_2] \begin{Bmatrix} \dot{u}_1^i \\ \dot{u}_2^i \end{Bmatrix} \tag{3.2.10}$$

where $S_1 = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$ and $S_2 = \begin{pmatrix} R_2 \\ I \end{pmatrix}$. Applying the same procedure to the third row of (3.2.1), $\dot{u}_3^d$ can be expressed as

$$\begin{aligned} \dot{u}_3^d &= -B_{33}^d(B_{32}\dot{u}_2 + B_{33}^i\dot{u}_3^i) = -B_{33}^d[B_{32}(S_1\dot{u}_1^i + S_2\dot{u}_2^i) + B_{33}^i\dot{u}_3^i] \\ &= V_1\dot{u}_1^i + V_2\dot{u}_2^i + V_3\dot{u}_3^i \end{aligned} \tag{3.2.11}$$

where $V_1 = -{B_{33}^d}^{-1}B_{32}S_1 = B_{32}S_1$, $V_2 = -{B_{33}^d}^{-1}B_{32}S_2 = B_{32}S_2$, $V_2 = -{B_{33}^d}^{-1}B_{33}^i = B_{33}^i$, and $\dot{u}_3$ can be written in terms of $\dot{u}_1^i$, $\dot{u}_2^i$, and $\dot{u}_3^i$ as

$$\dot{u}_3 = \begin{Bmatrix} \dot{u}_3^d \\ \dot{u}_3^i \end{Bmatrix} = \begin{bmatrix} V_1 & V_2 & V_3 \\ 0 & 0 & I \end{bmatrix} \begin{Bmatrix} \dot{u}_1^i \\ \dot{u}_2^i \\ \dot{u}_3^i \end{Bmatrix} = [W_1|W_2|W_3] \begin{Bmatrix} \dot{u}_1^i \\ \dot{u}_2^i \\ \dot{u}_3^i \end{Bmatrix} \tag{3.2.12}$$

where $W_1 = \begin{pmatrix} V_1 \\ 0 \end{pmatrix}$, $W_2 = \begin{pmatrix} V_2 \\ 0 \end{pmatrix}$, and $W_3 = \begin{pmatrix} V_2 \\ I \end{pmatrix}$. Combining (3.2.6), (3.2.10), and (3.2.12), we construct the physical velocities $\dot{\mathbf{u}}$ in terms of $\dot{\mathbf{u}}^i$ as

$$\begin{Bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \end{Bmatrix} = \begin{bmatrix} Q_1 & 0 & 0 \\ S_1 & S_2 & 0 \\ W_1 & W_2 & W_3 \end{bmatrix} \begin{Bmatrix} \dot{u}_1^i \\ \dot{u}_2^i \\ \dot{u}_3^i \end{Bmatrix}. \tag{3.2.13}$$

or

$$\dot{\mathbf{u}} = \mathbf{A}\dot{\mathbf{u}}^i \tag{3.2.14}$$

where $\mathbf{A}$ is the null space of the constraint Jacobian matrix that has been exploited in the previous section. Note that in the process of forming $\mathbf{A}$, the inversion of the dependent matrices can be obtained analytically as opposed to the generalized coordinate partitioning scheme that the inversion of the dependent matrices have to be carried out numerically. The scheme for constructing $\mathbf{A}$ provides a guideline to deal with MBD systems containing different topologies such as multiple open kinematic links and closed kinematic loops, which will be discussed in the following sections.

3.3 Natural Partitioning Scheme For Multiple Open Chain Systems

If the MBD systems have more than one branch as shown in Fig. 2, the present scheme lends itself to multiprocessor computers. This property can be demonstrated by the following MBD system where the constraint equations are given by

$$\begin{bmatrix} [B_{11}] & 0 & 0 & 0 & 0 \\ [B_{21}] & [B_{22}] & 0 & 0 & 0 \\ 0 & [B_{32}] & [B_{33}] & 0 & 0 \\ [B_{41}] & 0 & 0 & [B_{44}] & 0 \\ 0 & 0 & 0 & [B_{54}] & [B_{55}] \end{bmatrix} \begin{Bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \\ \dot{u}_4 \\ \dot{u}_5 \end{Bmatrix} = 0 \tag{3.3.1}$$

Applying the proposed scheme, the **A** matrix is selected as

$$\begin{Bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \\ \dot{u}_4 \\ \dot{u}_5 \end{Bmatrix} = \begin{bmatrix} Q_1 & 0 & 0 & 0 & 0 \\ S_1 & S_2 & 0 & 0 & 0 \\ W_1 & W_2 & W_3 & 0 & 0 \\ Y_1 & 0 & 0 & Y_4 & 0 \\ Z_1 & 0 & 0 & Z_4 & Z_5 \end{bmatrix} \begin{Bmatrix} \dot{u}_1^i \\ \dot{u}_2^i \\ \dot{u}_3^i \\ \dot{u}_4^i \\ \dot{u}_5^i \end{Bmatrix} \tag{3.3.2}$$

Note that, in the natural partitioning scheme, once the first row of (3.3.2) is constructed, the second and fourth row of (3.3.2) can be constructed simultaneously according to given Q_1. Again, if the first, second, and fourth rows of (3.3.2) are found, the third and fifth rows of (3.3.2) can be obtained according to their dependent branches respectly. Since MBD systems are the systems that include many kinematic loops, it is natural to utilize this development in a multiprocessor computer to compute the null space (at each branch) of the constraint Jacobian matrix.

3.4 Natural Partitioning Scheme For Closed-Loop MBD Systems

When the systems have one or more closed loops, difficulty arises in constructing the null space of the constraint Jacobian matrix as one will see from examining the following three body crank-slider problem (Fig. 3). The constraint equations for this problem are given by

$$\begin{bmatrix} [B_{11}] & 0 & 0 \\ [B_{21}] & [B_{22}] & 0 \\ 0 & [B_{32}] & [B_{33}] \\ 0 & 0 & [B_{43}] \end{bmatrix} \begin{Bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \end{Bmatrix} = 0 \tag{3.4.1}$$

It is obvious that joint 1 and 4 conflict in determining the null space of (3.4.1) according to preceding scheme. Fortunely, there is a technique to overcome this difficulty. The technique is called "cut joints" which means cut the joints that are necessary to force the system topologies to become open loops so that the existing solution procedure could be adopted. This technique is accomplished by partitioning (3.4.1) into the following form

$$\begin{bmatrix} [B_{11}] & 0 & 0 \\ [B_{21}] & [B_{22}] & 0 \\ 0 & [B_{32}] & [B_{33}] \\ --- & --- & --- \\ 0 & 0 & [B_{43}] \end{bmatrix} \begin{Bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \end{Bmatrix} = \begin{Bmatrix} \mathbf{B}_o \\ \mathbf{B}_c \end{Bmatrix} \dot{\mathbf{u}} = 0 \tag{3.4.2}$$

or

$$\mathbf{B}_o \dot{\mathbf{u}} = 0, \quad \mathbf{B}_c \dot{\mathbf{u}} = 0 \tag{3.4.3}$$

where $\mathbf{B}_o$ represents the open loop constraint Jacobian matrix, and $\mathbf{B}_c$ represents the remaining constraint Jacobian matrix after the joints have been cut. Performing the natural partitioning scheme to construct the null space of $\mathbf{B}_o$ as

$$\mathbf{B}_o \mathbf{A}_o = 0 \ ; \ \mathbf{A}_o^T \mathbf{B}_o^T = 0 \tag{3.4.4}$$

Performing algebraical calculations as in section 3.1 yields the equations of motion for a closed-loop MBD system as

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{B}_o^T \boldsymbol{\lambda}_o + \mathbf{B}_c^T \boldsymbol{\lambda}_c = \mathbf{F} \tag{3.4.5}$$

Premultiplying $\mathbf{A}_o^T$ to above equation yields

$$\mathbf{A}_o^T \mathbf{M}\ddot{\mathbf{u}} + \mathbf{A}_o^T \mathbf{B}_c^T \boldsymbol{\lambda}_c = \mathbf{A}_o^T \mathbf{F} \tag{3.4.6}$$

which can be solved either by employing the penalty constraint stabilization technique (P.C.S.T.) or by constructing the null space for the new equations of motion.

4. A Solution Procedure for MBD Systems

A common procedure for solving DAE is to augment (2.1) and (2.4) into the following system of differential equations

$$\begin{bmatrix} \mathbf{M} & \mathbf{B}^T \\ \mathbf{B} & 0 \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{u}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \mathbf{F} \\ -\dot{\mathbf{B}}\dot{\mathbf{u}} \end{Bmatrix} \tag{4.1}$$

so that numerical ordinary differential equation solvers can be applied. The drawbacks of this approach are : first, (2.4) does not represent the original constraint equations (2.2) ; second, the violation of the constraints occurred during the process of numerical integration. A constraint stabilization technique proposed by Baumgarte can be used to stabilize (4.1). The disadvantages of this technique have been studied and a new stabilized technique has been developed in [12,13] so that constraint violation can be stabilized efficiently. An alternative approach to avoid constraint violation is to obtain the null space of the constraint Jacobian matrix as suggested in the present scheme. *De Jalon et al.* have developed a formulation using the so called *natural coordinates* so that similar equations of motion to (3.1.8) are obtained. The drawbacks of their approach has been discussed in [10,11]. A solution that avoids these drawbacks can be achieved by augmenting (3.1.7) and (2.3) into

$$\begin{bmatrix} \mathbf{A}^T\mathbf{M} \\ \mathbf{B} \end{bmatrix} \ddot{\mathbf{u}} = \begin{Bmatrix} \mathbf{A}^T\mathbf{F} \\ -\dot{\mathbf{B}}\dot{\mathbf{u}} \end{Bmatrix} \tag{4.2}$$

which not only destroys the symmetry of the matrix in (4.2) but also violates the constraint conditions when time integration algorithms are used. The following section discusses an approach that overcomes these difficulties with parallel computation in mind.

4.1 Application of Parallel Computations

Since MBD systems may involve hundreds of bodies, solution for such systems require large amounts of computations. For the purpose of real-time simulation, existing parallel computers need to be utilized and new numerical algorithms need to be developed in order to speedup the solution process. So, instead of solving the second order differential equations (3.1.8), we augment (3.1.3) and (3.1.7) into the following form :

$$\begin{bmatrix} -\mathbf{M} & \mathbf{MA} \\ \mathbf{A}^T\mathbf{M} & 0 \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{u}} \\ \ddot{\mathbf{u}}^i \end{Bmatrix} = \begin{Bmatrix} -\mathbf{M}\dot{\mathbf{A}}\dot{\mathbf{u}}^i \\ \mathbf{A}^T\mathbf{F} \end{Bmatrix} \tag{4.3}$$

Following [24,25], we can partition $\mathbf{M}$, $\ddot{\mathbf{u}}$, and $\mathbf{MA}$ into the the following form

$$\begin{bmatrix} M_{(1,1)} & 0 & 0 & 0 & \cdots & D_{(1,n+1)} \\ 0 & M_{(2,2)} & 0 & 0 & \cdots & D_{(2,n+1)} \\ 0 & 0 & M_{(3,3)} & 0 & \cdots & D_{(3,n+1)} \\ 0 & 0 & 0 & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & M_{(n,n)} & \cdots \\ D_{(n+1,1)} & D_{(n+1,2)} & D_{(n+1,3)} & \cdots & \cdots & 0 \end{bmatrix} \begin{Bmatrix} \ddot{u}_1 \\ \ddot{u}_2 \\ \ddot{u}_3 \\ \cdots \\ \ddot{u}_n \\ \ddot{\mathbf{u}}^i \end{Bmatrix} = \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ \cdots \\ c_n \\ d \end{Bmatrix} \tag{4.4}$$

where n is the total number of bodies in the system. The above system with an *arrow head matrix* (4.4) can be written as

$$\begin{aligned} &-\mathbf{M}_j\ddot{\mathbf{u}}_j + \mathbf{D}_{(j,n+1)}\ddot{\mathbf{u}}^i = c_j, \quad j = 1,...,n \\ &\sum_{j=1}^{n} \mathbf{D}_{(n+1,j)}\ddot{\mathbf{u}}_j = d \end{aligned} \tag{4.5}$$

where

$$\sum_{j=1}^{n} \mathbf{D}_{(n+1,j)} = \sum_{j=1}^{n} \mathbf{A}_j^T\mathbf{M}_j$$

$$\mathbf{D}_{(j,n+1)} = \mathbf{M}_j\mathbf{A}_j, \quad j = 1,...,n$$

$$c_j = -(\mathbf{M}\dot{\mathbf{A}}\dot{\mathbf{u}}^i)_j, \quad j = 1,...,n$$

$$d = \sum_{j=1}^{n} \mathbf{A}^T\mathbf{F}_j$$

Each diagonal submatrix $\mathbf{M}_j$ represents the local mass matrix which is decoupled and can be factorized concurrently. An off-diagonal submatrix $\mathbf{D}_j$ denotes the coupling between two connecting bodies in the system. Since $\mathbf{M}$ is a constant matrix, (4.5a) becomes

$$\ddot{\mathbf{u}}_j = \mathbf{M}_j^{-1}(\mathbf{D}_{(j,n+1)}\ddot{\mathbf{u}}^i - c_j) \tag{4.6}$$

Substituting (4.6) into (4.6b) gives the well-known *Schur complement*

$$\sum_{j=1}^{n} \mathbf{D}_{(n+1,j)}\mathbf{M}_j^{-1}\mathbf{D}_{(j,n+1)})\ddot{\mathbf{u}}^i = \sum_{j=1}^{n} \mathbf{D}_{(n+1,j)}\mathbf{M}_j^{-1}\mathbf{F}_j - \sum_{j=1}^{n} \mathbf{D}_{(n+1,j)}\dot{\mathbf{A}}\dot{\mathbf{u}}^i \tag{4.7}$$

where (3.1.8) is recovered. Several aspects of the present procedure have been observed :

(1) The parallelism in the multibody system is exploited by mapping each processor onto a group of bodies so that independent computations such as the left hand side of (4.7) can be performed concurrently.
(2) Since M_i is a constant mass matrix, it needs to be factored only once.
(3) To solve for $\ddot{\mathbf{u}}^i$, a parallel sparse solver such as described in [25] may be utilized.
(4) Once $\ddot{\mathbf{u}}^i$ is obtained, the evaluation of $\ddot{\mathbf{u}}$ from (4.6) is trivially parallelized.

4.2 Parallel Solution Procedure for MBD Systems

The solution procedure using the natural partitioning scheme can be summarized with the following steps :

[1] Construct $\mathbf{A}$ at step n.
[2] Solve (4.3) at step n for $\ddot{\mathbf{u}}$, and $\ddot{\mathbf{u}}^i$.
[3] Integrate translational and angular velocities from n to $n+1$ by using $\ddot{\mathbf{u}}$, and $\ddot{\mathbf{u}}^i$.
[4] Integrate translational displacements and angular orientations from n to $n+1$ by using $\dot{\mathbf{u}}$, and $\dot{\mathbf{u}}^i$.

It is known that current MBD programs, which are developed in the last twenty years, were tailored for sequential computers with core memory limitations. Limited core memory is an issue motivating researchers to develop sparse matrix method that will dramatically decrease computer storage. In selecting a solution scheme from a multiprocessing system, iterative solution methods are often preferred over direct methods because they require fewer synchronization and / or interprocessor communication. Most studies of MBD algorithms often assume that the system equations have already been formed. As indicated in (4.5), the system equations can be generated independently and in parallel. It would be natural if the solution scheme can be processed at body-by-body level without forming the system equations. Among the iterative solution methods, the conjugate gradient method appears to be the most promising candidate because of its inherent parallelism [24-26]. The following parallel PPCG scheme, which is specified to MBD systems, is summarized into two steps with (4.1.9) as the system equations :

(1) Solve in parallel using all the processors $\mathbf{M}^*\ddot{\mathbf{u}}^i = \mathbf{b}$

- Form the right hand side of the Schur complement :

  For $j = 1$ to N_p do concurrently

  Form $T_r(j) = M(j)^{-1}c(j)$

  Form $b(j) = d(j) - D(j)T_r(j)$

- Initialize :

  $x_0 = 0$

  $r_0 = b$

For $k = 1, ..., n$

  If $r_{k-1} = 0$ then quit

  Else

- Compute new conjugate search direction :

  Solve $\mathbf{P}z_{k-1} = r_{k-1}$ for z_{k-1}

  $\beta_k = z_{k-1}^T r_{k-1} / z_{k-2}^T r_{k-2} \quad (\beta_1 = 0)$

  $p_k = z_{k-1} + \beta_k p_{k-1} \quad (p_1 = z_0)$

- Form the left hand side of the Schur complement :

  For $j = 1$ to N_p do concurrently

  Form $T_l(j) = D^T(j)p_k(j)$

  Form $T_l(j) = M(j)^{-1}T_l(j)$

  Form $\mathbf{M}(j)^* p_k(j) = -D(j)T_l(j)$

- Line search to update solution and residual :

  $\alpha_k = z_{k-1}^T r_{k-1} / p_k^T \mathbf{M}^* p_k$

  $x_k = x_{k-1} + \alpha_k p_k$

  $r_k = r_{k-1} - \alpha_k \mathbf{M}^* p_k$

  Endif

(2) Broadcast the part of x corresponding to the handled rows of $\mathbf{D}$ to neighboring processors and solve for $\ddot{\mathbf{u}}$ as in the following steps :

For j = 1 to N_p do concurrently

- Receive x
- Back substitute for $\ddot{\mathbf{u}}$
- Send $\ddot{\mathbf{u}}$ to host for output

As noted in (4.7), the conjugate gradient method is used to obtain system independent variables without forming the null space matrix of the constraint Jacobian matrix. The reason is that the major operation of the conjugate gradient involves the multiplication of

a matrix by a trial vector. Thus, we can rewrite (4.7) as

$$\begin{aligned} v &= BM^{-1}B^T p \\ &= \sum_{j=1}^{n} B_{(n+1,j)} M_j^{-1} B_{(j,n+1)} p \\ &= \sum_{j=1}^{n} B_{(n+1,j)} M_j^{-1} p^e \\ &= \sum_{j=1}^{n} B_{(n+1,j)} v^e \end{aligned} \tag{4.8}$$

where $v^e = [v^{(1)}, v^{(2)}, ..., v^{(p)}]$. This multiplication is performed in three steps, and they add different contributions from prospective bodies to the entry of the resulting vector. The matrix-vector multiplications are performed directly on the body level and resulted in the global vector v.

Preconditioning can be used to accelerate the convergence of the conjugate gradient method. This is achieved by solving the modified system

$$PM^*x = Pb \tag{4.9}$$

where P is the preconditioning matrix. Selection of an optimal preconditioner for present MBD problems will be addressed in future work.

A prototype code for dynamics analysis of MBD systems on a shared-memory multiprocessor is currently under development at Center for Space Structures and Controls (CSSC). The software architecture and the numerical algorithm presented in this paper are part of the code. A test version called PMBS (Parallel Multi-Body System) has been implemented on the Alliant FX/8 by using *Force* macros [29]. Several example problems have been experimented and the results will be shown in the following section.

5. Numerical Examples

Computer simulation of two MBD systems has been examined in this section by using the scheme and the algorithm developed in previous sections. The resulting robust algorithm solves the present equations of motion of any arbitrary system topologies.

5.1 Three Dimensional Three-Link Manipulator

In order to validate the feasibility, effectiveness, and accuracy of the present scheme, a three-link manipulator, which has been studied by Gawronski and Ih [27,28], was performed under the given specifications. The manipulator is under a specified nonholonomic tip velocity constraints throughout the whole simulation as shown in Fig. 4. The joints that connected the link are modeled as spherical and revolute joints. Initially, the Lagrange multipliers are introduced to enforce the joint constraints as well as the nonholonomic constraints at the tip of the manipulator. The Lagrange multipliers are then eliminated by adopting the present scheme so that the numerical algorithms can be performed. When

time stepping, the manipulator is maneuvering under the desired trajectories which are given in two different vertical planes as illustrated in Fig. 5 and 6. The corresponding joint velocities and accelerations, which are matched quite closely to the results that are given by Gawronski and Ih, are shown in Fig. 7 and 8. Numerical experiments, although not reported herein, show the present scheme and algorithm provide considerably less CPU time than the one with the penalty constraint stabilization technique due to the number of the operation counts. These will play an important role in the real-time simulation.

5.2 Double-Wishbone Auto-Suspension Systems

To explore the parallelism of the present scheme, we select a vehicle model with multiple suspension systems, in which the input data describing this system are provided by Nikravesh of the University of Arizona, as shown in Fig. 9. According to the scheme used in section 3, the vehicle can be easily partitioned into four subsystems where four independent processors can be assigned to each of the subsystem so that the null space of the constraint Jacobian matrix can be constructed in parallel. Note that the suspension systems possess four sets of springs and dampers with given locations, spring and damping coefficients. The tires of the vehicle are modeled by using unilateral spring elements. Initially, the vehicle is positioned in a height of one meter from the ground with initial velocities equal to zero. When the vehicle is been released, gravity acts as the external loads that force the vehicle to fall. Fig. 10 illustrates one of the spring that reacts to the given external load during one second simulation run time. The displacements of each body, which simulate the behavior of the bodies in this system, are given in Fig. 11-15. The interesting features of this simulation are the CPU time consumption (Fig. 16) and the speed-up (Fig. 17) of using different processors in Alliant FX/8. Note that present scheme (N.P.S.) has been used to compare the results that have produced by previous developed penalty constraint stabilization technique.

6. Conclusion

An efficient numerical method for the dynamic analysis of MBD systems has been presented. A scheme that requires less CPU time to generate the null space for the constraint Jacobian matrix has been developed. The present scheme, which is robust for kinematic chains with variable degrees of freedom, provides the system independent coordinates that can be integrated without violating the kinematic constraint conditions. A parallel preconditioned conjugate gradient is also developed to solve the system governing equations of motion which are written in the Schur complement form so that parallel computations can be applied. Finally, the application of two example problems, dealing with holonomic and nonholonomic constraints, show the generality of the scheme and its capability for a general purpose computer program for the dynamic analysis of MBD systems.

References

1. Bodley, C. S., Devers, A. D., Park, A. C., and Frish, H. P., "A Digital Computer Program for the Dynamic Interaction Simulation of Controls and Structures (DISCO)," NASA Technical Paper 1219, May 1978.

2. Chace, M. A., and Smith, D. A., "DAM-Digital Computer Program for the Dynamic Analysis of Generalized Mechanical Systems," SAE Paper No. 710244, Jan. 1971

3. Sheth, P. N., and Uicker, J. J., "IMP(Integrated Mechanism Program): A Computer-Aided Design Analysis System for Mechanisms and Linkages," ASME Journal of Engineering for Industry, Vol. 94, 1972, pp. 454-464.

4. Paul, B., "Analytical Dynamics of Mechanisms-A Computer Oriented Overview," Mechanism and Machine Theory, Vol. 10, No. 6, 1975, pp. 481-507.

5. Schiehlen, W. O., "Dynamics of Complex Multibody Systems," SM Archives, Vol. 9, 1984, pp. 159-195.

6. Orlandea, N., M. A. Chace, and D. A. Calahan, "A Sparsity-Oriented Approach to Dynamic Analysis and Design of Mechanical Systems - Part 1 and 2," ASME J. Engr. for Industry, Vol. 99, Aug. 1977, pp. 773-784.

7. Wehage, R. A., and E. J. Haug, "Generalized Coordinate Partitioning of Dimension Reduction in Analysis of Constrained Dynamic Systems," ASME J. Mech Design, Vol. 104, Jan. 1982, pp. 247-255.

8. Nikravesh, P. E., and O. K. Kwon, "Euler Parameters in Computational Kinematics and Dynamics, Part I amd II," ASME J. Mech. Tran. and Auto. in Design, Vol. 107, No. 3, Sep. 1985, pp. 358-369.

9. Nikravesh P. E., *Computer-Aided Analysis of Mechanical System*, Prentice Hall, 1988.

10. Garcia de Jalon, J., Unda, J., Avello, A., and Jimenez, J. M., "Dynamic Analysis of Three-dimensional Mechanisms in 'Natural' Coordinates," ASME J. Mech. Tran. and Auto. in Design, Vol. 109, Dec. 1987, pp. 460-465.

11. Unda, J., Garcia de Jalon, J., Losantos, F., and Enparantza, R., "A Comparative Study on Some Different Formulations of the Dynamic Equations of Constrained Mechanical Systems," Transactions of the ASME, Vol. 109, Dec. 1987, pp. 466-474.

12. Park, K. C., and Chiou, J. C., "Evaluation of Constraint Stabilization Procedures for Multibody Dynamical Systems," Proc. the 28th Structures, Structural Dynamics and Materials Conf., 1987, Part 2A, Monterey, CA, AIAA Paper No. 87-0927, pp. 769-773.

13. Park, K. C., and Chiou, J. C., "Stabilization of Computational Procedures for Constrained Dynamical System," Journal of Guidance, Control and Dynamical System, 1988, Vol. 11, No. 4, 3 pp. 65-370.

14. Park, K. C., Chiou, J. C., and Downer, J. D., "A Computational Procedure for Large Rotational Motions in Multibody Dynamics," Proc. the 29th Structures, Structural Dynamics and Material Conf., Part 3, 1988, Williamsburg, Va, AIAA Paper No. 88-2416, pp. 1593-1601.

15. Park, K. C., Chiou, J. C., and Downer, J. D., "An Explicit-Implicit Staggered Procedure for Multibody Dynamics Analysis, Part 1: Algorithm Aspects," Report No. CU-CSSC-88-08, Center for Space Structures and Controls, University of Colorado.
16. Park, K. C., Chiou, J. C., and Downer, J. D., "An Explicit-Implicit Staggered Procedure for Multibody Dynamics Analysis, Part 2: Implementation and Evaluations," Report No. CU-CSSC-88-09, Center for Space Structures and Controls, University of Colorado.
17. Baumgarte, J. W., "Stabilization of Constraints and Integrals of Motion in Dynamical System," Comp. Meth. Appl. Mech. Engr., 1, 1972, pp. 1-16.
18. Baumgarte, J. W., "A New Method of Stabilization for Holonomic Constraints," Journal of Applied Mechanics, 50, 1983, pp. 869-870.
19. Bayo, E., "A Modified Lagrangian Formulation for the Dynamic Analysis of Constrained Mechanical Systems," Comp. Meth. in Appl. Mech. and Eng. 71, 1988, pp. 183-195.
20. Walton, W. C., and Steeves, E. C., "A New Matrix Theorem and Its Application for Establishing Independent Coordinates for Complex Dynamical Systems with Constraints," NASA TR-R326, 1969.
21. Singh, R. P., and Likins, P. W., "Singular Values Decomposition for Constrained Dynamical Systems," Journal of Applied Mechanics, Vol. 52 Dec. 1985, pp. 943-948.
22. Liang, C. G., and Lance, G. M., "A Differentiable Null Space Method for Constrained Dynamic Analysis," J. Mech. Tran. and Auto. in Design, Vol. 109, Sep. 1987, pp. 405-411.
23. Lanczos, C., *The Variational Principle of Mechanics*, Dover, 1970.
24. Farhat, C., and Wilson, E., "A New Finite Element Concurrent Computer Program Architecture," International Journal for Numerical Methods in Engineering, Vol. 24, 1987, pp. 1771-1792.
25. Farhat, C., Crivelli, L., "A General Approach to Nonlinear FE Computations on Shared-Memory Multiprocessors," Comp. Meth. in App. Mech. and Engr. 72, 1989, pp. 153-171.
26. Hughes, T. J. R., Ferencz, R. M. and Hallquist, J. O., "Large-Scale Vectorized Implicit Calculations in Solid Mechanics on A Cray X-MP/48 Utilizing EBE Preconditioned Conjugate Gradients," Comp. Meth. in App. Mech. and Engr. 61, 1987, pp. 215-248.
27. Gawronski, W., Ih, C-H, C.,"3D Rigid Body Dynamic Modeling of Space Crane for Control Design and Analysis," JPL D-6878, Nov. 17, 1989.
28. Craig, J. J., *Robotics, Mechanics and Control*, Addison-Wesley, 1986.
29. Jordan, H., Benton M. and Arenstorf, N., *Force User's Manual*, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO, 1987.

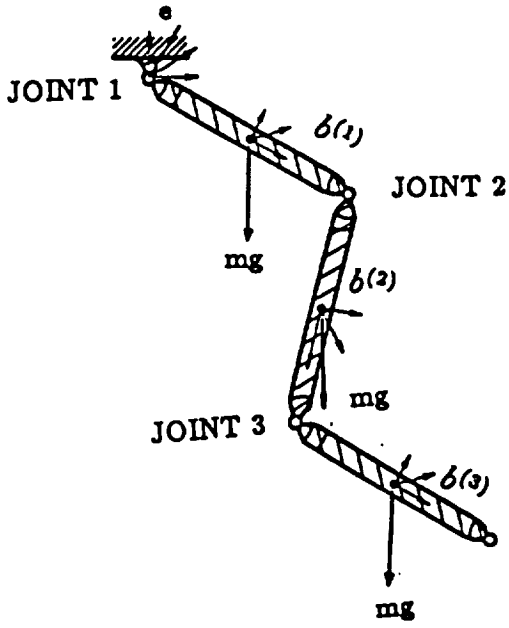


Fig. 1 The Triple Pendulum Problem

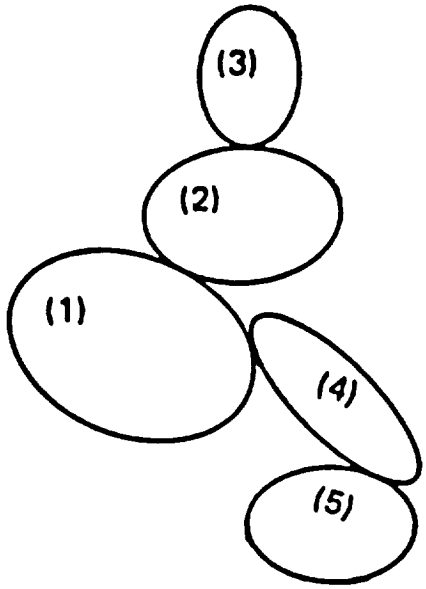


Fig. 2 Example of MBD Systems with Multiple Branches

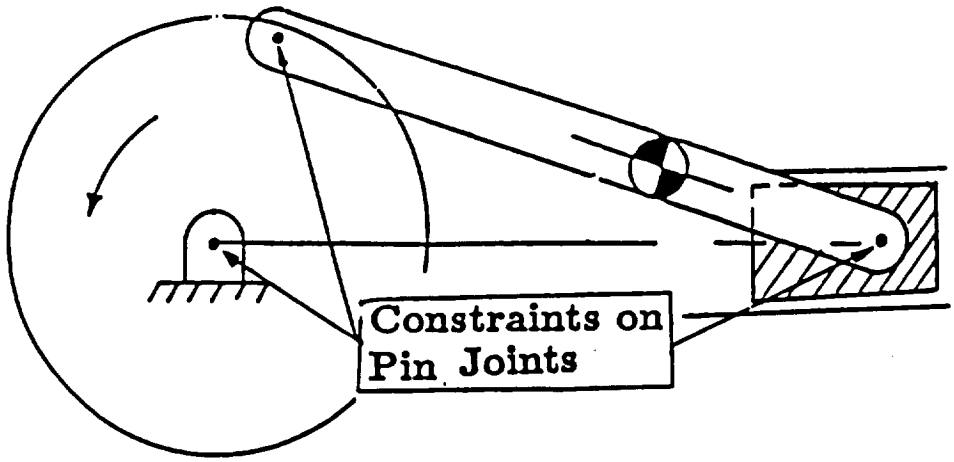


Fig. 3 The Crank-Slider Problem

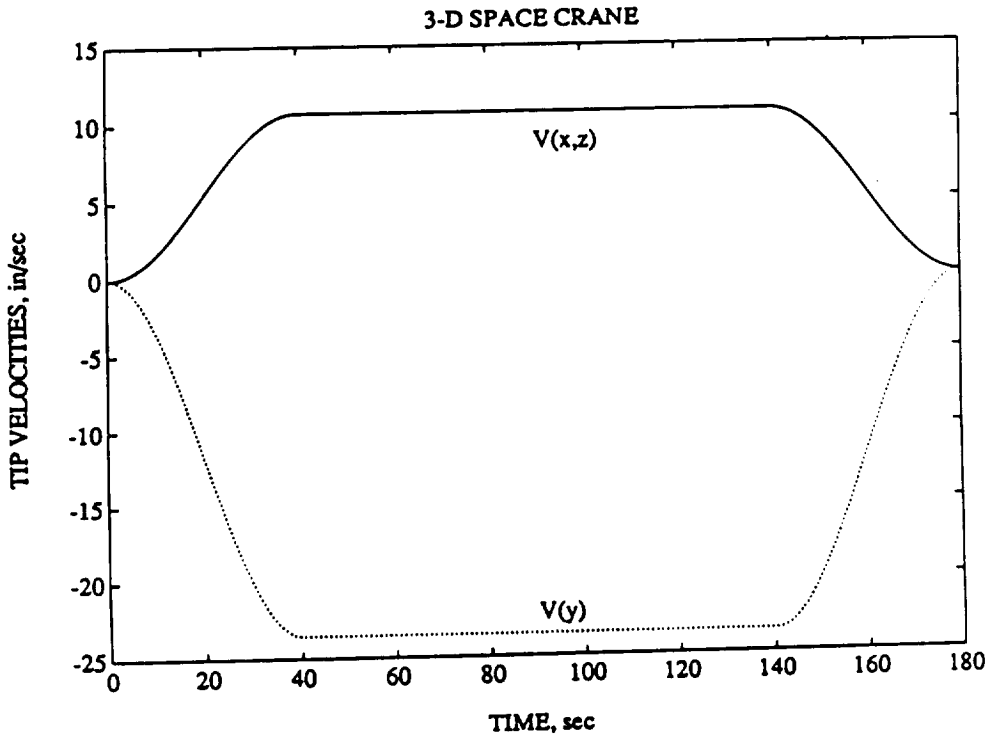


Fig. 4 Tip Velocity for the Three-Link Manipulator

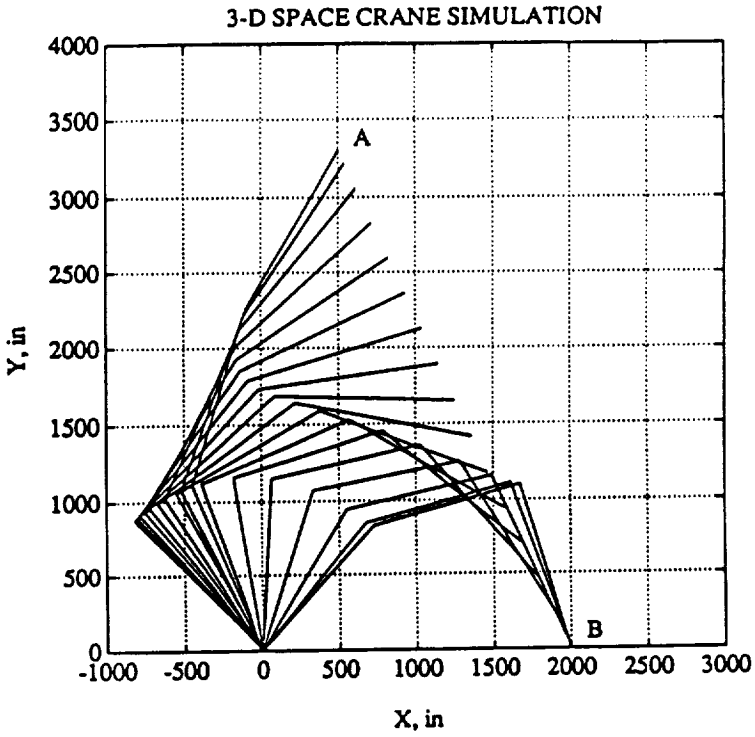


Fig. 5 Manipulator Configurations Along the X-Y Plane

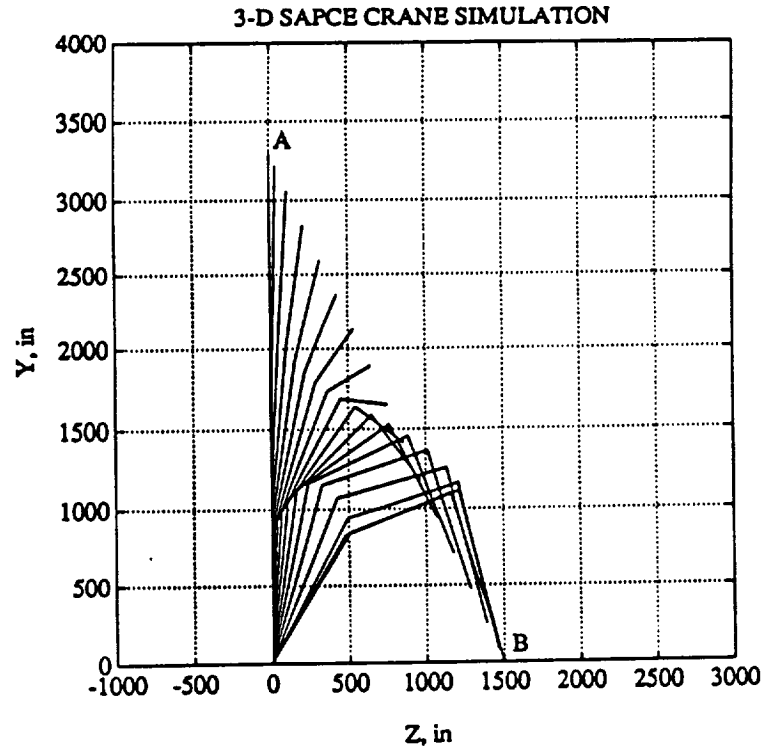


Fig. 6 Manipulator Configurations Along the Z-Y Plane

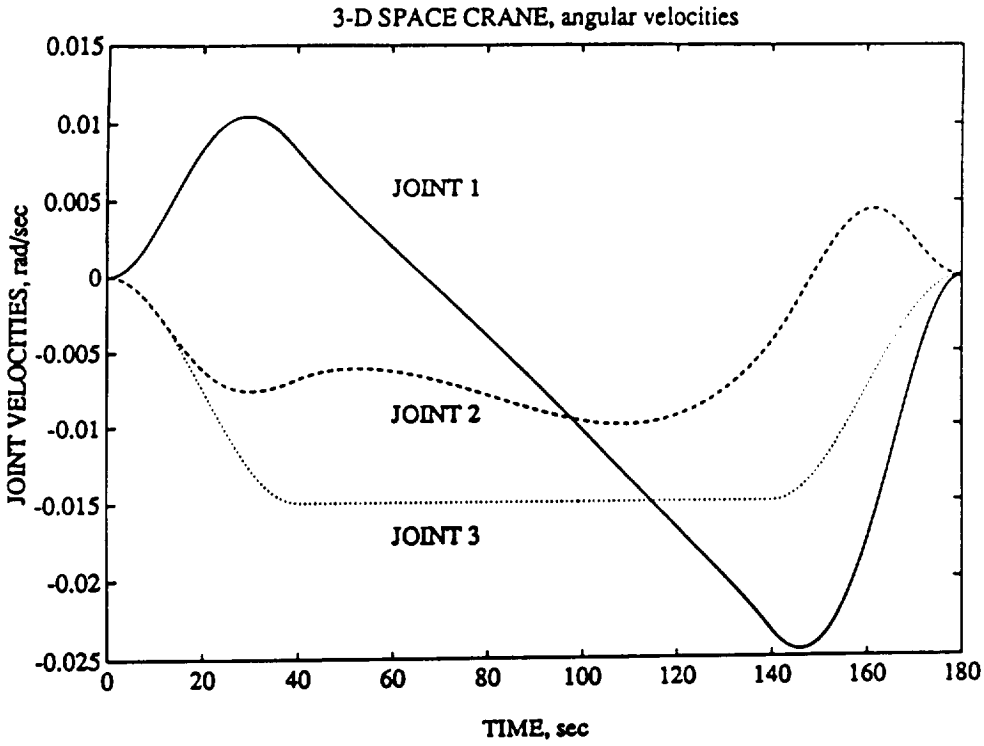


Fig. 7 Joint Angular Velocities for T = 180 sec

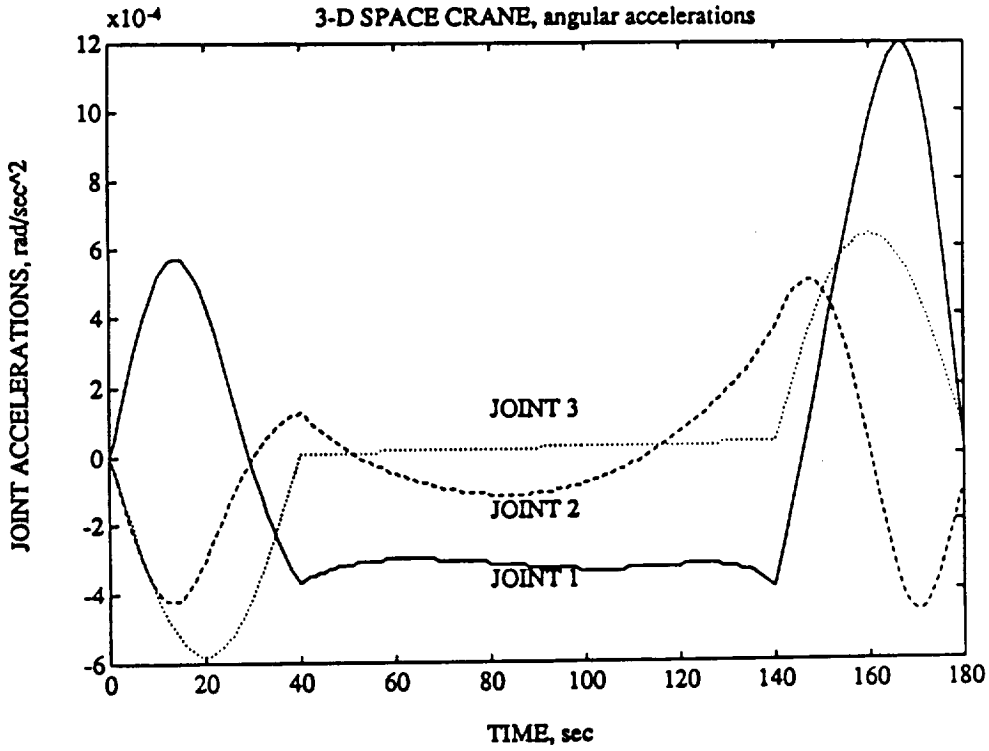


Fig. 8 Joint Angular Accelerations for T = 180 sec

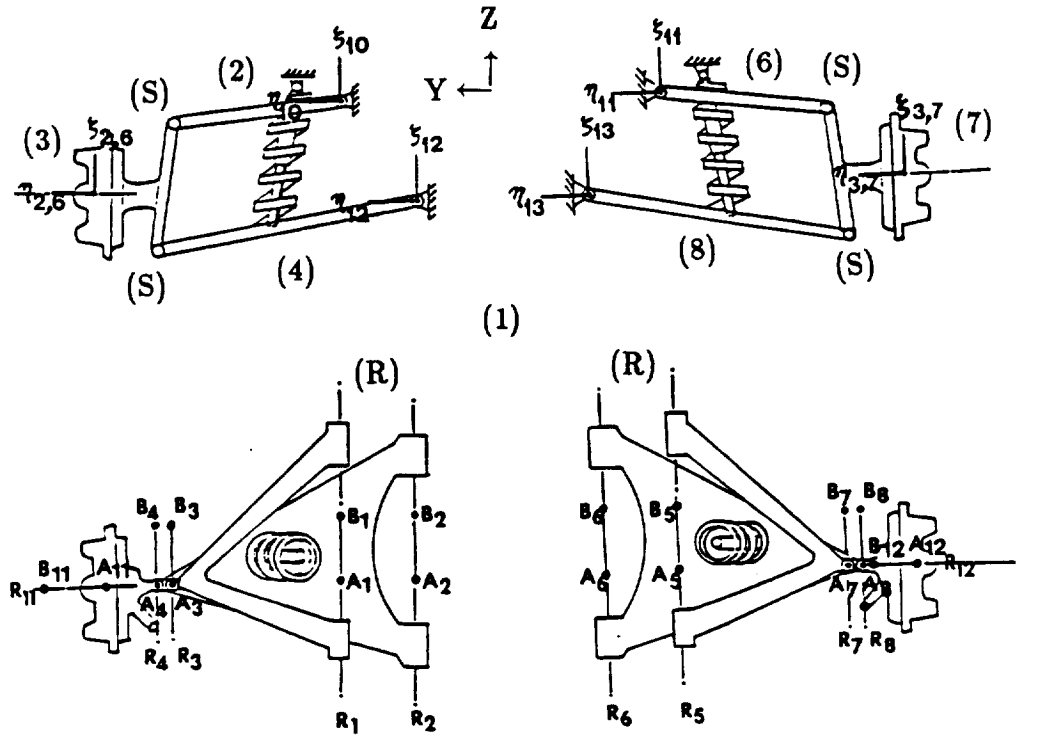


Fig. 9 Revolute and Spherical joints connecting different bodies of the system. Top and rear views : (a) front and (b) rear suspension systems

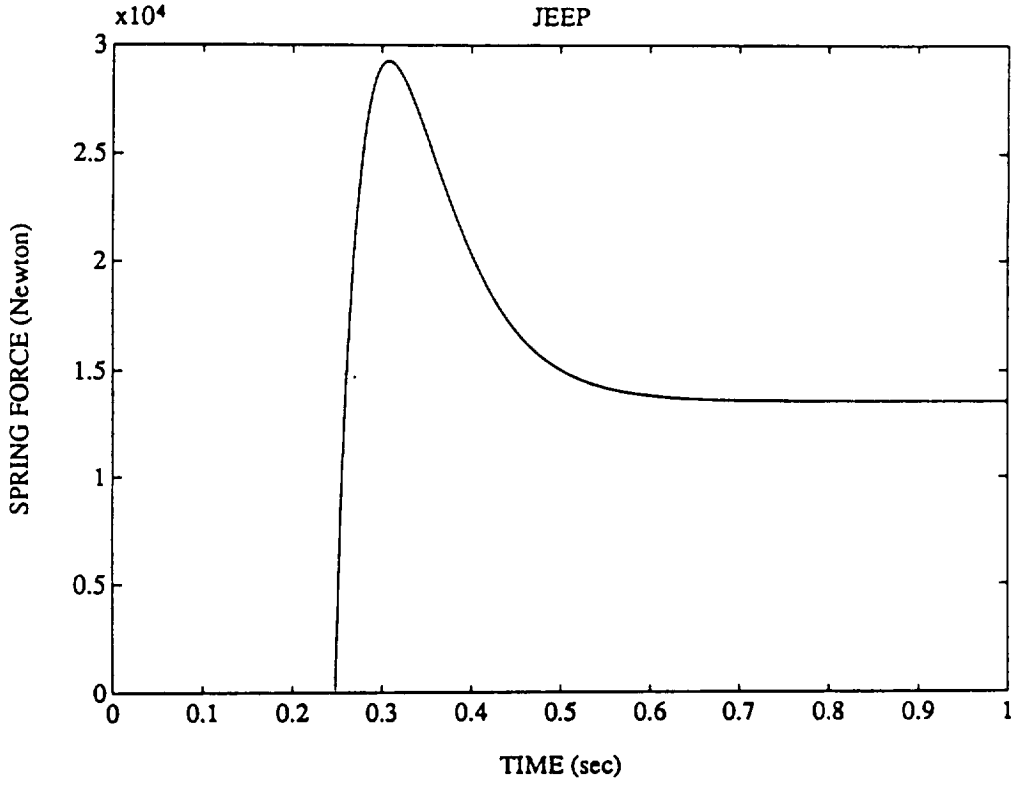


Fig. 10 Force Storage in Front Springs

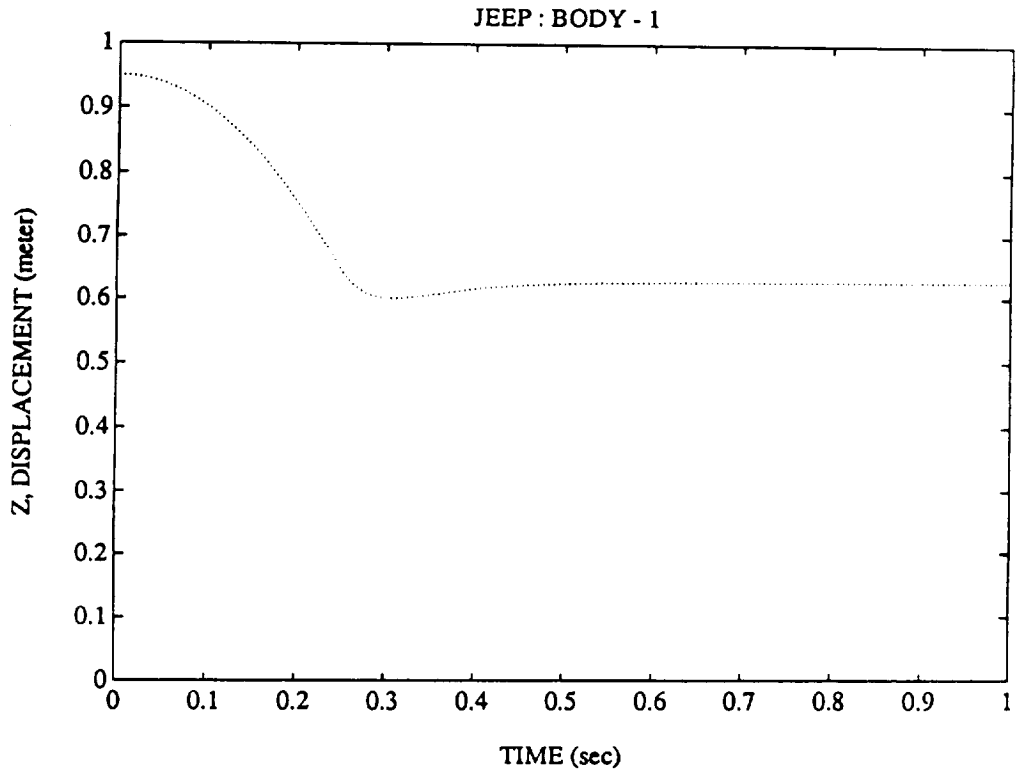


Fig. 11 Displacement History of Body 1

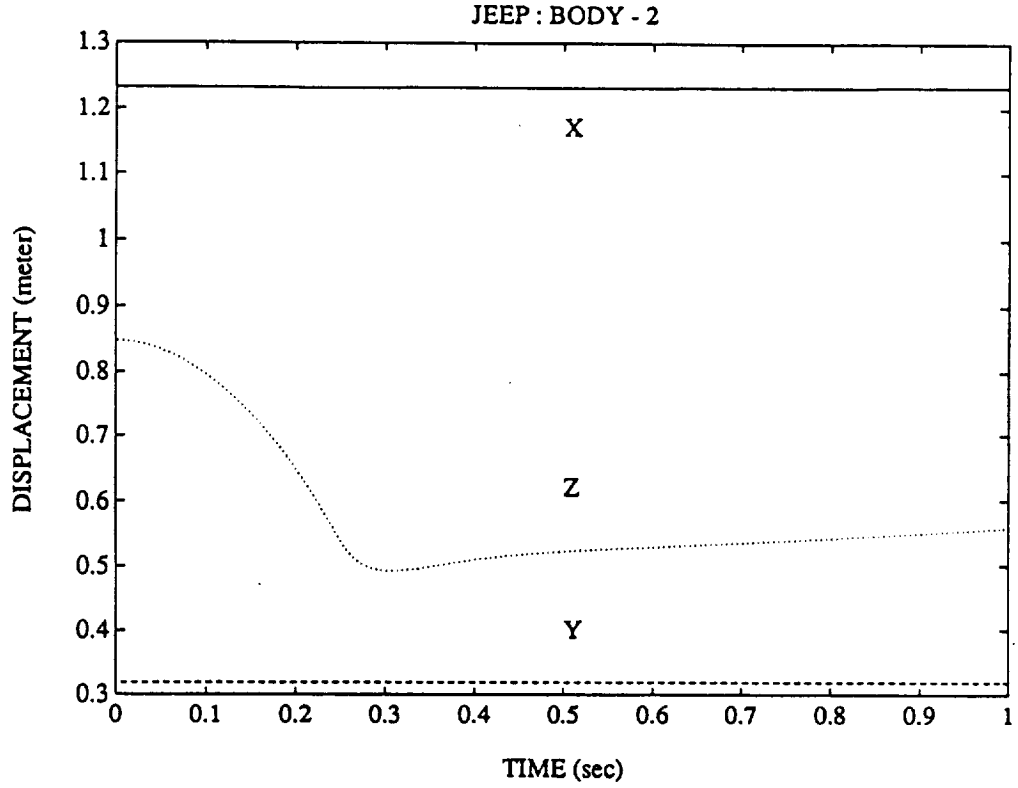


Fig. 12 Displacement History of Body 2

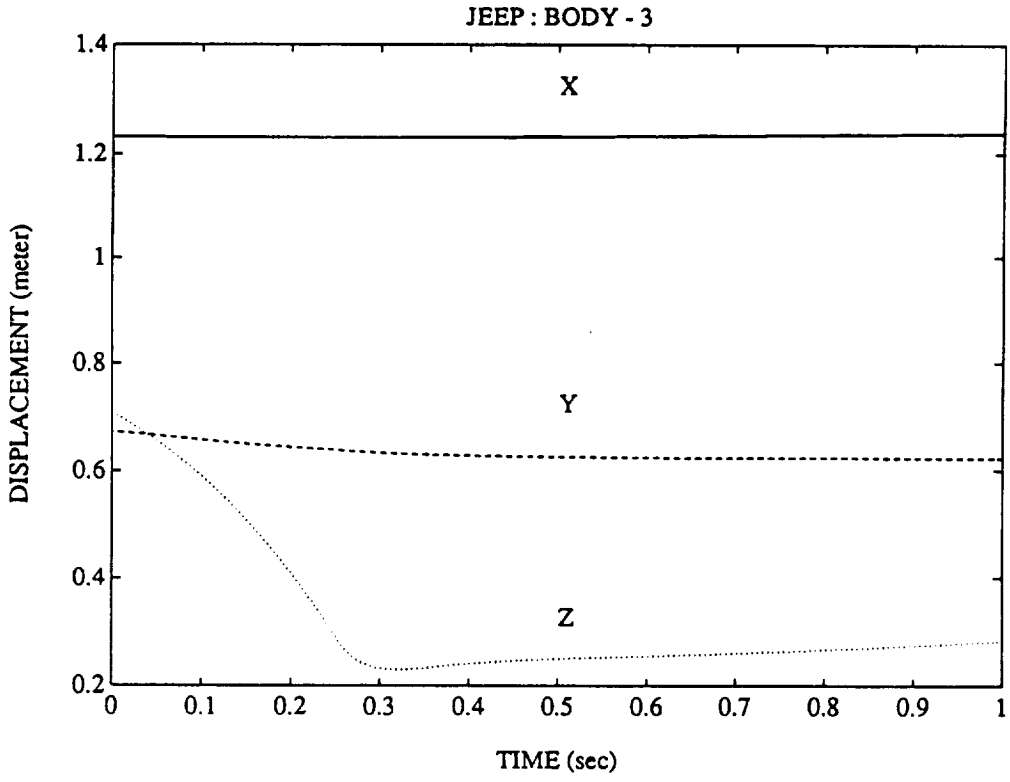


Fig. 13 Displacement History of Body 3

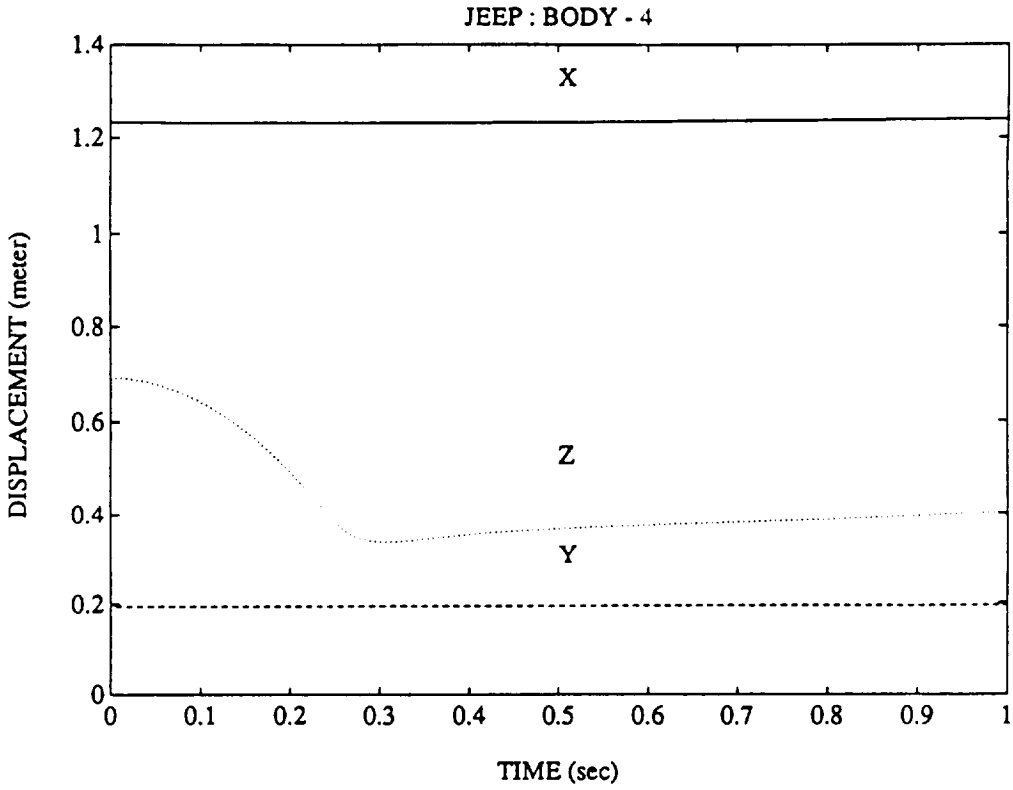


Fig. 14 Displacement History of Body 4

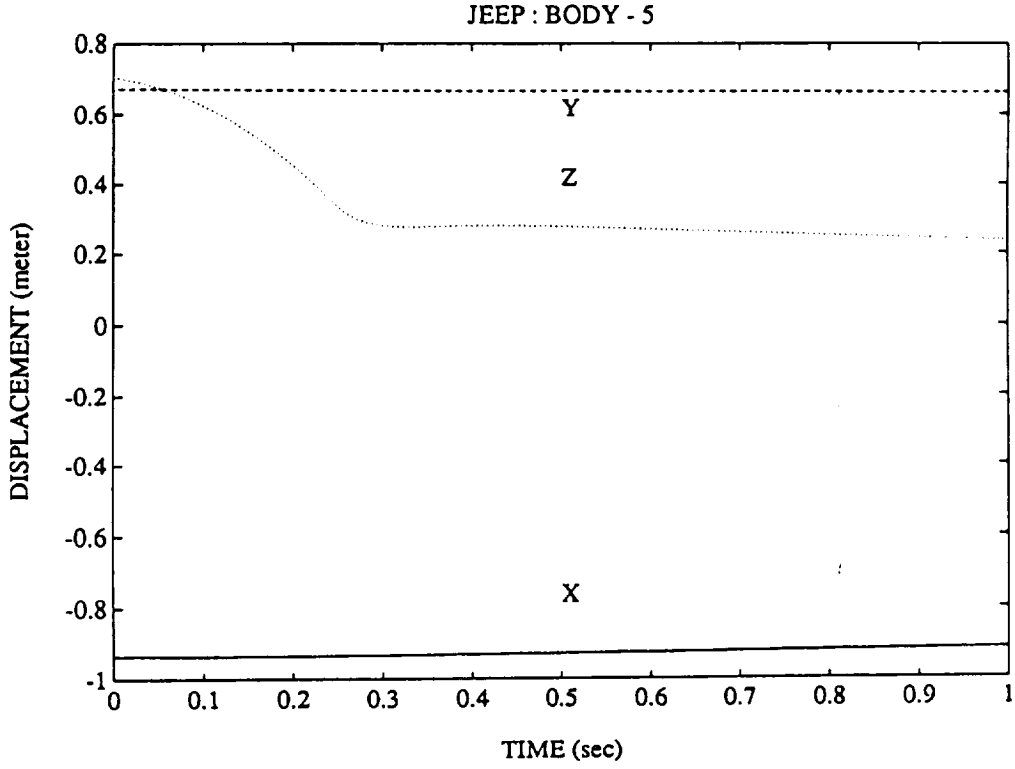


Fig. 15 Displacement History of Body 5

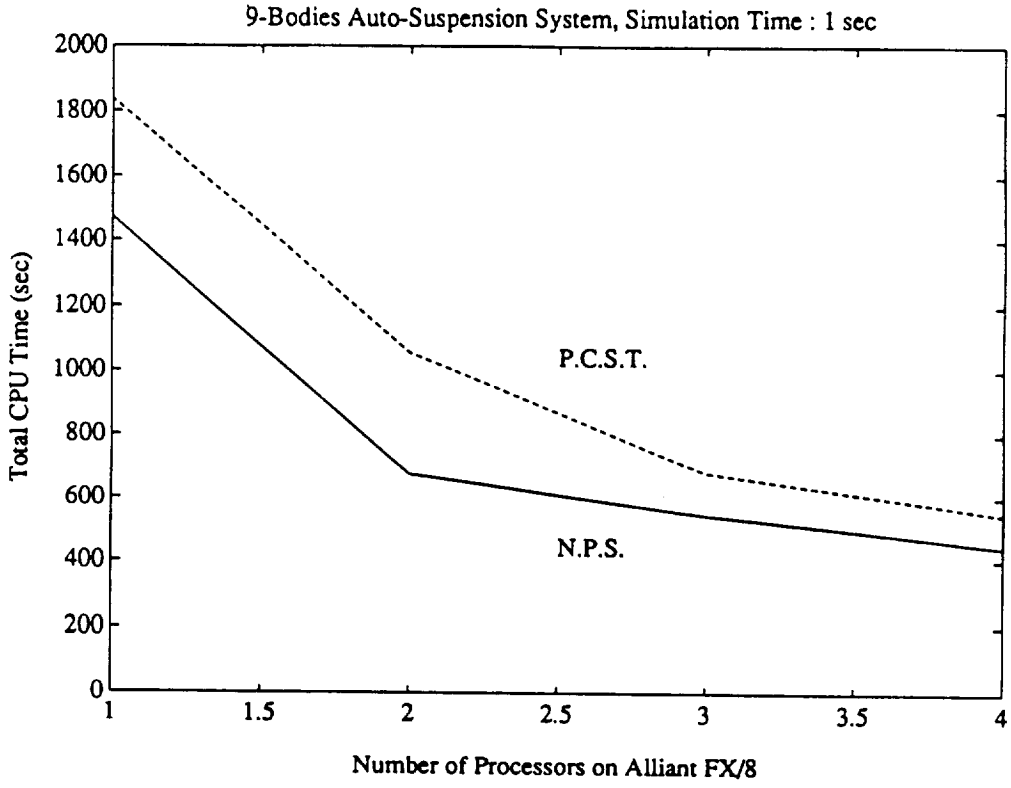


Fig. 16 Comparison of Total CPU Time Used by Both Techniques on Alliant FX/8

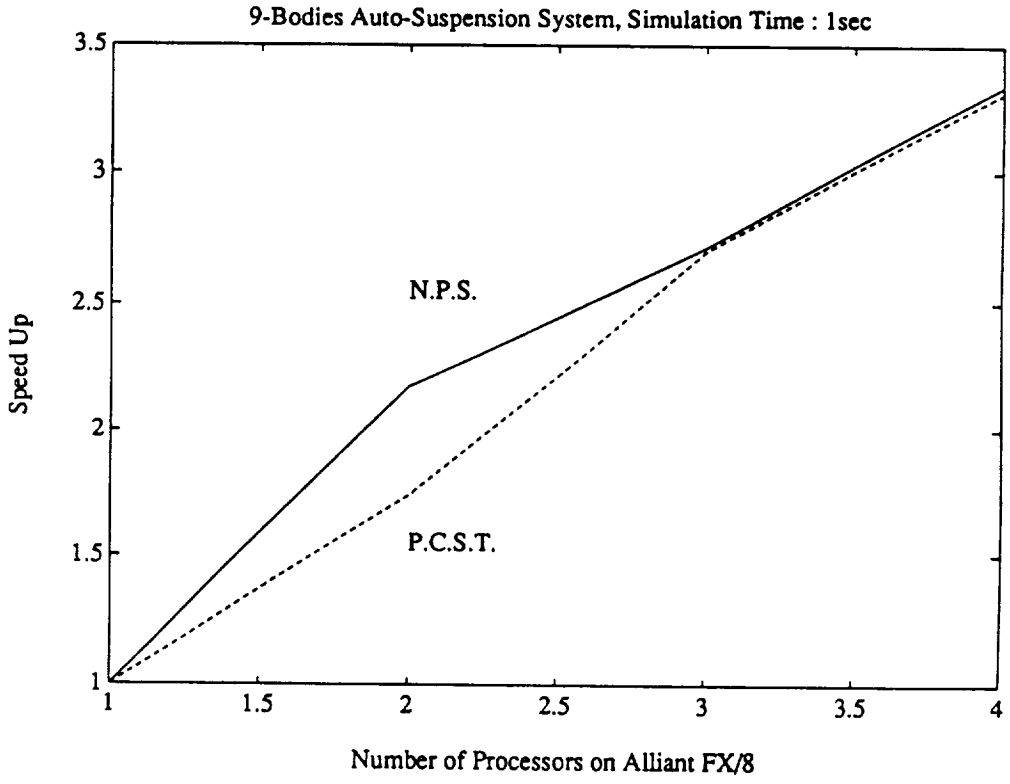


Fig. 17 Comparison of Speed Up by Both Techniques on Alliant FX/8

Stabilization of Computational Procedures for Constrained Dynamical Systems

K. C. Park and J. C. Chiou

Reprinted from

Journal of Guidance, Control, and Dynamics

Volume 11, Number 4, July-August 1988, Pages 365-370
AMERICAN INSTITUTE OF AERONAUTICS AND ASTRONAUTICS, INC.
370 L'ENFANT PROMENADE, SW • WASHINGTON, DC 20024

Stabilization of Computational Procedures for Constrained Dynamical Systems

K. C. Park* and J. C. Chiou†
University of Colorado, Boulder, Colorado

A new stabilization method of treating constraints in multibody dynamical systems is presented. By tailoring a penalty form of the constraint equations, the method achieves stabilization without artificial damping and yields a companion matrix differential equation for the constraint forces; hence, the constraint forces are obtained by integrating the companion differential equation for the constraint forces in time. A principal feature of the method is that the errors committed in each constraint condition decay with its corresponding characteristic time scale associated with its constraint force. Numerical experiments indicate that the method yields a marked improvement over existing techniques.

I. Introduction

THE dynamics of flexible multibody systems, such as the design of robotic manipulators, mechanical chains, and satellites, is becoming increasingly important in engineering. Computer simulation of such multibody dynamical (MBD) systems requires a concerted integration of several computational aspects. These include selection of a data structure for describing the system topology, computerized generation of the governing equations of motion, incorporation of constraint conditions, implementation of suitable solution algorithms, and easy interpretation of the simulation results.

Traditionally, the task of formulating the equations of motion has been of dominant concern to many dynamists. As a result, several MBD formulations have been proposed; these differ primarily in the manner in which they incorporate constraints and in their resulting system topologies.[1-13] Hence, reliability and cost of existing MBD simulation packages have been strongly affected by how well the equations of motion have been streamlined and how well the constraints are preserved during the numerical solution stage.

As dynamists face more complex problems, particularly in the field of large space structures, a new consensus is emerging: MBD simulation requires a data structure that can accommodate various system topologies. A primary motivation for espousing a maximum flexibility in the data structure is to allow, for each subsystem of a complex MBD system, the adoption of different modeling assumptions, different formulations of the equations of motion, and different solution techniques. Once this need is recognized, compatibility of subsystems as well as of various constraints becomes a focal computational issue. However, enforcing such subsystem and kinematical compatibilities leads to a formulation that involves a set of auxiliary constraints that must be satisfied at each integration step.

Because it is important in the simulation of MBD systems to treat the resulting constraints accurately and reliably, several computational procedures have been proposed. These include the technique for condensing dependent variables via singular-value decomposition by Walton and Steeves,[14] equilibrium correction strategies by Baumgarte,[15,16] penalty formulation by Orlandea et al.[17] and Lötstedt,[18] the coordinate partitioning technique by Wehage and Haug,[19] and the differential/algebraic approach by Gear[20] and Petzold.[21] In addition, recent reports by Huston and Kamman,[23] Fuehrer and Wallrapp,[24] Schwertassek and Roberson,[25] and Nikravesh[26] address various related techniques.

Among the procedures cited, it is generally agreed that Baumgarte's technique is the most reliable one for handling constraints. Thus, we believe that new methods for constraint stabilization should be compared with Baumgarte's technique. However, an examination of Baumgarte's technique has revealed that it has three important algorithmic and software difficulties.

First, according to Baumgarte's formulation that leads to his constraint stabilization, the error committed in all the constraint conditions during time integration steps can decay only with a uniform characteristic time constant. In other words, each of the constraint equations converges at the same rate regardless of its physical nature. This uniform convergence rate masks an important physical phenomenon: the characteristic time constants of each constraint equation are different, since Lagrangian multipliers associated with the constraint equations exhibit different physical response characteristics. Hence, Baumgarte's technique does not exploit the well-known observation that the principal errors in multi-degree-of-freedom systems behave the same way as do those associated with the individual physical components.

Second, Baumgarte's technique requires that the solution matrix, $B^T M^{-1} B$, can be invertible, where B is the gradient of the constraint equations and M is the mass matrix. It is noted that the solution matrix becomes singular (or ill-conditioned) if two or more constraints become numerically dependent (or almost dependent) upon one another. When that happens, the potential gain in accuracy realized by Baumgarte's stabilization is lost.

Third, Baumgarte's technique requires the solution of an augmented matrix equation that involves the constraint gradient matrix B. This means that whenever additional constraints are introduced or when some of the constraints are relaxed, the matrix profiles of the total-system equations will have to be varied. The task of dynamically varying matrix profiles of the total-system equations can significantly complicate software implementation.

The objective of the present paper is to report a new stabilization technique that is aimed at mitigating the three algorithmic and software difficulties of Baumgarte's technique. First, the new technique induces the errors in the constraint equations to decay according to their principal characteristic

Received July 6, 1987; revision received Sept. 16, 1987.

*Professor, Department of Aerospace Engineering Sciences and Center for Space Structures and Controls. Member AIAA.

†Graduate Research Assistant, Department of Aerospace Engineering Sciences and Center for Space Structures and Controls.

response time constants; the principal errors in the constraint equations diminish according to their corresponding physical response characteristics. Second, the new technique overcomes the nonconvergence difficulty when two or more constraints become *numerically* dependent. Third, the new technique yields a matrix differential equation for the constraint forces. Hence, the solution of the constraint forces can be carried out in a separate module from that for the primary solution variables (the position vector for the dynamical equations). To this end, the paper is organized as follows.

Section II presents a review of the Lagrangian λ-method[26] for formulating the equations of motion with constraints, including both configuration (*holonomic*) constraints and motion (*nonholonomic*) constraints. An examination of Baumgarte's stabilization for constraints is offered in Sec. III, delineating in detail the three noted algorithmic and software implementation difficulties of the Baumgarte stabilization technique.

Section IV presents a new stabilization technique based on a control synthesis approach. First, we introduce the well-known penalty technique so that the constraint forces are made proportional to violations of the constraint conditions. Second, by tailoring the governing equations of motion and by augmenting the constraint equations with the tailored form of the equations of motion, a stabilized differential form of constraint equation is derived. The resulting stabilized constraint equations are shown to be matrix differential equations with the constraint forces as the primary solution vector, yet possessing no artificial damping as is the case with Baumgarte's technique. Hence, one is left with a set of coupled differential equations of motion in which the generalized displacements and the constraint forces form a conjugate pair of unknowns. It should be mentioned that a similar approach has been successfully utilized for the solution of fluid-structure interaction equations[27] and of fluid-porous soil interaction equations[28] when the interaction equations are partitioned[29,30] and solved in a staggered manner. For this reason, the present method will be called *a staggered stabilization technique*.

Section V reports numerical experiments that illustrate the improved performance of the present staggered stabilization technique.[31] The paper ends with concluding remarks regarding computer implementation issues in production-level MBD simulation modules.

II. Equations of Motion with Constraints

The Lagrangian equations of motion for mechanical systems with constraints can be written as

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = Q_i + \sum_{k=1}^{m} \lambda_k B_{ki}, \qquad i = 1 \ldots n \tag{1}$$

$$\Phi_k(q, \dot{q}, \ddot{q}) = 0 \tag{2}$$

where L is the system Lagrangian, Φ_k are the constraint conditions imposed either on the subsystem boundaries or on the kinematical relations among the generalized coordinates, q_i are the generalized coordinate components, t is time, $(\dot{})$ denotes time differentiation, λ is the Lagrangian multiplier, Q_i is the generalized applied force, and B_{ki} is the ith gradient component of the kth constraint equation, Eq. (2).

In order to focus our subsequent discussions, we specialize Eq. (2) to the holonomic (configuration) case:

$$\Phi_k(q) = 0, \qquad B^h_{ki} = \frac{\partial \Phi_k}{\partial q_i}, \qquad k = 1 \ldots m \tag{3}$$

and to nonholonomic (motion) case:

$$\Phi_k(q, \dot{q}) = 0, \qquad B^{nh}_{ki} = \frac{\partial \Phi_k}{\partial \dot{q}_i}, \qquad k = 1 \ldots m \tag{4}$$

It should be noted that the constraint forces Q^c_i are obtained by

$$Q^c_i = \sum_{k=1}^{m} \lambda_k B_{ki}, \qquad i = 1 \ldots n \tag{5}$$

and *not* by λ_k alone.

Because the two constraints give rise to two different sets of equations of motion, we will treat their time discretization separately. It should be mentioned that a typical MBD system involves both cases; hence, the solution procedure should account for the two constraints concurrently.

Systems with Nonholonomic Constraints Only

When the system involves only nonholonomic constraints, the equations of motion become

$$\begin{bmatrix} M & B^T \\ B & 0 \end{bmatrix} \begin{Bmatrix} \ddot{q} \\ \lambda \end{Bmatrix} = \begin{Bmatrix} \bar{Q} \\ c \end{Bmatrix} \tag{6}$$

where M is the mass matrix, $\bar{Q}$ consists of the applied force Q, the centrifugal and Coriolis force, and the internal spring force, and c is given by

$$c = -\frac{\partial \Phi}{\partial t} \tag{7}$$

Systems with Holonomic Constraints Only

When the system involves only holonomic constraints, the equations of motion become

$$\begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} \ddot{q} \\ \lambda \end{Bmatrix} + \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix} \begin{Bmatrix} \dot{q} \\ \lambda \end{Bmatrix} = \begin{Bmatrix} \bar{Q} \\ c \end{Bmatrix} \tag{8}$$

III. Baumgarte's Stabilization Technique

In Baumgarte's technique, one replaces the second row of Eq. (6) for the case of nonholonomic constraints by

$$\dot{\Phi} + \gamma\Phi = 0 \tag{9}$$

Hence, the right-hand side of the second row of Eq. (6) is modified as

$$c = -\frac{\partial \Phi}{\partial t} - \gamma\Phi \tag{10}$$

Baumgarte sketched a solution scheme that uses the given parabolic stabilization technique as follows. First, the parabolically stabilized equation may be expanded as

$$B\ddot{q} + \frac{\partial \Phi}{\partial t} + \gamma\Phi = 0 \tag{11}$$

By substituting $\ddot{q}$ from the first row of Eq. (6), one obtains for λ in the form

$$(BM^{-1}B^T)\lambda = BM^{-1}\bar{Q} + \frac{\partial \Phi}{\partial t} + \gamma\Phi \tag{12}$$

Hence, λ in the preceding expression can be substituted into the governing equations of motion to yield

$$M\ddot{q} = \bar{Q} - B^T(BM^{-1}B^T)^{-1}\left\{BM^{-1}\bar{Q} + \frac{\partial \Phi}{\partial t} + \gamma\Phi\right\} \tag{13}$$

which can be integrated by an explicit integration formula.

For holonomic cases, he has recommended the following integro-differential form:[16]

$$\dot{\Phi} + 2\gamma\Phi + \gamma^2 \int_{t_0}^{t} \Phi \, dt = 0 \tag{14}$$

so that one obtains

$$c = -\frac{\partial \Phi}{\partial t} - 2\gamma\Phi - \gamma^2 \int_{t_0}^{t} \Phi \, dt \tag{15}$$

In the paper where Baumgarte presented this procedure, no solution scheme was suggested, except that he advocated the adoption of generalized momenta as the primary variables. In the present context of the generalized coordinates q, a plausible implementation of the stabilized integro-differential constraint equations may be realized as follows. First, one integrates the governing equation of motion, Eq. (8a), by an implicit integration formula

$$q^{n+1} = \delta \dot{q}^{n+1} + h_q^n \tag{16}$$

where δ is a formula-dependent stepsize and h_q^n is a historical vector. For example, for the trapezoidal rule, we have

$$\delta = (t_{n+1} - t_n)/2, \qquad h_q^n = q^n + \delta \dot{q}^n \tag{17}$$

Integrating the equations of motion with holonomic constraints once, by the preceding implicit formula, one obtains

$$\dot{q}^{n+1} = \delta M^{-1}\left(\bar{Q} - B^T \lambda^{n+1}\right) + h_q^n \tag{18}$$

We now substitute the preceding equation into the stabilized integro-differential constraint equation, Eq. (14), to yield

$$\delta B M^{-1} B^T \lambda^{n+1} = \delta B M^{-1} \bar{Q} + h_q^n + 2\gamma\Phi + \gamma^2 \int_{t_0}^{t} \Phi \, dt \tag{19}$$

After substituting the given expression for λ, one can integrate the resulting equation to obtain q^{n+1} by either an implicit or explicit integration formula. We now offer the following remarks.

Remark 1

Each of the constraints for both the holonomic and nonholonomic cases, $\{\Phi_k,\ k = 1 \ldots m\}$, possesses the same parabolic time constant γ, since its solution can be expressed as

$$\Phi_k = C_k e^{-\gamma t}, \qquad k = 1 \ldots m \tag{20}$$

Note that the errors committed in each of the constraints also decay with the same single time constant. However, regardless of their physical time constants, the errors in the constraint conditions by the stabilized constraint equations, Eqs. (9) and (14), are forced to decrease at the same rate. Hence, the technique does not take advantage of physically different time constants in order to minimize the errors being accumulated in the constraint equations.

Remark 2

Note that the generalized constraint forces λ in Eq. (12) exist only when the matrix $BM^{-1}B^T$ is not ill-conditioned. Even though the constraints are theoretically independent, such ill-conditioning can occur when two or more constraints become *numerically* nearly dependent, as B is in general state-dependent. If such situations develop, the accuracy of generalized constraint force λ can be considerably degraded, thus leading to a dramatic loss of solution accuracy for q.

Remark 3

From computer implementation considerations, the solution of MBD systems by Baumgarte's technique must be carried out in a tightly coupled program module. Therefore, any change in the number of constraints impacts the matrix structure of the solution procedures, requiring dynamically varying matrix profiles. This can considerably complicate the task of software implementation.

We will now present a new stabilization technique that mitigates the three algorithmic and software implementation difficulties in Baumgarte's stabilization technique pointed out in the preceding remarks.

IV. New Technique: Staggered Stabilization Procedure

In Baumgarte's stabilization technique, as discussed in the preceding section, the objective was to minimize the errors initiated in the constraint condition

$$\Phi = 0 \tag{21}$$

First, the difficulty associated with numerically dependent constraints alluded to in Remark 2 can be overcome by adopting the penalty procedure

$$\lambda = \frac{1}{\epsilon}\Phi, \qquad \epsilon \to 0 \tag{22}$$

as the basic constraint equations instead of Eqs. (3) and (4). It is noted that the penalty procedure as given by Eq. (22) tacitly assumes violations of the constraint condition in actual computations. If one substitutes Eq. (22) into the governing equations of motion, the result becomes

$$M\ddot{q} + \frac{1}{\epsilon} B^T \Phi = \bar{Q} \tag{23}$$

It can be shown that this penalty procedure mitigates nonconvergence difficulties in the constraint conditions. However, its major drawback is that once an error is committed in computing λ, there is no compensation scheme by which the drifting of the numerical solution can be corrected. It is this observation that has led to the development of a staggered stabilization procedure as described in the following paragraphs.

To illustrate the new procedure we will consider the case of nonholonomic constraints. Instead of substituting the penalty expression directly into the governing equations of motion, first we differentiate Eq. (22) once to obtain

$$\dot{\lambda} = \frac{1}{\epsilon}\left(B\ddot{q} + \frac{\partial \Phi}{\partial t}\right) \tag{24}$$

where we assume the penalty parameter ϵ to be constant.

Second, we obtain $\ddot{q}$ from Eq. (6a) in the form

$$\ddot{q} = M^{-1}\left(\bar{Q} - B^T \lambda\right) \tag{25}$$

and substitute it into Eq. (24) to yield

$$\epsilon\dot{\lambda} + BM^{-1}B^T\lambda = BM^{-1}\bar{Q} + \frac{\partial \Phi}{\partial t} \tag{26}$$

Notice that the homogeneous part of this stabilized equation in terms of the generalized constraint forces λ has the following companion eigenvalue problem:

$$\left(\gamma + BM^{-1}B^T/\epsilon\right) y = 0 \tag{27}$$

where $\{\gamma_k,\ k = 1 \ldots m\}$ are the eigenvalues of the homogeneous operator for the new stabilized constraint equations, Eqs. (26). Since γ_k also dictates how the errors in the constraint equations will diminish with time, the errors committed in the constraint conditions will decay with their corresponding different response time constants. This physically oriented stabilization property of the present technique is in contrast to that of Baumgarte's technique wherein all the error components diminish according to a single time constant.

Third, the new technique enables us to solve for λ from the stabilized differential equation, Eq. (26). Specifically, we now have a set of coupled equations, one for the generalized coordinates q and the other for the generalized constraint forces λ, which are recalled here from Eqs. (6a) and (26) for the case of nonholonomic constraints:

$$\begin{bmatrix} M & 0 \\ 0 & \epsilon \end{bmatrix} \begin{Bmatrix} \ddot{q} \\ \dot{\lambda} \end{Bmatrix} + \begin{bmatrix} 0 & B^T \\ 0 & BM^{-1}B^T \end{bmatrix} \begin{Bmatrix} \dot{q} \\ \lambda \end{Bmatrix} = \begin{Bmatrix} \bar{Q} \\ BM^{-1}\bar{Q} + \dfrac{\partial \Phi}{\partial t} \end{Bmatrix} \tag{28}$$

Note that these coupled equations directly provide the desired differential equations for a conjugate pair of $[q\ \lambda]$.

Remark 4

For holonomic constraints, one has several stabilization possibilities. The one we have chosen is to integrate the governing equations of motion once to obtain

$$\dot{q}'' = \delta M^{-1}\left(\bar{Q}'' - B^T\lambda''\right) + h_q'' \tag{29}$$

which is substituted into

$$\dot{\lambda} = \frac{1}{\epsilon}\left(B\dot{q} + \frac{\partial \Phi}{\partial t}\right) \tag{30}$$

to yield

$$\epsilon\dot{\lambda}'' + \delta BM^{-1}B^T\lambda'' = B\left(\delta M^{-1}\bar{Q}'' + h_q''\right) + \frac{\partial \Phi}{\partial t} \tag{31}$$

Remark 5

It is observed that even if $BM^{-1}B^T$ is almost singular, the new stabilization technique as derived in Eqs. (26) and (31) would not cause numerical difficulty in computing λ since the solution iteration matrix becomes $(\epsilon + \delta BM^{-1}B^T)$ for nonholonomic cases and $(\epsilon + \delta^2 BM^{-1}B^T)$ for holonomic cases.

Remark 6

The present staggered stabilization technique and Baumgarte's technique can be presented in control-synthesis block diagrams, as shown in Figs. 1a and 1b. For nonholonomic constraints, the present technique can be viewed as a combination of gain plus rate feedback stabilization, whereas Baumgarte's technique is seen as a simple gain feedback stabilization. For holonomic constraints, a similar distinction can be observed. The resulting feature of a rate feedback manifested in the present staggered stabilization technique constitutes an important attribute as it copes with the dynamical nature of the problem.

V. Numerical Evaluation

The first problem is a one-bar rigid pendulum problem studied in Ref. 15. The equations of motion consist of both horizontal and vertical trajectories of the pendulum's tip plus one constraint equation for the circular motion of the tip; thus, there are two position variables and one holonomic constraint condition. First, we fix the integration stepsize and carry out the numerical solution by the trapezoidal rule without iteration for both stabilization techniques. Figure 2 shows the errors in the constraint condition for the two techniques. The results show that the present technique yields accuracy about two orders of magnitude higher than that yielded by Baumgarte's technique. In order to gain further insight, the accuracy level in the constraint condition is fixed to be the same (10^{-6}) at each time step and the solution matrix is iterated to satisfy the accuracy requirement. Figure 3 illustrates the number of iterations needed at each step vs time. Note that the average iteration number for the present technique is about four, whereas with Baumgarte's technique it is about six.

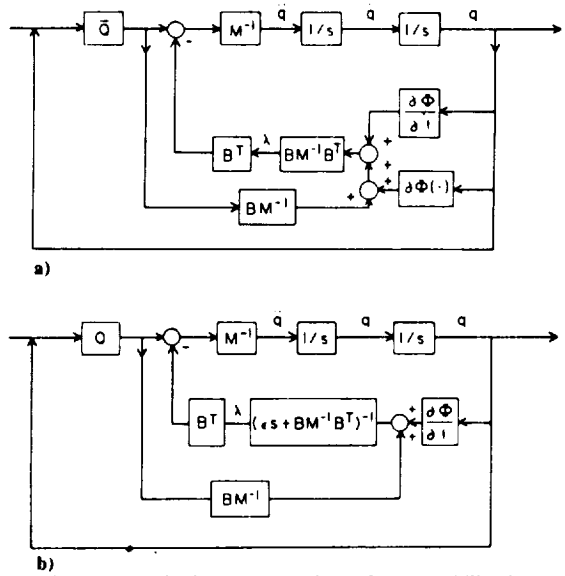


Fig. 1 Control synthesis representation of two stabilization techniques: a) Baumgarte's technique, and b) stabilized technique.

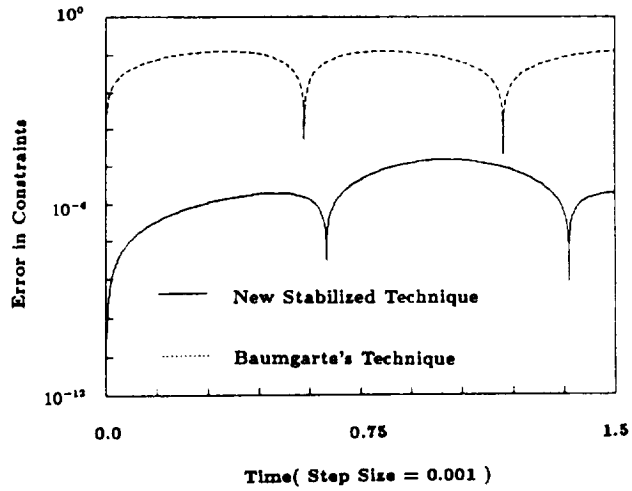


Fig. 2 Errors in constraint with no iteration, performance of two stabilized techniques (single pendulum problem).

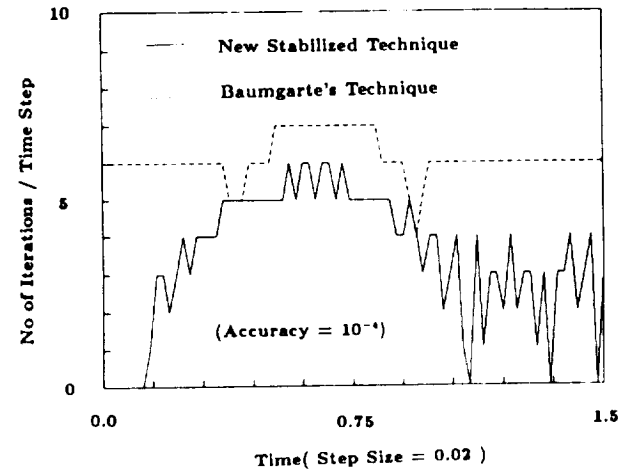


Fig. 3 Number of iterations required for given error tolerance, performance of two stabilized techniques (single pendulum problem).

The second example is a classical crank mechanism whose governing equations of motion are characterized by the following matrices and constraints [see Eqs. (3-8) for their definitions]:

$$M = \begin{bmatrix} J_1 & & & \\ & J_2 & & \\ & & m & \\ & & & m \end{bmatrix} \tag{32}$$

$$\Phi = \begin{Bmatrix} r\cos\theta - (x - l_1\cos\phi) \\ r\sin\theta - (y - l_1\sin\phi) \\ (l - l_1)\sin\phi + y \end{Bmatrix} = 0 \tag{33}$$

$$B^T = \begin{bmatrix} -r\sin\theta & r\cos\theta & 0 \\ -l_1\sin\theta & l_1\cos\phi & (l - l_1)\cos\phi \\ -1 & 0 & 0 \\ 0 & -1 & 1 \end{bmatrix} \tag{34}$$

and

$$q = \begin{bmatrix} \theta & \phi & x & y \end{bmatrix}^T, \quad \lambda = \begin{bmatrix} \lambda_1 & \lambda_2 & \lambda_3 \end{bmatrix}^T$$
$$Q = \begin{Bmatrix} 0 & 0 & 0 & -mg \end{Bmatrix}^T \tag{35}$$

Figure 4 shows the problem definition along with the numerical performance of the two procedures, Baumgarte's technique and the staggered stabilization technique. The performance of the Baumgarte technique and that of the staggered stabilization technique for this problem are also presented in Fig. 4. In carrying out the computations, the trapezoidal rule has been used to time-discretize the equations of motion [Eqs. (2)], the constraints [Eqs. (3)], and their stabilized forms [Eqs. (19) and (28)]. A sufficiently small step increment was used, corresponding to 82 increments for one cycle of the mechanism, with the time increment $h = 0.01$ for the period $T = 0.82$. In order to measure the performance of the two techniques directly, in terms of violation of the constraint conditions vs time during one complete cycle, no iteration was performed at each integration step. In each technique, the three constraint conditions exhibited the same order of accuracy level. Hence, we illustrate only one constraint violation history, i.e., the pin joint constraint between the crank and the connecting rod. Note that the error in the constraint condition for Baumgarte's technique remains about two digits above that with the staggered stabilization technique. In addition, we have experimented with several values of α and β that are required in Baumgarte's technique, and the best parameter choice was found to be $\alpha = \beta = 70$. For the staggered stabilization technique, the penalty parameter chosen was $\epsilon = 10^{-6}$, which yielded an accuracy level about 10^{-5} for the technique.

The third problem tested is a simplified version of the seven-link manipulator deployment problem.[13] The three links are initially folded and, for modeling simplicity, between the two joints is a coil spring that resists a constant deploying force at the tip of the third link. Also, the left-hand end of the first link is fixed through the same coil spring to the wall. These three coil springs are to be *locked up* once the links are deployed straight. The deployment sequence of the manipulator is illustrated in Fig. 5. The time-discretized difference equations both for Baumgarte's technique and the staggered stabilization technique have been solved at each time increment by a Newton-type iterative procedure to meet a specified accuracy level. Hence, the performance of the two techniques can be assessed by the average number of iterations taken per time increment. This is presented in Fig. 6 for the accuracy of 10^{-4}. Notice that the staggered stabilization technique requires on the average about 4.5 iterations per step, whereas Baumgarte's technique requires about 22 iterations per step.

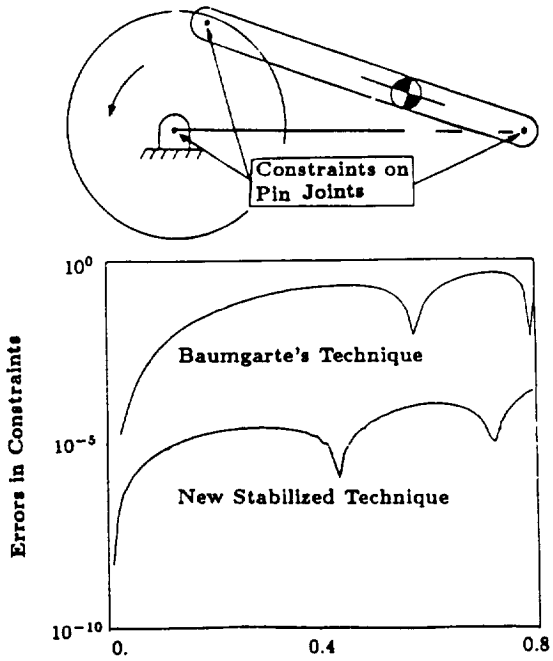


Fig. 4 Errors in pin-joint constraint with no iteration, performance of two techniques.

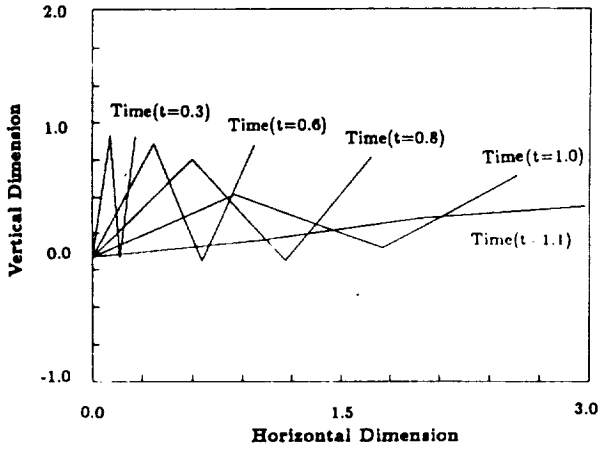


Fig. 5 Deployment of three-link remote manipulator.

Note that Baumgarte's technique fails to converge for time, $t \approx 1.1$, as manifested in Fig. 6 because the rows in B become numerically dependent upon one another when the links are in a straight configuration. This corroborates the theoretical prediction of nonconvergence whenever the solution matrix $BM^{-1}B^T$ for Baumgarte's technique [see Eq. (12)] becomes singular. On the other hand, the staggered stabilization technique still converges within 30 iterations because it overcomes this singularity difficulty, since $\dot{\lambda}$ still exists, as can be seen from Eqs. (26) and (31). Although not reported here, the same relative performance has been observed for different accuracy levels, i.e., for the accuracy of 10^{-5} and 10^{-6}.

From the sample test problems, we conclude that the staggered stabilization technique yields both improved accuracy over and greater computational robustness than the Baumgarte technique. In addition, the staggered stabilization technique offers software modularity in that the solution of the constraint force λ can be carried out separately from that of the generalized displacement q. The only data each solution module needs to exchange with the other is a set of vectors, plus a common module to generate the gradient matrix of the

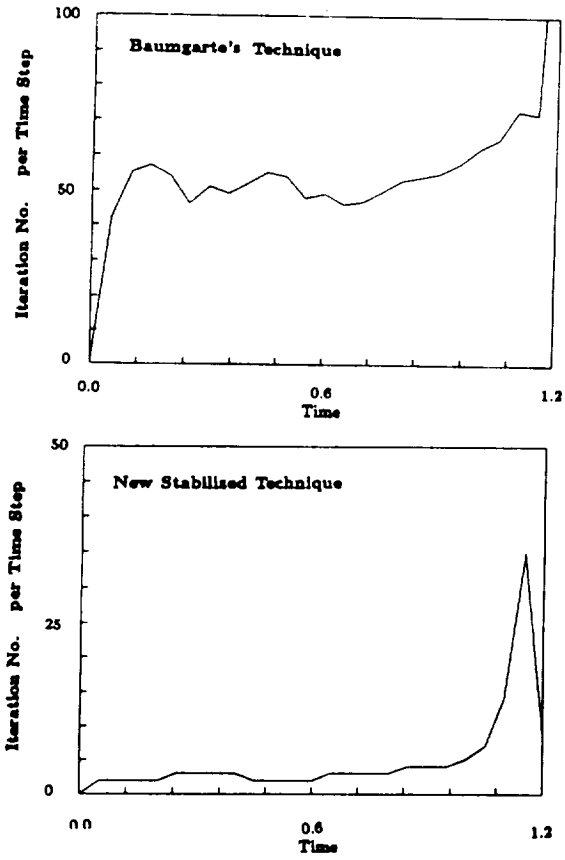


Fig. 6 Performance of two stabilization techniques for three-link manipulator (solution accuracy = 10^{-6}).

constraints, B. However, one should be cautioned not to extrapolate blindly to complex problems the results of the present simple examples. Further judicious experiments are needed in applying the present staggered stabilization technique to complex production-level problems before it can be adopted for general applications in multibody dynamic simulations.

Acknowledgments

The work reported herein was supported by NASA Langley Research Center under Contract NAS1-17660. The authors wish to thank Drs. Jerry Housner and Jeff Stroud for their keen interest and encouragement during the course of the present work.

References

[1]Hooker, W. and Margulies, G., "The Dynamical Attitude Equations for an N-body Satellite," *Journal of Astronautical Science*, Vol. 12, 1965, pp. 123–128.

[2]Roberson, R. and Wittenburg, J., "A Dynamical Formalism for an Arbitrary Number of Interconnected Rigid Bodies with Reference to the Problem of Satellite Attitude Control," *Proceedings of the Third International Congress of Automatic Control*, Butterworth, London, 1965.

[3]Likins, P., "Analytical Dynamics and Nonrigid Spacecraft Simulation," TR 32-1593, Jet Propulsion Laboratory, Pasadena, CA, 1974.

[4]Ho, J., "The Direct Path Method for Deriving the Dynamic Equations of Motion of a Multibody Flexible Spacecraft with Topological Tree Configuration," AIAA Paper 74-786, 1974.

[5]Roberson, R., "A Form of the Translational Dynamical Equations for Relative Motion in Systems of Many Non-Rigid Bodies," *Acta Mechanica*, Vol. 14, 1972, pp. 297–308.

[6]Boland, P., Samin, J., and Willems, P., "Stability Analysis of Interconnected Deformable Bodies in a Topological Tree," *AIAA Journal*, Vol. 12, Aug. 1974, pp. 1025–1030.

[7]De Veubeke, B.F., "The Dynamics of Flexible Bodies," *International Journal of Engineering Science*, Vol. 14, 1976, pp. 895–913.

[8]Keat, J.E., *Dynamical Equations of Body Systems with Application to Space Structure Deployment*, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1983.

[9]Kane, T. and Levinson, D., "Formulation of Equations of Motion for Complex Spacecraft," *Journal of Guidance and Control*, Vol. 3, March–April 1980, pp. 99–112.

[10]Bodley, C.S., Devers, A.D., Park, A.C., and Frish, H.P., "A Digital Computer Program for the Dynamic Interaction Simulation of Control and Structures (DISCOS)," NASA TP-1219, May 1978.

[11]*The ADAMS User's Guide*, Mechanical Dynamics, Inc., Ann Arbor, MI, 1979.

[12]Keat, J.E., *Dynamical Equations of Rigid Body Systems with Application to Space Structure Deployment*, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1983.

[13]Housner, J.M., "Convected Transient Analysis for Large Space Structure Maneuver and Deployment," *Proceedings of the 25th Structures, Structural Dynamics and Materials Conference*, Part 2, AIAA New York, 1984, pp. 616–619.

[14]Walton, W.C. and Steeves, E.C., "A New Matrix Theorem and Its Application for Establishing Independent Coordinates for Complex Dynamical Systems with Constraints," NASA TR-R326, 1969.

[15]Baumgarte, J.W., "Stabilization of Constraints and Integrals of Motion in Dynamical Systems," *Computational Methods in Applied Mechanics and Engineering*, Vol. 1, 1972, pp. 1–16.

[16]Baumgarte, J.W., "A New Method of Stabilization for Holonomic Constraints," *Journal of Applied Mechanics*, Vol. 50, 1983, pp. 869–870.

[17]Orlandea, N., Chase, M.A., and Calahan, D.A., "A Sparsity-Oriented Approach to the Dynamic Analysis and Design of Mechanical Systems—Parts I and II," *Transactions of the ASME, Journal of Engineering for Industry, Series B*, Vol. 99, 1977, pp. 773–784.

[18]Lötstedt, P., "On a Penalty Function Method for the Simulation of Mechanical Systems Subject to Constraints," TRITA-NA-7919, Royal Institute of Technology, Stockholm, Sweden, 1979.

[19]Wehage, R.A. and Haug, E.J., "Generalized Coordinate Partitioning for Dimension Reduction in Analysis of Constrained Dynamic Systems," *ASME Journal of Mechanical Design*, Vol. 104, 1982, pp. 247–255.

[20]Gear, C.W., "Simultaneous Numerical Solution of Differential/Algebraic Equations," *IEEE Transactions on Circuit Theory*, CT-18, 1971, pp. 89–95.

[21]Petzold, L., "Differential/Algebraic Equations are not ODEs," *SIAM Journal of Scientific Statistical Computation*, Vol. 3, 1982, pp. 367–384.

[22]*Penalty-Finite Element Methods in Mechanics*, edited by J.N. Reddy, American Society of Mechanical Engineers, AMD Vol. 51, New York, 1982.

[23]Huston R.L. and Kamman, J.W., "A Discussion on Constraint Equations in Multibody Dynamics," *Mechanical Research Communication*, Vol. 9, 1982, pp. 251–256.

[24]Fuehrer, C. and Wallrapp, O., "A Computer-Oriented Method for Reducing Linearized Multibody System Equations by Incorporating Constraints," *Computational Methods in Applied Mechanics and Engineering*, Vol. 46, 1984, pp.169–175.

[25]Schwertassek, R. and Roberson, R.E., "A State-Space Dynamical Representation for Multibody Mechanical Systems, Part II," *Acta Mechanica*, Vol. 51, 1984, pp. 15–29.

[26]Nikravesh, P.E., "Some Methods for Dynamic Analysis of Constrained Mechanical Systems: A Survey," *Computer Aided Analysis and Optimization of Mechanical System Dynamics*, edited by E.J. Haug, NATO ASI Series, F9, Springer-Verlag, Berlin, 1984, pp. 351–367.

[27]Lanczos, L., *The Variational Principles of Mechanics*, 4th ed., University of Toronto Press, Toronto, 1970, pp. 141–147.

[28]Park, K.C., Felippa, C.A., and DeRuntz, J.A., "Stabilization of Staggered Solution Procedures for Fluid-Structure Interaction Analysis," *Computational Methods for Fluid-Structure Interaction Problems*, edited by T. Belytschko and T.L. Geers, ASME, AMD Vol. 26, New York, 1977, pp. 95–124.

[29]Park, K.C., "Partitioned Transient Analysis Procedures for Coupled-Field Problems: Stability Analysis," *Journal of Applied Mechanics*, Vol. 47, 1980, pp. 370–376.

[30]Park, K.C., "Stabilization of Partitioned Solution Procedures for Pore Fluid-Soil Interaction Analysis," *International Journal of Numerical Methods in Engineering*, Vol. 19, 1983, pp. 1669–1673.

[31]Park, K.C., "Stabilization of Computational Procedures for Constrained Dynamical Systems: Formulation," AIAA Paper 86-0926, May 1986.

Explicit-Implicit Staggered Procedure for Multibody Dynamics Analysis

K. C. Park, J. C. Chiou, J. D. Downer

Reprinted from

Journal of Guidance, Control, and Dynamics

Volume 13, Number 3, May-June 1990, Pages 562-570
AMERICAN INSTITUTE OF AERONAUTICS AND ASTRONAUTICS, INC.
370 L'ENFANT PROMENADE, SW • WASHINGTON, DC 20024

Explicit-Implicit Staggered Procedure for Multibody Dynamics Analysis

K. C. Park,* J. C. Chiou,† and J. D. Downer†
University of Colorado, Boulder, Colorado 80309

An explicit-implicit staggered time-integration procedure is presented for the solution of multibody dynamical equations involving large rotations and constraints. The algorithm adopts a two-stage modification of the central difference algorithm for integrating the translational coordinates and the angular velocity vector, and the midpoint implicit algorithm to solve the kinematical relation in terms of the Euler parameters for updating the angular orientations. The Lagrange multipliers to enforce the system constraints are obtained by implicitly integrating a parabolically regularized differential equation for the multipliers. The performance of the present procedure has been evaluated by applying the procedure to solve several sample problems. The results indicate that the procedure is robust in dealing with a variety of constraints and spatial kinematic motions, hence it is recommended for applications to general multibody dynamics analyses.

I. Introduction

COMPUTER simulation of multibody dynamical (MBD) systems has enjoyed substantial progress during the past several years. As a result, it is now almost routine to perform realistic modeling and assessment of some practical problems such as mechanical linkages and manipulations of robotic arms.[7] Recently, a new need for the large-scale, real-time simulation of flexible MBD systems is emerging primarily in support of deployment and construction of large space structures in orbit. The development of an MBD simulation software system for space applications must meet several needs, which include a versatile data structure for implementation of candidate MBD topologies, an automatic derivation of the equations of motion, a streamlined incorporation of the system constraints, a robust and efficient direct-time integration package, a modular interface with active-control systems, and timely visualization of the simulation results. Of these, the present paper focuses on a robust and efficient time-integration package with parallel/concurrent computers as its primary computational environment.

In general, there have been two types of direct-time integration algorithms for the transient response analysis of dynamical systems: explicit and implicit algorithms. Currently, implicit algorithms appear to be favored by many MBD specialists when both the generalized coordinates and the Lagrange multipliers are treated as the unknowns. In this case, the corresponding formulations incorporate the system constraints by the penalty augmentation through the Lagrange multipliers. It is well known that the resulting Newton-like solution matrix is stiff. This has led to implicit time discretization of the constraint-augmented equations and simultaneous solution of both the generalized coordinates and the Lagrange multipliers.[6,13,15,22,25]

On the other hand, if the constraints are eliminated so as to reduce the number of unknowns, it is possible for one to employ either implicit or explicit algorithms. For this situation, if the system topology is an open tree, one may invoke either a geometric or an algebraic procedure to streamline the resulting equations of motion. Geometric procedures rely on the use of the incidence matrix[26] and the body-array matrix.[11] Some of the proposed algebraic procedures include singular decomposition,[24] the use of generalized speed,[12] the coordinate partitioning technique,[25] and the so-called order-N procedure.[14]

In developing the present MBD solution procedure, we have been guided by the following considerations, which have led to the selection of an explicit algorithm. First, the algorithm must be robust; experience suggests that explicit algorithms remain robust provided computations are stable. Second, the algorithm should be easily interfaced with a constraint processor as well as an active control synthesizer; the task of interfacing a software module with other software modules becomes easier if its data structure is simple, thus favoring an explicit algorithm. To this end, as the central difference integration algorithm has been most widely used for the explicit transient analysis of structural dynamics problems, we have decided to adopt the central difference algorithm as our basic integration algorithm. The rest of the paper is organized as follows.

In Sec. II, we introduce basic equations of motion for MBD systems. For computational efficiency, the translational coordinates are expressed in the fixed-inertial frame, whereas the rotational coordinates are expressed in the moving body-fixed frame in terms of the Euler parameters. Section III introduces the partitioning of the governing equations of motion into two groups: translational and rotational. Such partitioning paves the way for the efficient treatment of the rotational motions via the singularity-free Euler parameters, which treatment is a major feature of the present paper.

Section IV introduces the standard form of the central difference method for updating both the translational and the angular velocities. Once the angular velocities are obtained, the angular orientations are updated via the midpoint implicit formula employing the Euler parameters; update of the translational coordinates is achieved by the central difference method. It is shown that the standard form of the central-difference method is not applicable to the MBD equations, due to the unavailability of the generalized velocity vector at the time step at which the acceleration vector is evaluated. To overcome this difficulty, a staggered form of the central-difference method is developed.

Received July 19, 1988, revision received Nov. 17, 1988.

*Professor, Department of Aerospace Engineering Sciences and Center for Space Structures and Controls. Member of AIAA.

†Graduate Research Assistant, Department of Aerospace Engineering Sciences and Center for Space Structures and Controls.

To complete the description of the solution procedure for constrained MBD systems in Sec. V, the staggered-stabilized technique for the solution of the constraint forces as independent variables is summarized from Park and Chiou.[16,17] When the two algorithms—namely, the two-stage explicit algorithm for the generalized coordinates and the implicit, staggered procedure for the constraint Lagrange multipliers—are brought together in a staggered manner, they form an explicit-implicit staggered procedure.

Numerical evaluations of the present algorithm are reported in Sec. VI. Finally, Sec. VIII discusses several computational aspects of the present procedure and summarizes the main contributions of the present paper.

II. Equations of Motion for Multibody Systems

The discrete equations of motion for flexible multibody systems can be expressed as[4]

$$M\ddot{u} + D(\dot{u}) + S(u) + B_N^T\lambda_N + B_H^T\lambda_H = f(t) \tag{1}$$

$$\Phi_N(\dot{u},u,t) = 0, \qquad \Phi_H(u,t) = 0 \tag{2}$$

where M is the mass matrix, $D(\cdot)$ the generalized velocity-dependent force operator, $S(\cdot)$ the internal force operator due to member flexibility, B_N and B_H the gradients of the nonholonomic and holonomic constraints [Eq. (2)], λ_N and λ_H are the corresponding constraint forces, $f(t)$ is the applied force, u is the generalized displacement vector, $(\dot{})$ denotes time differentiation, and $()^T$ designates the matrix transposition.

The numerical solution of the constrained dynamical system governed by Eqs. (1) and (2) consists of two tasks: the satisfaction of the constraint conditions [Eq. (2)] to obtain λ and the computation of the generalized coordinates u from Eq. (1). A staggered, stabilized computational procedure to obtain λ_N and λ_H by satisfying Eq. (2) was presented in Park and Chiou[16,17] and is summarized in Sec. IV. The major thrust of the present paper is therefore devoted to the computation of the generalized coordinates u.

III. Partitioning of the Multibody Dynamical Equations

A basic difficulty in direct integration of Eq. (1) is that ω is not directly integrable, except for some special kinematic configurations, to yield angular orientations. This motivates us to partition $\dot{u}$ into the translational velocity vector $\dot{d}$, which is directly integrable, and the angular velocity vector ω, which is not, and to treat them by a partitioned solution procedure,[5,18-20] viz

$$\ddot{u} = \begin{Bmatrix} \ddot{d} \\ \dot{\omega} \end{Bmatrix}, \qquad \dot{u} = \begin{Bmatrix} \dot{d} \\ \omega \end{Bmatrix} \tag{3}$$

The equations of motion [Eq. (1)] can be rearranged according to the preceding partitioning:

$$\begin{bmatrix} M_d & 0 \\ 0 & M_\omega \end{bmatrix} \begin{Bmatrix} \ddot{d} \\ \dot{\omega} \end{Bmatrix} + \begin{Bmatrix} Q_d \\ Q_\omega \end{Bmatrix} = \begin{Bmatrix} f_d \\ f_\omega \end{Bmatrix} \tag{4}$$

where

$$\begin{Bmatrix} Q_d \\ Q_\omega \end{Bmatrix} = \begin{Bmatrix} Q_d(\dot{d},d,q,\lambda) \\ Q_\omega(\omega,d,q,\lambda) \end{Bmatrix} = \begin{Bmatrix} D_d(\dot{d}) + S_d(d,q) - B_d^T\lambda \\ D_\omega(\omega) + S_\omega(d,q) - B_\omega^T\lambda \end{Bmatrix} \tag{5}$$

in which q is the angular orientation parameter vector, and B_d and B_ω are the partitions of the combined gradient matrices of the constraint conditions (2) that are symbolically expressed as

$$B = B_N + B_H, \qquad \lambda = \lambda_N + \lambda_H \tag{6}$$

To effect the integration of the rotational degrees of freedom, we partition $\dot{\omega}$ further into

$$\dot{\omega} = \lfloor \dot{\omega}^1, \dot{\omega}^2, \ldots, \dot{\omega}^P \rfloor^T \tag{7}$$

where $\dot{\omega}^{(j)}$ is a (3×1) angular acceleration vector for the jth body,

$$\dot{\omega}^{(j)} = \lfloor \omega_1^{(j)}, \omega_2^{(j)}, \omega_3^{(j)} \rfloor \tag{8}$$

IV. Staggered Explicit Method for Multibody Dynamical Equations

One of the most popular explicit time integration formulas for the solution of the second-order dynamical equations is the central difference method, which can be implemented as

$$\dot{u}^{n+1/2} = \dot{u}^{n-1/2} + h\ddot{u}^n \tag{9a}$$

$$u^{n+1} = u^n + h\dot{u}^{n+1/2} \tag{9b}$$

where the superscript n designates the discrete time station $t = nh$ and h is the step increment.

It should be noted that the conventional form of the central difference method

$$u^{n+1} = 2u^n - u^{n-1} + h^2\ddot{u}^n \tag{10a}$$

$$\dot{u}^{n+1} = \dot{u}^n + (h/2)(\ddot{u}^{n+1} + \ddot{u}^n) \tag{10b}$$

is not applicable to the MBD equations since ω cannot in general be directly integrated to yield suitable angular orientations, let alone unfavorable accuracy problems associated with Eq. (10a) as succinctly discussed by Henrici.[8]

A. Integration of the Translational Coordinates

Assuming that $\{\dot{d}^{n-1/2}, \dot{d}^n, d^n, q^n, \lambda^n\}$ are given at the time steps, $t = (n - 1/2)h$ and nh, one can proceed to obtain from the partitioned equations of motion [Eq. (4)], the translational velocity and coordinates as

$$\dot{d}^{n+1/2} = \dot{d}^{n-1/2} + hM_d^{-1}[f_d^n - Q_d(\dot{d}^n, d^n, q^n, \lambda^n)] \tag{11a}$$

$$d^{n+1} = d^n + h\dot{d}^{n+1/2} \tag{11b}$$

Note that, due to the intrinsic time-stepping nature of Eqs. (9), $\dot{d}^n$ that is needed in computing Q_d^n is not available. This difficulty can be overcome if Q_d has the form

$$Q_d = D_d\dot{d} + S_d(d,q) - Bd(d,q) \tag{12}$$

where D_d is a constant diagonal matrix. For this special case, one can employ the averaging operator

$$D_d\dot{d}^n = D_d(1/2)(\dot{d}^{n+1/2} + \dot{d}^{n-1/2}) \tag{13}$$

so that Eq. (11a) is modified to

$$(M_d + 1/2hD_d)\dot{d}^{n+1/2} = (M_d - (h/2)D_d)\dot{d}^{n-1/2} + h(f_d^n - S_d(d^n,q^n) - B_d^n\lambda^n) \tag{14}$$

If, on the other hand, D_d is not diagonal or B_d contains $\dot{d}$, the modification offered in Eq. (14) loses the advantage of the central-difference method in that one must either factor the matrix $[M_d + 1/2hD_d]$ or iterate on B_d. This difficulty is more pronounced for updating the angular velocity vector as discussed next.

B. Integration of the Angular Orientations

One can update the angular velocity vector by Eq. (9a) using $\dot{\omega}$ from Eq. (4b)

$$\omega^{n+1/2} = \omega^{n-1/2} + hM_\omega^{-1}[f_\omega^n - Q_\omega(\omega^n, q^n, d^n, \lambda^n)] \tag{15}$$

A key feature of the present algorithm is the use of the following kinematic relation (e.g., Wittenburg[26]) to update the angular orientations:

$$\dot{q} = \frac{1}{2}\begin{bmatrix} 0 & -\omega^T \\ \omega & -\tilde{\omega} \end{bmatrix}, \qquad q = \frac{1}{2}A(\omega)q$$

$$q = \lfloor q_0 \quad q_1 \quad q_2 \quad q_3 \rfloor^T \tag{16}$$

that is subject to

$$q^T \cdot q = 1 \tag{17}$$

where

$$\tilde{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}, \quad \omega = \lfloor \omega_1 \quad \omega_2 \quad \omega_3 \rfloor^T \tag{18}$$

Of several procedures tested, we have found the following midpoint implicit rule is the most robust and accurate:

$$q^{n+1/2} = q^n + (h/2q)^{n+1/2}$$

$$= q^n + (h/2)A(\omega^{n+1/2}) \cdot q^{n+1/2} \tag{19a}$$

$$q_{n+1} = 2q^{n+1/2} - q^n, \qquad (q^{n+1})^T \cdot q^{n+1} = 1 \tag{19b}$$

where $q^{n+1/2}$ is obtained by

$$q^{n+1/2} = 1/\Delta[I + (h/4)A(\omega^{n+1/2})]q^n,$$

$$\Delta = 1 + (h^2/4)(\omega_1^2 + \omega_2^2 + \omega_3^2) \tag{20}$$

Finally, once q^{n+1} is computed from Eq. (19), one can update the angular orientation matrix R:

$$b = Re,$$

$$R = \begin{bmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & 2(q_0^2 + q_3^2) - 1 \end{bmatrix} \tag{21}$$

which relates the body-fixed basis vector, $b = \lfloor b_1 \; b_2 \; b_3 \rfloor^T$, to the integral-basis vector, $e = \lfloor e_1 \; e_2 \; e_3 \rfloor^T$.

It should be remarked that the update of the angular orientation parameters through the kinematical relation [Eq. (16)] is in contrast to the conventional algorithm in which one substitutes $\dot{\omega}$ in Eq. (4b) in terms of $\ddot{q}$ and $\dot{q}$ by

$$\dot{\omega} = T(q)\ddot{q} + T(\dot{q})\dot{q}$$

$$T = 2\begin{bmatrix} -q_1 & q_0 & q_3 & -q_2 \\ -q_2 & -q_3 & q_0 & q_1 \\ -q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \tag{22}$$

and integrates the resulting equations of motion to update q.

However, computations of $\omega^{n+1/2}$ by Eq. (15) assume that ω^n is available for every integration step. Note the $D_\omega(\omega)$ in Eq. (5) takes for each body the form of

$$D_\omega(\omega) = \tilde{\omega}J\omega \tag{23}$$

where J is the moment of inertia matrix. This term often dominates the momentum exchange in multibody systems and presents numerical difficulties if ω^n in D_ω is approximated by $\omega^{n-1/2}$, leading to inaccurate solutions or numerical instabilities.

A linearized computational stability analysis for the algorithm based on Eqs. (15) and (19), although we do not report it here, has been performed when $D_\omega(\omega^n)$ is approximated by $D_\omega(\omega^{n-1/2})$. The analysis result, as corroborated in Sec. VI, shows that such a naive approximation leads to unacceptable accuracy loss on outright instability. This has motivated us to implement both Eqs. (11) and (15) in a two-stage time-stepping procedure as detailed next.

C. Staggered Integration of the Translational and Rotational Coordinates

To alleviate the computational and stability issues encountered in the single-stage implementation of the central-difference method for MBD simulations, the basic algorithm presented in the preceding section needs to be modified as follows. Specifically, at an arbitrary integration step from $t = nh$ to $t = (n+1)h$, it is necessary for accuracy and stability that $\dot{d}^n$ and ω^n are available for Eqs. (11) and (15), respectively. Within the algorithmic context of the central difference method, this can be accomplished if we stagger the integration as follows.

First, instead of marching from the $(n+1)$ to the $(n+2)$ step at the completion of the $(n+1)$ step, we go back one-half step and march a full step from the $(n+1/2)$ to $(n+3/2)$ step:

$$\dot{d}^{n+1} = \dot{d}^n + h\ddot{d}(\dot{d}^{n+1/2}, d^{n+1/2}, q^{n+1/2}, \lambda^{n+1/2}) \tag{24a}$$

$$d^{n+3/2} = d^{n+1/2} + h\dot{d}^{n+1} \tag{24b}$$

for the update of the translational coordinates and

$$\omega^{n+1} = \omega^n + h\dot{\omega}(\omega^{n+1/2}, q^{n+1/2}, d^{n+1/2}, \lambda^{n+1/2}) \tag{25a}$$

$$q^{n+1} = (1/\Delta)[I + (h/4)A(\omega^{n+1})] \cdot q^{n+1/2} \tag{25b}$$

$$q^{n+3/2} = 2q^{n+1} - q^{n+1/2}, \qquad (q^{n+3/2})^T \cdot q^{n+3/2} = 1 \tag{25c}$$

for the rotational coordinates.

For the next integration step, we march from the $(n+1)$ step to the $(n+2)$ step, and so on, hence the name "two-stage staggered explicit procedure." The net result is that, even though we take a full step (h instead of $h/2$), we only advance half the step at a time. In other words, we evaluate the acceleration and the angular acceleration vectors twice for each full step.

V. Implicit Solution of Constraint Forces

The solution of the governing equations of motion is carried out by the two-stage explicit integration procedure presented in the preceding section. For systems with constraints, one must either eliminate the constraints or solve them as part of the system unknown. Many MBD systems involve constraints that are either difficult or computationally cumbersome to eliminate. For this reason, we will adopt the staggered stabilized procedure,[16,17] which is reviewed here for convenience. First, instead of augmenting Eq. (2) to Eq. (9), and simultaneously solving the generalized coordinates and the Lagrange multipliers, we employ a partitioned solution procedure[5,19-21] to solve the generalized coordinates separately from the Lagrange multipliers. To effect a partitioned solution of the constraints, we introduce the following penalty expression

$$\lambda_n = (1/\epsilon)\Phi_N(\dot{u},u,t), \quad \lambda_H = (1/\epsilon)\Phi_H(u,t), \quad 0 < \epsilon \ll 1 \tag{26}$$

It is noted that the penalty procedure as given by Eq. (26) tacitly assumes violations of the constraint condition [Eq. (2)] in actual computations. Now, to solve λ separately, it is necessary to cast λ in a differential form instead of in the algebraic form. This is accomplished as follows.

Instead of substituting the penalty expression directly into the governing equations of motion [Eq. (1)], first we differen-

tiate Eq. (35) once to obtain

$$\dot{\lambda}_N = \frac{1}{\epsilon}\left(B_N\ddot{u}_N + \frac{\partial \Phi_N}{\partial t}\right) \tag{27a}$$

$$\dot{\lambda}_H = \frac{1}{\epsilon}\left(B_H\ddot{u}_H + \frac{\partial \Phi_H}{\partial t}\right) \tag{27b}$$

where we assume the penalty parameter ϵ to be constant.

In practice, both the holonomic and the nonholonomic constraints may be associated with a common set of generalized coordinates. For such cases, we time-differentiate the holonomic constraints and combine those sets of Φ_H into Φ_N in Eq. (26). In this way λ_N and λ_H become uncoupled in Eq. (27)

Let use rewrite Eq. (1) in the form

$$\begin{Bmatrix}\ddot{u}_N\\ \ddot{u}_H\end{Bmatrix} = \begin{bmatrix}M_N & 0\\ 0 & M_H\end{bmatrix}^{-1}\begin{Bmatrix}f_N - \dot{p}_N - B_N^T\lambda_N\\ f_H - \dot{p}_H - B_H^T\lambda_H\end{Bmatrix} \tag{28}$$

where p is a generalized momenta

$$p = D(u) + S(u) \tag{29}$$

so that, upon substituting Eq. (28a) into Eq. (27a) for the nonholonomic case, one obtains

$$\epsilon\dot{\lambda}_N + B_N M_N^{-1}B_N^T\lambda_N = r_{\lambda_N} = B_N M_N^{-1}(f_N - \dot{p}_N) + \frac{\partial \Phi_N}{\partial t} \tag{30}$$

For the holonomic case, we integrate $\ddot{u}_H$ once by the midpoint implicit formula [see e.g., (Eq. (19a)] to obtain

$$\dot{u}_N^{n+1/2} = \dot{u}_N^n + (h/2)\ddot{u}_N^{n+1/2} = \dot{u}_N^n + (h/2)M_H^{-1}(f_H^{n+1/2} - \dot{p}_H^{n+1/2} - B_H^T\lambda_H^{n+1/2}) \tag{31}$$

Substituting Eq. (31) into the Eq. (27b), we obtain

$$\epsilon\dot{\lambda}_H^{n+1/2} + \frac{h}{2}B_H M_H^{-1}B_H^T\lambda_H^{n+1/2} = r_{\lambda_H} = B_H\left[\dot{u}_N^n + \frac{h}{2}M_H^{-1}(f_H^{n+1/2}) - \dot{p}_H^{n+1/2})\right] + \frac{\partial \Phi_H}{\partial t} \tag{32}$$

Equations (30) and (32) can be written as

$$\epsilon\dot{\lambda} + BM^{-1}B^T\lambda = r_\lambda \tag{33}$$

Integration of the preceding equation by the midpoint implicit rule yields the following difference equation:

$$(\epsilon I + (h/4)\,BM^{-1}B^T)\lambda^{n+1/4} = (h/4)(r_\lambda^{n+1/2} + r_\lambda^n) + \epsilon\lambda^n \tag{34a}$$

$$\lambda^{n+1/2} = 2\lambda^{n+1/4} - \lambda_n \tag{34b}$$

It has been shown that the preceding staggered, stabilized procedure for the solution of the constraints offers not only a modular software package to treat the constraints but also yields more robust solutions compared to the techniques proposed by Baumgarte.[2,3] In particular, even when $BM^{-1}B^T$ becomes nearly singular, the staggered stabilized procedure [Eq. (34)] gives stable and acceptable solutions, whereas the constraint forces computed by the Baumgarte's technique diverge.

The present explicit-implicit, staggered procedure given by Eqs. (11), (15), and (19) together with the constraint solver [Eq. (9)] constitutes a complete solution procedure for a multibody dynamics analysis for systems with constraints that undergo large motions.

VI. Computer Implementation and Performance Evaluations

The preceding procedure for the numerical integration of the equations of motion for constrained MBD systems has been implemented in two separate integration modules: generalized coordinate integrator (CINT) and Lagrange multiplier solver (LINT). The CINT employs a two-stage modified form of the central-difference method for updating the angular velocity vector and the midpoint-implicit rule for updating the angular orientations via the Euler parameters. The Lagrange multiplier solver adopts a staggered form of the midpoint implicit method. It should be noted that CINT needs the constraint force vector, viz, $f_\lambda = B^T\lambda$, as an applied force from LINT. Similarly, LINT needs the generalized coordinates and their time derivatives from CINT. Hence, the step advancing of the present procedure is accomplished in a staggered manner.

The module LINT receives $f_\lambda^n = B^T\lambda^n$ from LINT and advances the solution of the MBD equation [(1) or (4)] from time t^n to t^{n+1}. Once $(\dot{d}, d, \omega, q)$ are available at time t^{n+1} from CINT, LINT computes the Lagrange multipliers from Eq. (34).

To complete the solution of both the generalized coordinates and the Lagrange multipliers, we invoke the following sequence calls:

$t = t^n$
Call CINT (p^n, g^n, h, p^{n+1})
Call LINT $(\ell^{n+1/2}, h, \lambda^{n+1/2}, f_\lambda^{n+1/2})$
$t = t^n + h/2\ (n \leftarrow n + 1/2)$
Call CINT $(p^{n+1/2}, g^{n+1/2}, h, p^{n+3/2})$
Call LINT $(\ell^{n+1}, h, \lambda^{n+1}, f_\lambda^{n+1})$
$t = t^n + h$

where

$$p^n = (\dot{d}^{n-1/2}, d^n, \omega^{n-1/2}, q^n)$$

$$g^n = [\omega^n, f_\lambda^n = B^T\lambda^n)]$$

$$\ell^{n+1/2} = (\dot{d}^{n+1/2}, d^{n+1/2}, \omega^{n+1/2}, q^{n+1/2}, \lambda^n)$$

In summary, the present procedure requires two function evaluations and two λ-solutions per each full step, hence the name "explicit-implicit staggered procedure." We now present three sample problems whose efficient and accurate solutions will confirm in their combined totality not only the viability of the present integration procedure for the solution of the multibody equations of motion with or without constraints, but also the constraint stabilization procedure.

A. Dynamics of a Bowling Ball

This problem was investigated by Huston et al.,[10] however, their equations do not involve the constraint force λ. In the present analysis, we employ a formulation that incorporates the constraint force as part of the system variables. Figure 1 illustrates the ball, with its radius a and an offset center r_0 that is to follow a sine curve,

$$\Phi = y - \sin x = 0 \tag{35}$$

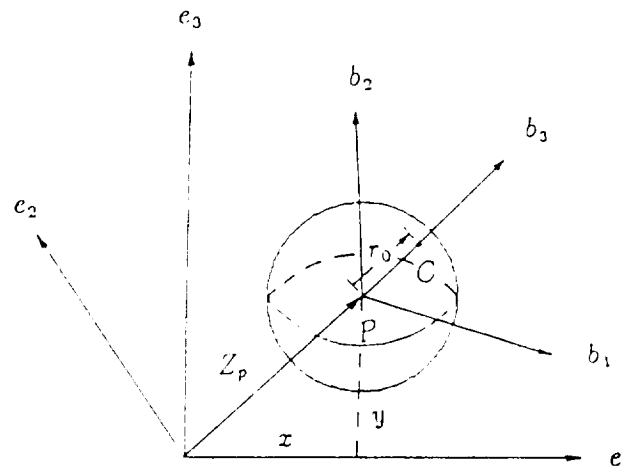


Fig. 1 Solid spherical ball rolling on a flat surface.

Table 1 Physical dimensions and initial conditions for a rolling sphere

$m = 71.32$ N,	$a = 10.9$ cm,	$r_0 = 0$ or 0.15 cm
$J_1 = J_2 = J_3 = 2/5\ ma^2$,	$\epsilon = 10^{-6}$	
$x^0 = y^0 = 0$,	$\{\omega_2^0 = -\omega_1^0 = 1,\ \omega_3^0 = 0\}$	
$\dot{x}^0 = \dot{y}^0 = a\omega_1^0$,	$\{q_0^0 = 1,\ q_1^0 = q_2^0 = q_3^0 = 0\}$	

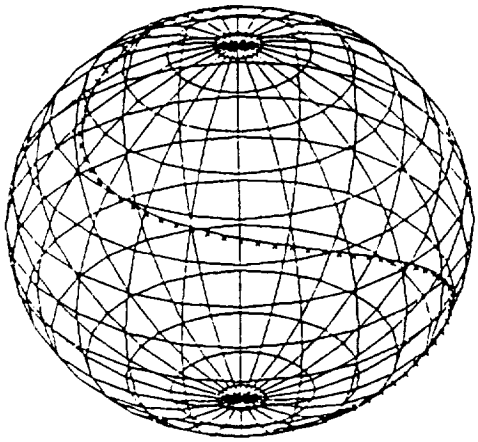


Fig. 2 Ball track projected on three-dimensional sphere surface.

The various matrices and vector quantities for Eqs. (26) and (35) can be derived as

$$M = \begin{bmatrix} m & 0 & -mr_0\boldsymbol{e}_1\cdot\boldsymbol{b}_2 & mr_0\boldsymbol{e}_1\cdot\boldsymbol{b}_1 & 0 \\ 0 & m & -mr_0\boldsymbol{e}_2\cdot\boldsymbol{b}_2 & mr_0\boldsymbol{e}_2\cdot\boldsymbol{b}_1 & 0 \\ & & J_1 & 0 & 0 \\ & \text{sym.} & & J_2 & 0 \\ & & & & J_3 \end{bmatrix} \tag{36a}$$

$$B = \begin{bmatrix} 1 & 0 & -a\boldsymbol{b}_1\cdot\boldsymbol{e}_2 & -a\boldsymbol{b}_2\cdot\boldsymbol{e}_2 & -a\boldsymbol{b}_3\cdot\boldsymbol{e}_2 \\ 0 & 1 & a\boldsymbol{b}_1\cdot\boldsymbol{e}_1 & a\boldsymbol{b}_2\cdot\boldsymbol{e}_1 & a\boldsymbol{b}_3\cdot\boldsymbol{e}_1 \\ \cos x & -1 & 0 & 0 & 0 \end{bmatrix} \tag{36b}$$

$$F_d = -mr_0 \begin{Bmatrix} \omega_1\omega_3\boldsymbol{e}_1\cdot\boldsymbol{b}_1 + \omega_2\omega_3\boldsymbol{e}_1\cdot\boldsymbol{b}_2 - (\omega_1^2+\omega_2^2)\boldsymbol{e}_1\cdot\boldsymbol{b}_3 \\ \omega_1\omega_3\boldsymbol{e}_2\cdot\boldsymbol{b}_1 + \omega_2\omega_3\boldsymbol{e}_2\cdot\boldsymbol{b}_2 - (\omega_1^2+\omega_2^2)\boldsymbol{e}_2\cdot\boldsymbol{b}_3 \end{Bmatrix} \tag{37}$$

$$F_\omega = -\begin{Bmatrix} \omega_2\omega_3(J_2 - J_3) \\ \omega_3\omega_1(J_3 - J_1) \\ \omega_1\omega_2(J_1 - J_2) \end{Bmatrix}$$

$$f_d = 0, \qquad f_\omega = mgr_0 \begin{Bmatrix} \boldsymbol{e}_3\cdot\boldsymbol{b}_2 \\ -\boldsymbol{e}_3\cdot\boldsymbol{b}_1 \\ 0 \end{Bmatrix} \tag{38}$$

$$d = \lfloor x, y \rfloor^T, \quad \omega = \lfloor \omega_1 \quad \omega_2 \quad \omega_3 \rfloor^T, \quad \lambda = \lfloor \lambda_1 \quad \lambda_2 \quad \lambda_3 \rfloor \tag{39}$$

where the inertial-basis vector $\boldsymbol{e}$ and the corotational-basis vector $\boldsymbol{b}$ are related according to

$$\boldsymbol{b} = \mathrm{R}\boldsymbol{e} \tag{40}$$

There is a total of eight variables to describe the equations of motion for the constrainted ball. However, in adopting the present solution procedure—viz, Eqs. (1-3)—we solve for nine variables as we employ the four Euler parameters for angular orientations.

Numerical solutions of the rolling of a sphere on a flat sinusoidal curve have been obtained with the data summarized in Table 1.

The ball track that follows the constraint sinusoidal curve [Eq. (26)] is projected on the ball itself as shown in Fig. 2, with the corresponding angular velocities in Fig. 3. The time histories of the three constraint forces are shown in Fig. 4, where λ_1 and λ_2 correspond to the x and y components of the constraint forces in order to maintain the rolling-contact condition, and λ_3 corresponds to the constraint force to maintain the sinusoidal trajectory as imposed by Eq. (26). Hence, the first two constraints are indicative of the skidding phenomenon, whereas the third corresponds to the steering force required to maneuver the ball. Notice that, although periodic, they exhibit highly nonlinear behavior.

We have performed convergence studies with increasing step sizes; these indicate that the present two-stage staggered explicit procedure—viz, Eqs. (1-3)—maintains both the solution accuracy and stability for the step size up to $h \leq 0.15$.

Figure 5 shows the angular velocities for a ball with an offset center ($r_0 = 0/15a$). Note that the angular velocities no longer exhibit periodic response, whereas for the no-offset case they are periodic (see Fig. 3). Likewise, the steering force causing the ball to follow the sinusoidal curve ($y = \sin x$) becomes highly nonlinear (see Fig. 6) although it is nonlinearly periodic. The x- and y-direction contact forces, which maintain the rolling-contact condition between the ball and the

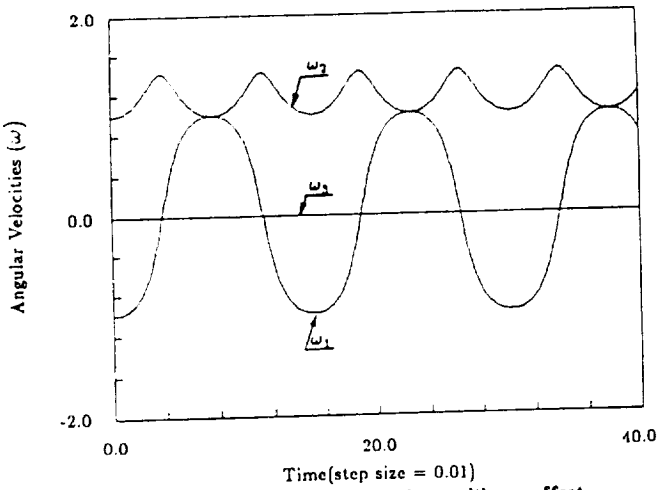


Fig. 3 Angular velocities of the sphere with no offset.

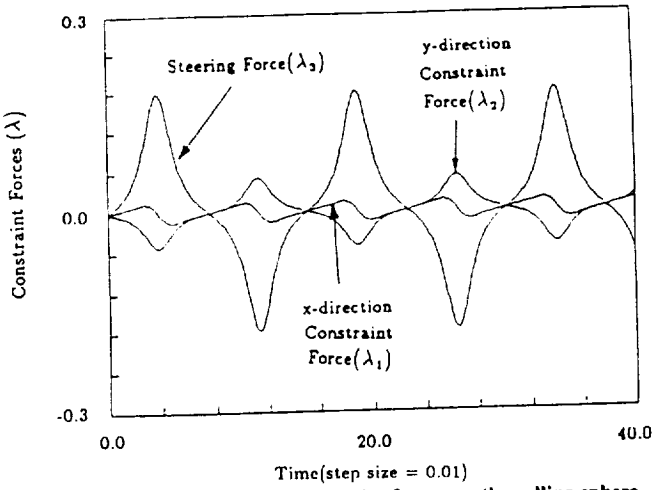


Fig. 4 Time histories of three constraint forces on the rolling sphere.

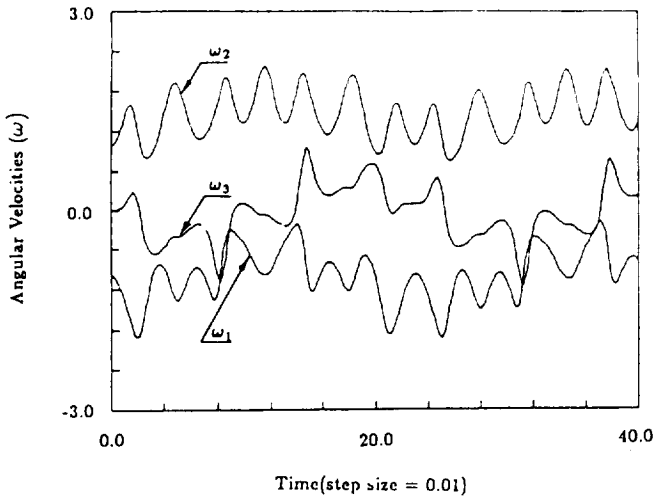


Fig. 5 Angular velocities of the sphere with offset center.

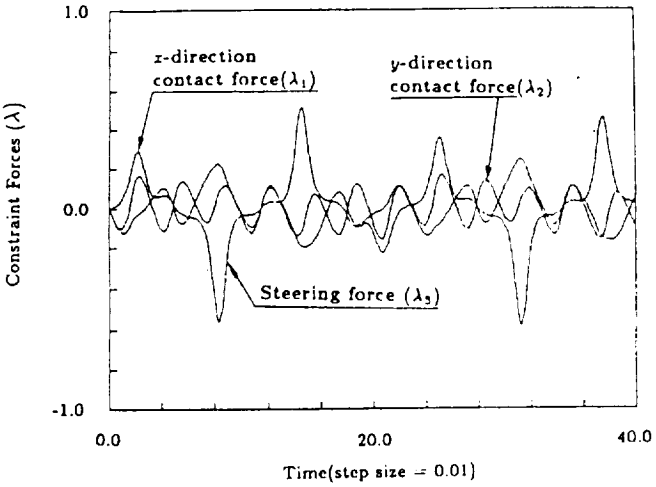


Fig. 6 Time histories of three constraint forces with offset center.

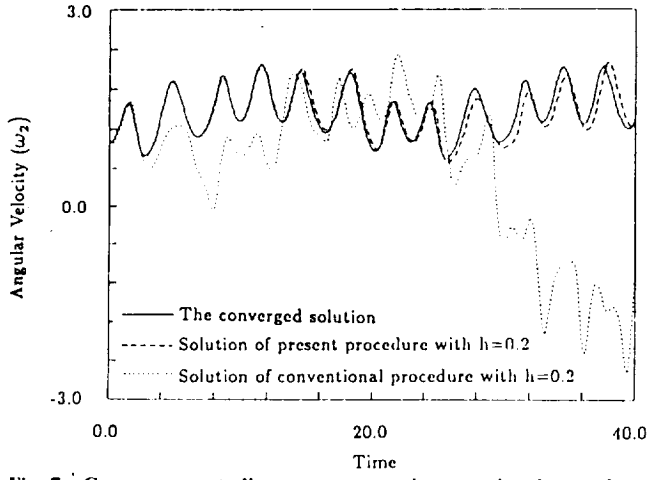


Fig. 7 Convergence studies on present and conventional procedure.

surface, although bounded, manifest extremely nonlinear behavior.

To corroborate the instability of a naive approximation of $\omega^n \times \omega^{n-1/2}$ for the computation of D_ω in computing $\dot{\omega}^n$—as alluded to in Sec. IV.B, the equations of motion for the rolling ball have been integrated by the following formula:

$$\omega^{n+1/2} \approx \omega^{n-1/2} - hM_\omega^{-1}[f_\omega^n + B(\omega^{n-1/2}, d^n)\lambda^n - F(\omega^{n-1/2})] \tag{41}$$

Figure 7 shows ω_2 vs time for the converged solution, the present two-stage, explicit-implicit, staggered procedure with step size ($h = 0.2$), and the conventional procedure with step size ($h = 0.2$). The diverging solution by the conventional procedure is clearly manifested, thus confirming the instability of the conventional procedure. On the other hand, the present staggered procedure faithfully traces the converged solution.

Finally, the solution accuracy vs the step size has been assessed for the offset center ball with different step sizes. Figure 8 represents the performance of the present procedure for different step sizes. Note that if one chooses the step size that corresponds to more than 15 samples per period, viz., $h \leq 0.2$, a reasonable engineering accuracy can be maintained. Although not reported herein, the problem was also solved by the trapezoidal rule. For $h \geq 0.1$, the computational overhead with the trapezoidal rule was an order of magnitude higher than by the present two-stage staggered explicit-implicit procedure without an accuracy improvement. Our experience with the example problem indicates that the present computational procedure for handling large rotational and translational motions with constraints is robust and efficient. It is important to note that the present procedure accurately traces not only the angular motions but, more important, the constraint forces and the four Euler parameters (although these are not presented here).

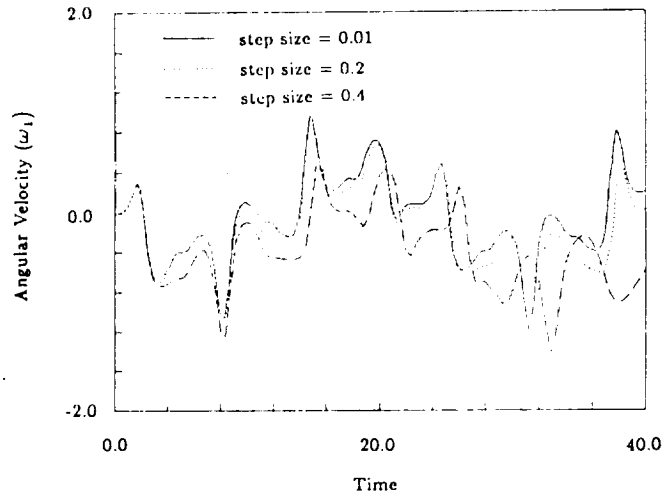


Fig. 8 Accuracy comparison on angular velocity ω_1 for three different step sizes.

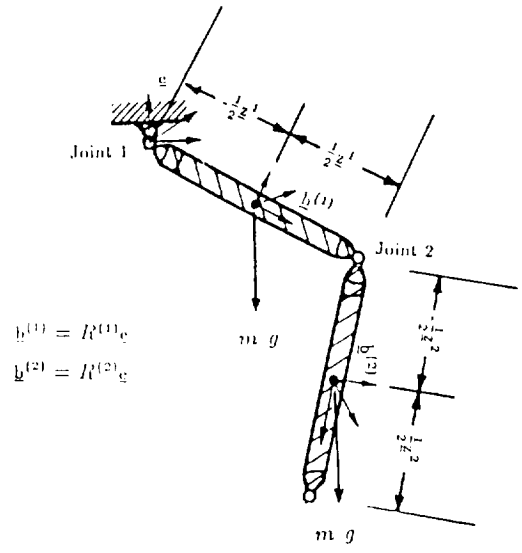


Fig. 9 Double pendulum with spatial joints.

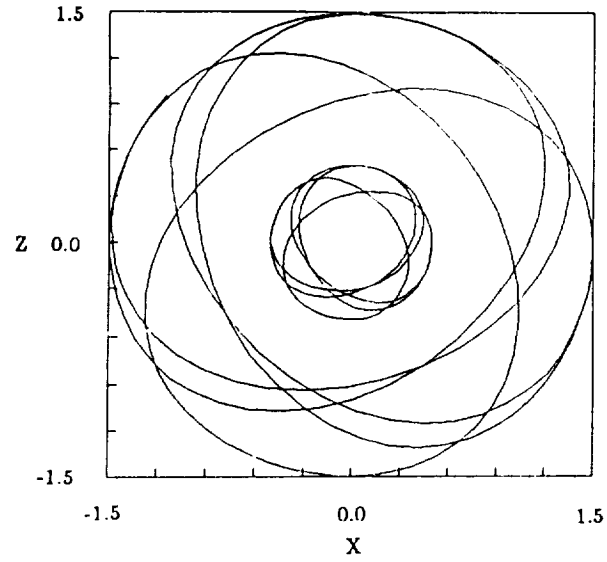


Fig. 10a Trajectories of double pendulum on X-Z plane.

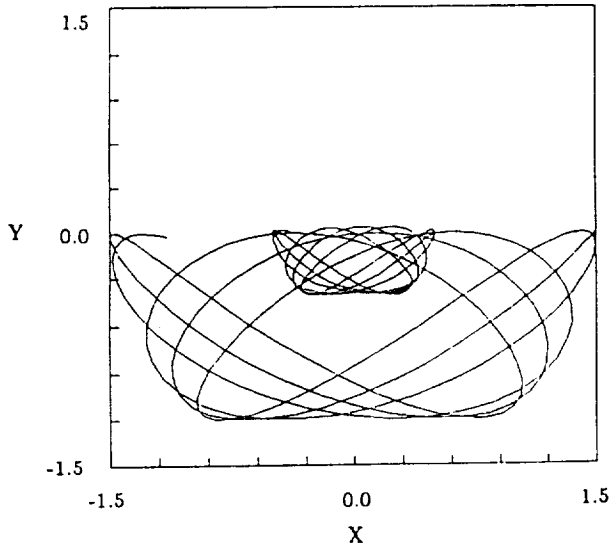


Fig. 10b Trajectories of double pendulum on X-Y plane.

B. Three-Dimensional Double Pendulum

The second problem with which we have tested the present procedure is a spatially moving double pendulum as shown in Fig. 9. The governing equations of motion become those of two separate rigid bars, except that they are connected by two spherical joints. From Fig. 9 we have the following quantities:

$$\Phi^i = \dot{d}^i - \tfrac{1}{2}\,\omega^i \times z^i = 0, \qquad i = 1,2 \tag{42}$$

$$M = \text{diag}\left\{m^1, J^1, m^2, J^2\right\} \tag{43}$$

$$B = \begin{bmatrix} I & \tfrac{1}{2}\tilde{z}^1 \times & 0 & 0 \\ I & -\tfrac{1}{2}\tilde{z}^1 \times & -I & -\tfrac{1}{2}\tilde{z}^2 \times \end{bmatrix} \tag{44}$$

$$F\omega = \begin{Bmatrix} f^1 \\ f^2 \end{Bmatrix}$$

$$f^i = -\begin{bmatrix} 0 \\ 0 \\ 0 \\ \omega_2\omega_3(J_2 - J_3) \\ \omega_3\omega_1(J_3 - J_1) \\ \omega_1\omega_2(J_1 - J_2) \end{bmatrix}^i, \qquad i = 1, 2 \tag{45}$$

$$\ddot{u}^i = \left\{\ddot{d}, \dot{\omega}\right\}^i, \quad \ddot{d}^i = [\ddot{x}, \ddot{y}, \ddot{z}]^T, \quad \dot{\omega}^i = [\dot{\omega}_1, \dot{\omega}_2, \dot{\omega}_3]^T \tag{46}$$

$$\lambda = [\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6]^T \tag{47}$$

In the preceding equations, $\frac{1}{2}z$ is the vectorial distance from the center of the bar to the spherical joint constraints, m and J are the three translational and rotatory-inertia matrices, $\tilde{z}$ is the skew symmetric matrix formed by the three components of z, $\times$ implies a vector cross multiplication, and the superscript designates the ith bar.

The pendulum is originally positioned in a gravity field with initial horizontal angular velocities ($\omega_z^{(1)} = \omega_z^{(2)} = 1$). Figure 10 shows the spatial trajectories of the two mass centers as projected on the horizontal surface and on the vertical plane. It is noted that the two trajectories form a similar pattern. The constraint forces and angular velocities, although not reported here, exhibit patterns that are analogous in their characteristics for the two joints and two mass centers, respectively.

We have performed convergence studies by using different step sizes h. Numerical evaluations indicate, as with the rolling-ball problem, that when the step-size samples are more than 20/period, the present procedure yields both good accuracy and stability.

C. Closed Four-Bar Linkage

The final problem is a simple closed four-bar linkage, composed of four individual bars connected with five spherical joints (see Fig. 11). The governing equations of motion for this problem are identical with those of the previous section, except that the gradient of the constraint equations B is given by

$$B = \begin{bmatrix} B_b^1 & & 0 & & \\ B_u^1 & B_t^2 & & & \\ & B_t^2 & B_t^3 & & \\ & & B_u^3 & B_t^4 & \\ & 0 & & B_u^4 & \end{bmatrix} \tag{48}$$

where

$$B_b^i = \left\{I, \tfrac{1}{2}\tilde{z} \times\right\}, \quad B_u^i = \left\{I, -\tfrac{1}{2}\tilde{z} \times\right\}$$
$$B_t^i = \left\{-I, -\tfrac{1}{2}\tilde{z} \times\right\} \tag{49}$$

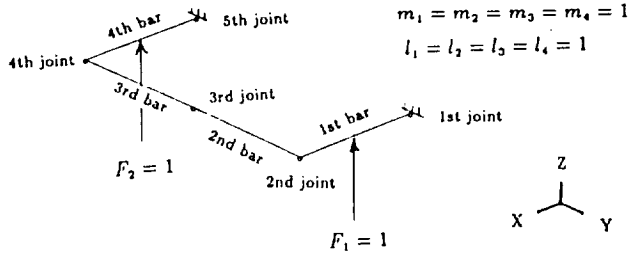


Fig. 11a Initial configuration of the closed four-bar linkage.

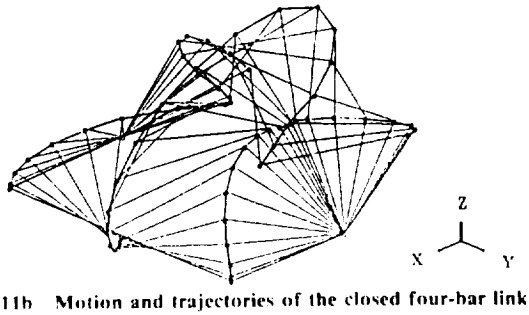


Fig. 11b Motion and trajectories of the closed four-bar linkage.

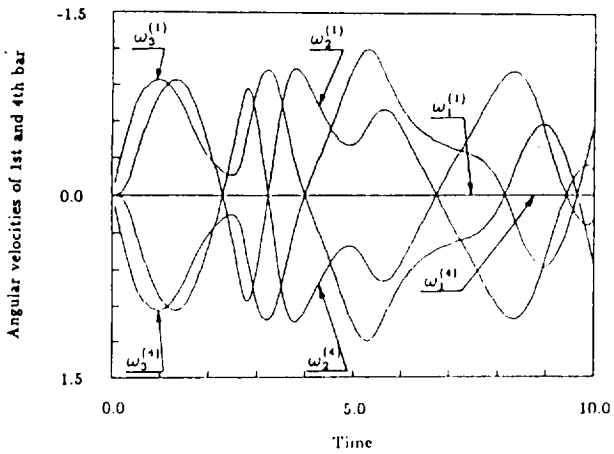


Fig. 12a Angular velocities of the closed four-bar linkage.

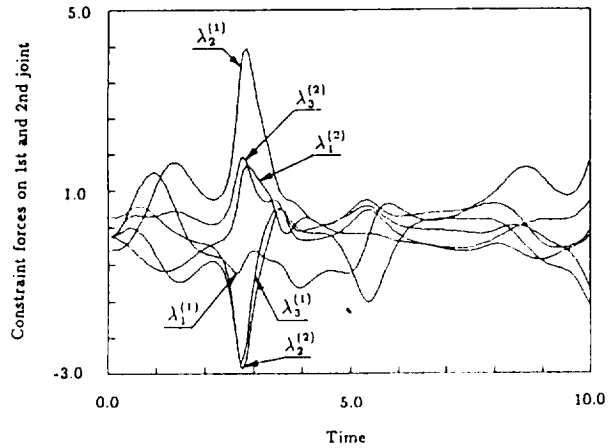


Fig. 12b Constraint forces of the closed four-bar linkage.

The body-fixed coordinates and constraint conditions for this problem have adopted the same procedure as in the preceding pendulum problem. To trigger large rotational motions, two vertical forces ($F_y^{(1)} = F_y^{(2)} = 1$) are applied at the center of mass of the first and fourth bar (see Fig. 11a). Figure 11b indicates the motion of each bar for 8 s run time. Note that the trajectories of each joint can also be seen from the plot. Because of the symmetry of the geometry and the applied forces, one should expect corresponding symmetries between the angular velocities of the first bar compared with those of the fourth bar, and so on (see Fig. 12a). This is also the case with the constraint forces as manifested in Fig. 12b.

We investigated numerical solutions for different step sizes h. The results show that when step size h is less than 0.075, the procedure proposed here maintains stability with acceptable accuracy.

VII. Discussion

In this paper, we have presented a computational procedure for direct integration of the MBD equations with constraints. Because of its step-advancing nature, the procedure is labeled an explicit-implicit staggered algorithm explicit for solving the CINT and implicit for Lagrange multipliers to incorporate constraints (LINT). The present generalized coordinate solver (CINT) carries out its task in a partitioned manner in which the translational motions are integrated separately from those of the rotational parameters.

Numerical experiments reported herein and additional applications investigated so far indicate that the present procedure yields robust solutions if the step size gives more than 20 samples for the period of the apparent highest response frequency of a given multibody system.[21] Hence, the present procedure appears to have accomplished the following.

Because of the modular implementation of the present MBD solution procedure, the task of interfacing the present MBD solution modules with additional capabilities such as active controller, observer, and other analysis and design software modules becomes relatively straightforward. Such software architecture is in contrast to most existing programming practice in which several analysis capabilities are embedded into a single monolithic program.

For closed-loop multibody systems and/or problems with complex topology, in which it is impractical and inadvisable to eliminate the constraints, the present procedure facilitates a straightforward construction of the governing equations of motion with appropriate constraints. The generalized coordinates and the Lagrange multipliers can then be solved in a partitioned manner.

The update of angular orientations is based on the Euler parameters by adopting the midpoint implicit formula. This avoids potential computational complications, as the angular orientation matrices remain singularity free.

Application of the present procedure to flexible multibody systems is currently being carried out, and preliminary results are quite encouraging. We hope to report in the near future on results with flexible-body dynamics as well as on results with large-scale multibody problems.

Finally, a preliminary stability analysis of the present procedure, although not reported here, has been conducted. The analysis results indicate that the procedure is stable provided the step size satisfies

$$h \leq 2/(\omega_1^2 + \omega_2^2 + \omega_3^2)^{1/2} \tag{50}$$

A separate article on the stability issue is presently under preparation; we plan to publish it in the near future.

Acknowledgments

The work reported herein was supported by the NASA Langley Research Center under Grant NAG-1-756. The authors wish to thank Jerry Housner and Jeff Stroud for their keen interest and encouragement during the course of the present work. We also thank the reviewers for constructive comments that have led to an improvement of this paper.

References

[1]Armstrong, W. W., "Recursive Solution to the Equations of Motion of an n-Link Manipulator," *Proceedings of the 5th World Congress, Theory of Machines, Mechanisms*, Vol. 2, July 1979, pp. 1343-1346.

[2]Baumgarte, J. W., "A New Method of Stabilization for Holonomic Constraints," *Journal of Applied Mechanics*, Vol. 50, Dec. 1983, pp. 869-870.

[3]Baumgarte, J. W., "Stabilization of Constraints and Integrals of Motion in Dynamical Systems," *Computer Methods in Applied Mechanics Engineering*, Vol. 1, 1972, pp. 1-16.

[4]Bodley, C. S., Devers, A. D., Park, A. C., and Frish, H. P., "A Digital Computer Program for the Dynamic Interaction Simulation of Controls and Structures (DISCOS)," NASA TP 1219, May 1978.

[5]Felippa, C. A., and Park, K. C., "Staggered Transient Analysis Procedures for Coupled Mechanical Systems," *Computer Methods in Applied Mechanics and Engineering*, Vol. 24, Oct. 1980, pp. 61-111.

[6]Gear, C. W., "Simultaneous Numerical Solution of Differential/Algebraic Equations," *IEEE Transactions Circuit Theory*, CT-18, 1971, pp. 89-95.

[7]Haug, E. J. (ed.), *Computer Aided Analysis and Optimization of Mechanical System Dynamics*, Springer-Verlag, Berlin, 1984.

[8]Henrici, P., *Discrete Variable Methods in Ordinary Differential Equations*, New York, 1962, pp. 336-339.

[9]Hollerbach, J. M., "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10, 1980, pp. 730-736.

[10]Huston, R. L., Passerello, C. Winget, J. M., and Sears, J., "On the Dynamics of a Weighted Bowling Ball," *Journal of Applied Mechanics*, Vol. 46, Dec. 1979, pp. 937-943.

[11]Huston, R. L., *Lecture Notes on Dynamics*, Preprint, Univ. of Cincinnatti, Cincinnati, OH, 1988.

[12]Kane, T., and Levinson, D., "Formulation of Equations of Motion for Complex Spacecraft," *Journal of Guidance and Control*, Vol. 3, No. 2, 1980, pp. 99-112.

[13]Nikravesh, P. E., "Some Methods for Dynamic Analysis of Constrained Mechanical Systems: A Survey," *Computer Aided Analysis and Optimization of Mechanical System Dynamics*, NATO ASI series, F9, edited by E. J. Haug, Springer-Verlag, Berlin, 1984, pp. 351-367.

[14]Orin, D. E., McGhee, R. B., Vukobratovic, M., and Hartoch, G., "Kinematic and Kinetic Analysis of Open-Chain Linkages Utilizing Newton-Euler Methods," *Mathematics of Bioscience*, Vol. 43, 1979, pp. 106-130.

[15]Orlandea, N., Chase, M. A., and Calahan, D. A., "A Sparsity-oriented Approach to the Dynamic Analysis and Design of Mechanical Systems—Part I and II," *Transactions of the American Society of Mechanical Engineers for Industry, Ser. B*, Vol. 99, 1977, pp. 773-784.

[16]Park, K. C., and Chiou, J. C., "Evaluation of Constraint Stabilization Procedures for Multibody Dynamical Systems," *Proceedings of the AIAA 28th Structures, Structural Dynamics and Materials Conference*, Part 2A, AIAA, New York, 1987, pp. 769-733.

[17]Park, K. C., and Chiou, J. C., "Stabilization of Computational Procedures for Constrained Dynamical Systems," *Journal of Guidance, Control and Dynamics*, Vol. 11, No. 4, 1988, pp. 365-370.

[18]Park, K. C., "Partitioned Analysis Procedures for Coupled-Field Problems: Stability Analysis," *Journal of Applied Mechanics*, Vol. 47, June 1980, pp. 370-378.

[19]Park, K. C., and Felippa, C. A., "Partitioned Analysis of Coupled Systems," *Computational Methods for Transient Analysis*, edited by T. Belytschko and T. J. R. Hughes, Elsevier, 1983, pp. 157-219.

[20]Park, K. C., Felippa, C. A., and DeRuntz, J. A., "Stabilization of Staggered Solution Procedures for Fluid-Structure Interaction Analysis," *Computational Methods for Fluid-Structure Interaction Problems*, edited by T. Belytschko, and T. L. Geers, American Society of Mechanical Engineers, AMD Vol. 26, New York, 1977, pp. 95-124.

[21]Park, K. C., and Underwood, P. G., "A Variable-Step Central Difference Method for Structural Dynamics Analysis—Part I. Theoretical Aspects," *Computer Methods in Applied Mechanics and Engineering*, Vol. 22, Sept. 1980, pp. 241-258.

[22]Petzold, L., "Differential/Algebraic Equations Are Not ODEs" *SIAM Journal of Scientific Statistical Computing*, Vol. 3, 1982, pp. 367-384.

[23]Schwertassek, R., and Roberson, R. E., "A State-Space Dynamical Representation for Multibody Mechanical Systems, Part II," *Acta Mechanica*, Vol. 51, 1984, pp. 15-29.

[24]Walton, W. C., and Steeves, E. C., "A New Matrix Theorem and its Application for Establishing Independent Coordinates for Complex Dynamical Systems with Constraints," NASA TR-R326, 1969.

[25]Wehage, R. A., and Haug, E. J., "Generalized Coordinate Partitioning for Dimension Reduction in Analysis of Constrained Dynamic Systems," *ASME Journal of Mechanical Design*, Vol. 104, Jan. 1982, pp. 247-255.

[26]Wittenburg, J., *Dynamics of Systems of Rigid Bodies*, B. G. Teubner, Stuttgart, 1977.

TRANSIENT FINITE ELEMENT COMPUTATIONS ON 65536 PROCESSORS: THE CONNECTION MACHINE

Charbel Farhat, N. Sobh and K. C. Park
Department of Aerospace Engineering Sciences
and Center for Space Structures and Controls
University of Colorado at Boulder
Boulder, CO 80309-0429

Abstract. This paper reports on our experience in solving large-scale finite element transient problems on the Connection Machine. We begin with an overview of this massively parallel processor and emphasize the features which are most relevant to finite element computations. These include virtual processors, parallel disk I/O and parallel scientific visualization capabilities. We introduce a distributed data structure and discuss a strategy for mapping thousands of processors onto a discretized structure. The combination of the parallel data structure with the virtual processor mapping algorithm is shown to play a pivotal role in efficiently achieving massively parallel explicit computations on irregular and hybrid two- and three-dimensional finite element meshes. The finite element kernels written in C* have run with success to solve several examples of linear and nonlinear dynamic simulations of large problem sizes. From these example runs, we have been able to assess in detail their performance on the Connection Machine. We show that mesh irregularities induce an MIMD (Multiple Instruction Multiple Data) style of programming which impacts negatively the performance of this SIMD (Single Instruction Multiple Data) machine. Finally, we address some important theoretical and implementational issues that will materially advance the application ranges of finite element computations on this highly parallel processor.

I. INTRODUCTION

Parallel computers are having a profound impact on computational mechanics. This is reflected by the continuously increasing number of publications on finite elements and parallel processing. Not only have some computational strategies been re-designed for implementation on commercially available multiprocessors, but also some innovative algorithms have been spurred by the advent of these new

machines. However, many of the reported parallel finite element simulations have been on systems with a few processors. Examples of these systems are Intel's iPSC with 32 processors (reported by Farhat and Wilson [1]), JPL/Caltech's hypercube with 32 processors (Lyzenga, Raefsky and Hager [2], and Nour-Omid and Park [33]), Alliant's FX8 model with 8 processors (Belytschko and Gilbersten [3], and Farhat and Crivelli [4]), and CRAY's systems with up to 4 processors (Benten, Farhat and Jordan [5]). (For more complete lists of references on this topic see White and Abel [6] and Noor [7].) While great speed-ups were measured on these coarse to medium grain machines, Farhat [8] has shown that traditional vector supercomputers could not be outperformed in finite element simulations (except of course on systems which connect more than one vector superprocessor, such as the CRAY X-MP and CRAY-2 systems, each of which has 4).

Recently, massively parallel machines have demonstrated their potential to be the fastest supercomputers, a trend that may accelerate in the future. While solving the shallow water equations, McBryan has reported that the Connection Machine (CM_2 in the sequel) (65536 processors) was three times faster than the four-processor CRAY X-MP [9]. Gustafson, Montry and Benner have developed highly parallel solutions for baffled surface wave equations, unstable fluid flow and beam strain analysis, and have reported performances on NCUBE's 1024-processor hypercube which are close to those of vector supercomputers [10].

The objective of the present study has been: first, to evaluate the multiprocessing features of the CM_2 that are relevant to finite element computations, second, to develop a suitable finite element data structure which exploits the system architecture, third, to implement a decomposition/mapping procedure that matches as far as possible the layout of the processors to the finite element meshes, and fourth, to assess those implications of finite element analysis on the CM_2 that should be considered in the design of future massively parallel processors. Hence, we focus primarily on implementational issues that are critical for the full exploration of the multiprocessing capabilities of the CM_2, and only secondarily on solution algorithms, as far as they impact the present study on implementational issues.

The finite element equations of motion for structural systems can be expressed as:

$$\mathbf{M}\ddot{\mathbf{d}} + \mathbf{F}^{in}(\dot{\mathbf{d}}, \mathbf{d}) = \mathbf{F}^{ex} \tag{1}$$

where $\mathbf{M}$ denotes the positive definite lumped mass matrix, $\mathbf{F}^{in}$ and $\mathbf{F}^{ex}$ denote the internal and external force vectors, and $\ddot{\mathbf{d}}$, $\dot{\mathbf{d}}$ and $\mathbf{d}$ denote respectively the

acceleration, velocity and displacement vectors. In the linear case, the internal force vector becomes:

$$\mathbf{F}^{in} = \mathbf{D}\dot{\mathbf{d}} + \mathbf{K}\mathbf{d} \tag{2}$$

where $\mathbf{D}$ and $\mathbf{K}$ are the damping and stiffness matrices respectively, which are positive semi definite. In this work, an eventual damping is assumed to be proportional to the mass and stiffness.

The algorithmic nature of a candidate solution method for the structural dynamics equation (1) can significantly influence the software requirements, data communications and arithmetic efficiency. As our main focus is on implementational issues rather than algorithmic ones, we have decided on a simple explicit time integration procedure. Hence, we choose to integrate equation (1) with the fixed step explicit central difference algorithm because (a) it is inherently parallel, and (b) it has the largest undamped stability limit among second-order accurate explicit linear multistep algorithms, as has been demonstrated by Krieg [11] and Park [12]. In our context, it is expressed as:

$$\begin{aligned} \dot{\mathbf{d}}^{n+1/2} &= \dot{\mathbf{d}}^{n-1/2} + h\mathbf{M}^{-1}(\mathbf{F}^{ex}(t^n) - \mathbf{F}^{in}(\dot{\mathbf{d}}^n, \mathbf{d}^n)) \\ \mathbf{d}^{n+1} &= \mathbf{d}^n + h\dot{\mathbf{d}}^{n+1/2} \end{aligned} \tag{3}$$

where h is the fixed time step and the superscript n indicates the value at the discrete time t^n.

The remainder of this paper deals with the massively parallel solution of (1) using (3), and is organized as follows. In Section II, we give an overview of the CM_2 hardware configuration and empasize those features which are pertinent to finite element computations. In particular, we address issues that are related to the processor memory size, to the SIMD architecture, and to the fast interprocessor communication package, the *NEWS grid*. In Section III, we discuss the floating point arithmetic performance of the CM_2 and highlight its current dependence on the selected language compiler. Algebraic manipulations coded in *Lisp are shown to be three times as fast as when written in C*. A general purpose finite element distributed data structure is presented in Section IV. Designed originally to handle massively parallel finite element explicit computations on irregular and hybrid meshes, this parallel data structure is also very efficient for parallel I/O manipulations and parallel graphic animation. Since the often-encountered mesh irregularities inhibit the use of the *NEWS grid* communication package, we discuss in Section V an alternative decomposition/mapping strategy.

The decomposition technique is designed to minimize both the amount of communication between different chips and the amount of wire contention within a chip. The mapping algorithm attempts to reduce the distance that information must travel. Section VI summarizes the overall organization of the massively parallel transient simulation. In Section VII, our parallel data structure and processor mapping are applied to (3) for the solution of various large-scale transient problems. Measured performances are analyzed in detail. Mesh irregularities are shown to be the source of several factors which considerably slow down the machine. Finally, in Section VIII, we address some important theoretical and implementational issues that will materially advance the application ranges of finite element computations on the CM_2. In particular, we note that time integration numerical algorithms such as explicit finite differences and equation solvers such as the preconditioned conjugate gradient are implemented using the same parallel data structure and mapping algorithm which are presented in this paper. We compare the substructuring technique and the virtual processor approach, and comment on the implications of implicit algorithms for the effective use of the CM_2.

II. THE CONNECTION MACHINE HARDWARE ARCHITECTURE

Here we present an overview of the CM_2 system organization and discuss issues that are pertinent to massively parallel finite element computations. See Hillis [13] for an indepth discussion on the rationale behind the CM_1 (a previous model of the Connection Machine), the Technical Summary of Thinking Machines Corporation [14] for further architectural information, and McBryan [9] for initial studies of scientific computations on the CM_1. For the sake of clarity, we summarize the architectural features before discussing their impact on finite element simulations.

II.1. System Organization

II.1.1 CM_2: The Parallel Processing Unit

The CM_2 is a cube 1.5 meters on a side, made of up to eight subcubes (fig. 1). Each subcube contains 512 chips and every chip includes 16 bit serial processors which are connected by a switch. Each individual processor has 64 Kbits (8 Kbytes) of bit-addressable local memory and an arithmetic-logic unit (ALU) that can operate on variable-length operands. Every two chips may share an optional Weitek floating point accelerator chip. A fully configured CM_2 thus has 4096 (2^{12}) chips, 2048 floating point accelerator chips, 65536 processors, and 512

Mbytes of memory. The chips are arranged in a 12 dimensional hypercube. A chip i is directly connected to 12 other chips j, with the binary representation of i and j differing only by 1 bit.

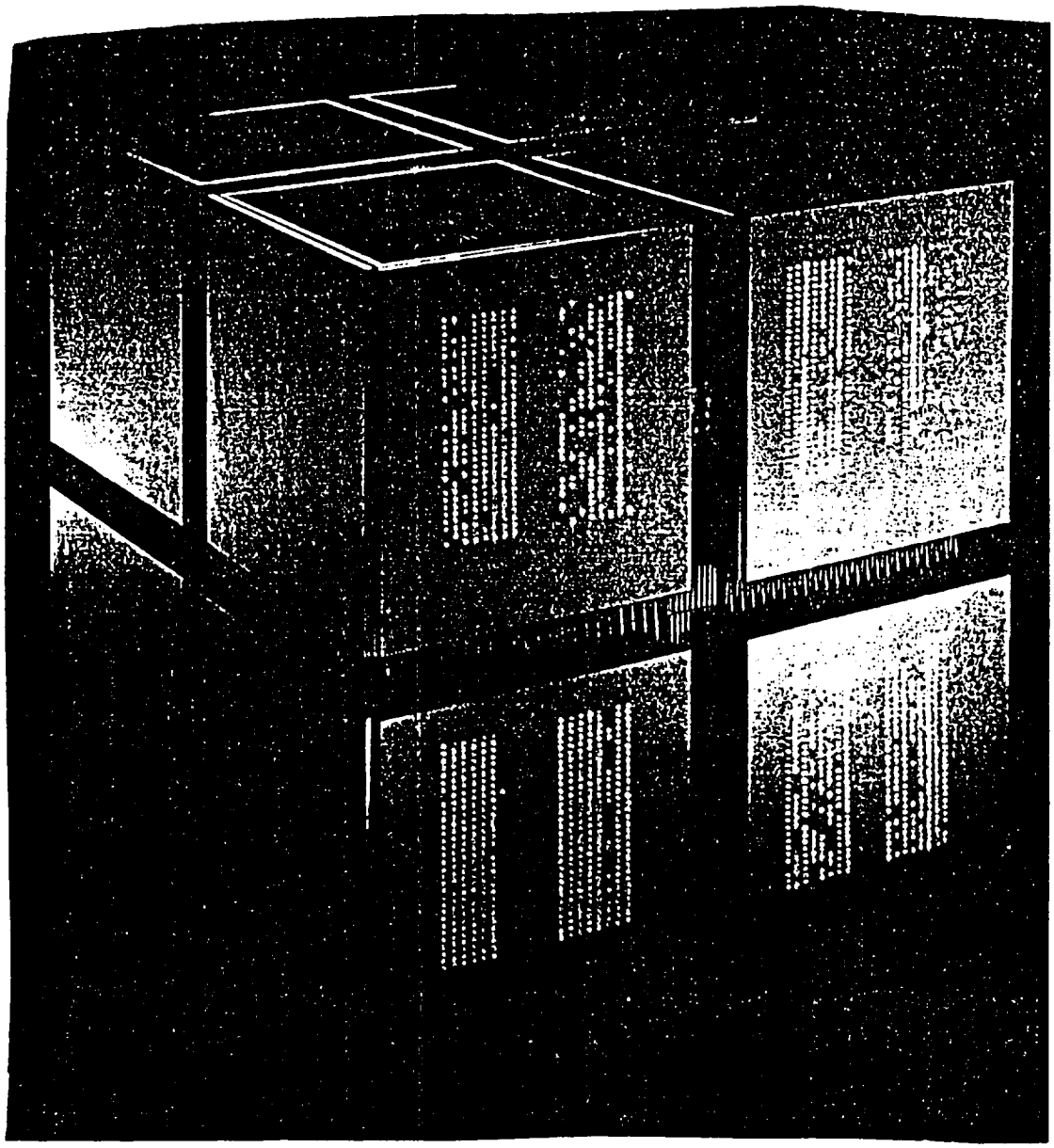


FIG. 1. The CM_2

The CM_2 system provides two forms of communication between the processors:

- a general mechanism known as the *router* which allows any processor to communicate with any other processor. Each CM_2 chip contains one router node i which serves the 16 processors on the chip, numbered $16i$ through $16i + 15$. The router nodes on all the chips are wired together in a 12-dimensional boolean cube and together form the complete router network (fig. 2). For example, suppose that processor 117 (processor 5 on router node 7), has a message to send to processor 361 (processor 9 on router node 22). Since $22 = 7 + 2^4 - 2^0$, router 7 forwards the message to router 6 ($6 = 7 - 2^0$) which forwards it to router 22 ($6 + 2^4$), which delivers the message to processor 361.
- a more structured and somewhat faster communication mechanism called the *NEWS grid*. Each processor is wired to its four nearest neighbors in a two-dimensional rectangular grid (fig. 3). Communication on the *NEWS grid* is extremely fast and recommended whenever it is possible.

An important practical feature of the CM_2 is the support for *virtual* processors. When the CM_2 is initialized for a run, the number of virtual processors (vp in the sequel) may be specified. If it exceeds the number of available physical processors, then the local memory of each processor is split up into a number of regions equal to the ratio between the number of vps and the number of physical processors. Automatically, for every Paris (PARallel Instruction Set) instruction, the processors are time-sliced among the regions. If a physical processor is simulating N vps, each Paris instruction is decoded by the sequencer (as explained below) only once for N executions. This results in an enhanced *user* performance. Also, the use of a vp > 1 allows the pipelining of floating point operations in the Weitek chips, which provides an additional enhancement to *machine* performance.

The CM_2 is an SIMD machine. All processors must execute identical instructions or some processors may choose to ignore any instruction. Consequently, an instruction which involves a nested binary branch can see its exection time increased by a factor of two. The SIMD nature of the CM_2 has some disadvantages in finite element computations, as will be shown.

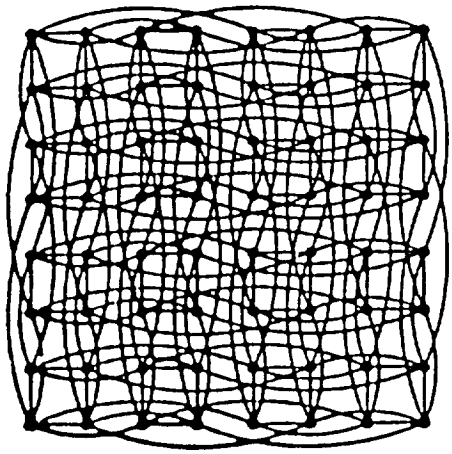


FIG. 2. The Router Network

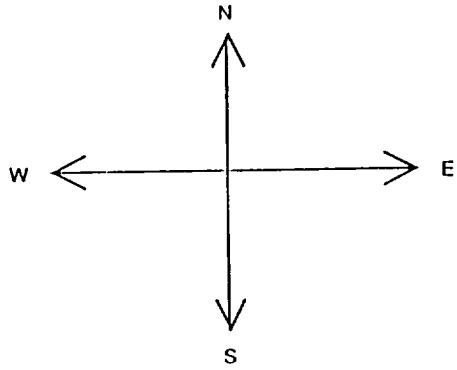


FIG. 3. The NEWS Grid

II.1.2 The Front End Computer

The parallel processing unit described above is designed to operate under the programmed control of a front-end computer (FE in the sequel) which may be either a Symbolics 3600 Lisp Machine or a DEC VAX 8000 series computer. The FE provides the program development and execution environment. It transmits instructions and associated data to the CM_2. Instructions from Paris are not handled directly by the CM_2. After they are issued from the FE, they are processed by a *sequencer* which broadcasts them to the CM_2 in the form of low level operations.

II.1.3 The Data Vault System

I/O has traditionally been the Achilles heel of computers and supercomputers. Moreover, it is very well known that I/O manipulations can easily dominate the execution time of a finite element code. The CM_2 I/O system appears to offer hope for the solution of this problem.

The Data Vault is the CM_2 mass storage system. Each Data Vault unit is associated with one eighth of a fully configured CM_2. It stores its data in an array of 39 individual disk drives. With this disk farming system, the concept of performing parallel I/O is carried through: instead of regarding a file as a serial stream of bits, the CM_2 file system regards it as many streams of bits, which are read or written in parallel, one stream per processor. When eight Data Vaults operate in parallel, they offer a combined data transfer rate of 320 mbytes per second and hold up to 80 gigabytes of data.

II.1.4 The Graphic Display System

The CM_2 graphic display system known as the *Frame Buffer* also incorporates the concept of parallelism. It allows the user to visualize on a color monitor screen the data in the processors. The display can be updated as computations are performed. We have found this tool very useful, not only for real-time animations, but also for debugging purposes.

The system organization of a CM_2 is summarized in figure 4.

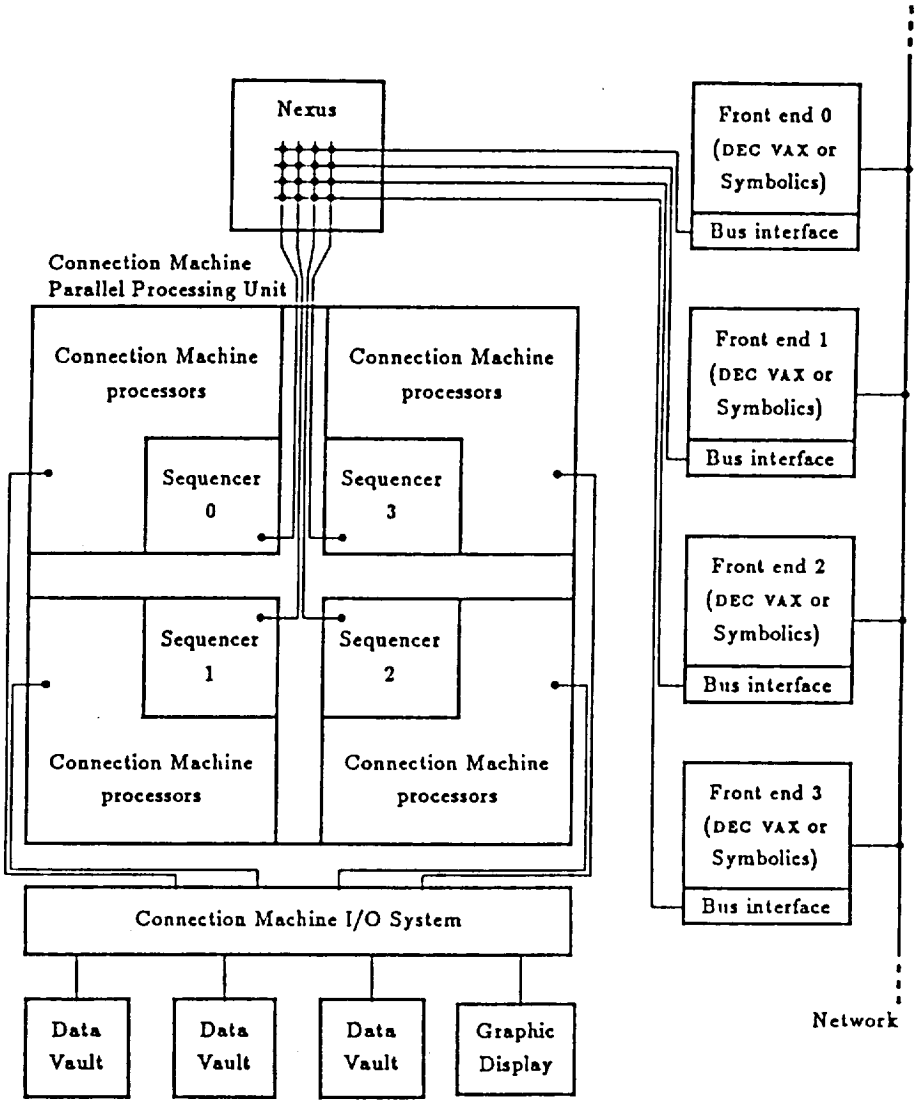


FIG. 4. System Organization of a CM-2

II.2. Impact on Finite Element Computations

It is well-known that the solution algorithm (3) can be implemented using only element-level computations. Hence, if each vp of the CM_2 is mapped onto one finite element, equation (1) can be efficiently integrated in parallel. The rationale behind this processor-to-element assignment will be analyzed in Sections IV and VIII. Here, we discuss the direct impact of the CM_2 hardware on such a decision.

The Local Memory and Element Level Computations

Consider the 9-node curved shell element shown in figure 5.

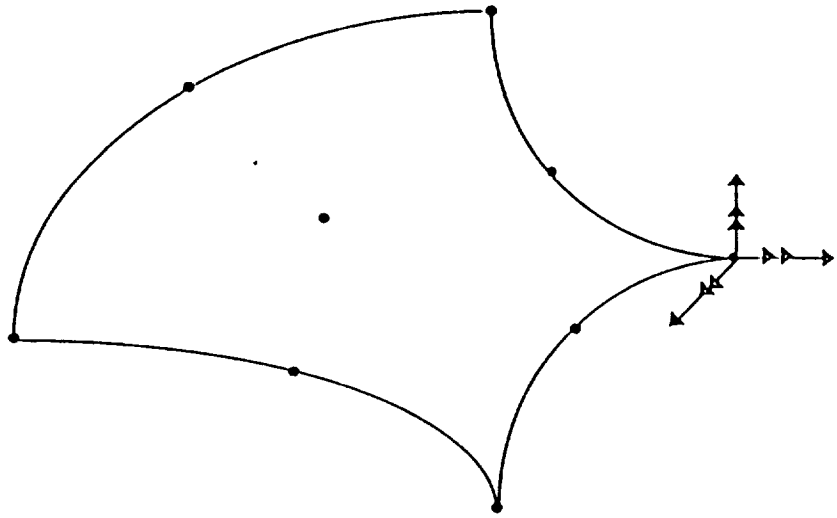


FIG. 5. A 9-Node Shell Element

Three displacements and two rotations are attributed to each node, which amounts to a total of 45 degrees of freedom per element. Consequently, the symmetric part of the elemental stiffness matrix, $\mathbf{K}^{(e)}$, contains 45*(45 + 1)/2 = 1035 words. If double precision is used, the storage of $\mathbf{K}^{(e)}$ amounts to 1035*64 = 66240 bits, which exceeds the 65536 bits that are available on a single CM_2 processor. On the other hand, if single precision is used, the storage of $\mathbf{K}^{(e)}$ requires 33120 bits, so that 32416 bits are left for the storage of the vectors $\mathbf{d}^{(e)}$, $\dot{\mathbf{d}}^{(e)}$, the elemental lumped mass vector $\mathbf{M}^{(e)}$, and the forces $\mathbf{F}^{ex(e)}$ and $\mathbf{F}^{in(e)}$. However, even in the latter case, only a vp ratio of 1 can be used. This limits the size of the finite element mesh to the maximum number of processors available on

the CM_2 at hand. Also, it inhibits further performance enhancement as outlined in Section II.1.

Fortunately, in our case the above storage requirements can be considerably decreased. The nature of explicit computations is such that $\mathbf{F}^{in}(\mathbf{d}^n)$ can be directly computed from the displacements at t^n and the stress-strain constitutive equation. As a result, the solution process defined in (3) involves only vector quantities which do not require a large amount of storage, so that vp ratios between 1 and 4 are possible. However, the reader should keep in mind that the current local memory size of a CM_2 processor may penalize sophisticated high order elements and implicit finite element algorithms in general. This restriction is not encountered on other commercially available hypercubes such as iPSC, NCUBE and AMETEK among others.

The NEWS Grid and Finite Element Patches

Consider the regular finite element mesh shown in figure 6. Except on the boundaries, each element is connected in the same pattern to exactly eight other elements. Consequently, during the explicit time integration algorithm, each processor communicates with its neighbors in the same manner. Interprocessor communication can be performed with a two step *NEWS* mechanism (fig. 6).

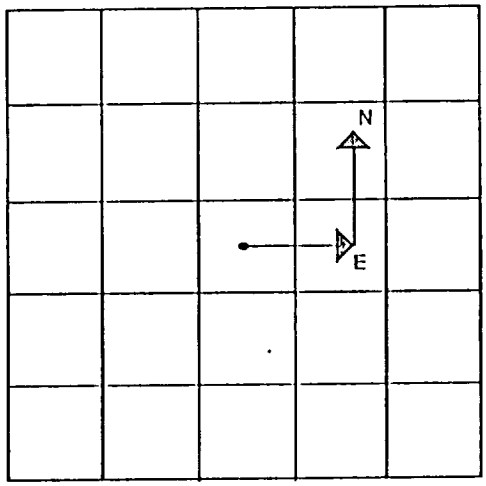


FIG. 6. A Two Step NEWS Mechanism on a Regular Mesh

However, the beauty of the finite element method resides in the fact that it solves models with irregular meshes. Typically, a finite element mesh consists of several

patches which are connected together using irregular transition regions (fig. 7). For these often encountered cases, the *NEWS grid* becomes impractical. Rather, the *router* has to be utilized. In Section IV, we describe how a distributed data structure can guide the router during this process.

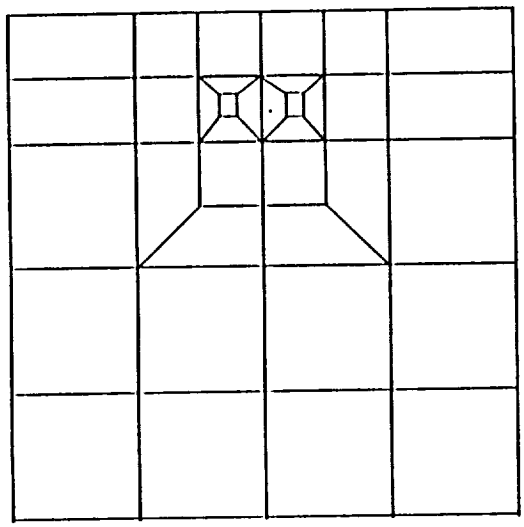


FIG. 7. Transition Zones

SIMD Hardware vs. MIMD Finite Element Computations

Typical finite element meshes comprise more than one type of element. Consider the case where a discretized region is modeled with shell elements that are stiffened with beam elements. Clearly, the instructions associated with the shell elements differ from those associated with the beam elements. Consequently, the vps which are assigned to shell elements and the vps which are assigned to beam elements cannot execute their segments of code in parallel; for example, the beam processors have to execute first, then the shell processors. If T_b and T_s denote the execution times associated with the instructions for a beam and a shell element respectively, the total elapsed parallel time for a single instruction over the set (beams + shells) on an SIMD multiprocessor is $T_b + T_s$. On an MIMD multiprocessor, this elapsed parallel time is $max(T_b + T_s)$. Similar situations arise when during the loading some elements turn to be materially nonlinear and some remain linear. In this case, one should always compute the linear component of the response (the elastic stiffness for example) before attempting to test the yielding criterion. However, in spite of these disadvantages SIMD programs can

still be attractive, because they tend to be easier to debug and rarely suffer from the synchronization errors which are typical of MIMD codes.

Parallel I/O in Finite Element Computations

At each time step, the computed displacements, velocities, accelerations as well as strains and stresses need to be stored on disks. This represents a significant amount of I/O traffic. It has been our experience that the CM_2 Data Vault system is efficient at reducing the corresponding elapsed time (see Section VII).

Real-time Graphic Animations

The massively parallel real-time animation of the mesh deformations is a direct consequence of the availability of the *Frame Buffer* and the decision of assigning a vp to a finite element. At each time step, after the node displacements are found all of the vps concurrently draw the outline of their assigned elements on the graphic screen. The result is a real-time finite element animation.

III. BENCHMARKING THE CM_2

At the time of writing this paper, the CM_2 supports three high level languages: C* (pronounced see-star), *Lisp (pronounced star-lisp), and CM-Lisp (pronounced see-m-lisp). The first two are extensions of C and Lisp respectively. Paris is somewhat the assembly language of this parallel processor.

In this section we comment on the results of a set of timing experiments that were carried out on the CM_2 of the Center for Applied Parallel Processing (CAPP), at the University of Colorado, Boulder. Since only one eighth of a cube was available on this system, all results were obtained using 8192 processors. McBryan [9] has shown that all results demonstrated on subcubes of the CM_2 scale essentially linearly to the 65536 processor system. Consequently, throughout this paper, megaflop rates are reported after they are linearly scaled to the full configuration. These experiments provided us with:

- a reference performance for the evaluation of our approach to massively parallel finite element explicit computations.
- the influence of the vp ratio and that of the high level language compiler on attainable performances. At this point, we remind the reader that, if an application requires an amount of local memory (per processor) m_a, the highest vp ratio possible is equal to the closest power of two to the ratio

between the maximum amount of local memory available on the machine (currently 8 Kbytes), and m_a.

Table 1 reports the megaflop rates for some scientific computations on the CM_2 at different vp ratios. All statements were written in C*. Each statement is performed by each processor on its variables. All variables were declared parallel (local) and float (simple precision), except variable *dp* which was declared mono (serial) float, and variable *i* which was declared mono integer. Timings were measured using the *cmtimer* routines. Each '+' operation or '*' operation was counted as one flop.

TABLE 1. Megaflop Rates Using C*

Parallel Processor = CM_2 - Language = C* - Variable = float									
Statement	Vp Ratio								
	1	2	4	8	16	32	64	128	256
$y[i] += \alpha * x[i]$	740	808	848	850	880	-	-	-	-
$y = y + \alpha * x$	569	654	699	728	743	761	778	791	800
$z = x * y$	409	485	535	569	579	585	600	610	623
$dp \mathrel{+}= x * y$	202	359	583	839	1075	1240	1348	1400	1500

Based on these results, we have observed the following:

1. Floating point performance is enhanced at higher vp ratios. This is due to the fact that for vp ratios greater than one, computations in the Weitek chip are pipelined.
2. vector saxpys are not slower than scalar ones. This is because memory addresses are computed on the front end. The additional speed noticed for

vector saxpys is thought to be due to the overlapping of addressing and floating point computations.

3. C* appears to handle poly (parallel) assignments poorly. This can be seen by comparing the performances of the dot product and the vector multiply. Each of these two vector operations requires one floating point per processor. In addition, the dot product requires a reduction (accumulation phase) which necessitates communication. However, at high vp ratios, the dot product is twice as fast as the vector multiply! (At low vp ratios, the amount of floating point computations is not large enough to amortize the price of communication.) Since the dot product does not store any value in the processor memory and the vector multiply stores the result of $x * y$ back into z, this leads us to believe that the C* compiler generates a code which is very inefficient at handling assignments. This also explains why the saxpy exhibits a higher megaflop rate than the vector multiply: it has twice as many floating point computations for one assignment.

The same computations were repeated using *Lisp. The comparison of both sets of timings for the maximum vp demonstrates a formidable superiority of the *Lisp compiler (see fig. 8). This is partly due to the fact that it has been used longer on the CM_2 than C*. In spite of the proven superior efficiency of *Lisp over C*, we have chosen to implement our finite element code using C* because of our familiarity with C.

FIG. 8. A Comparison of *Lisp and C* Performances

IV. FINITE ELEMENT PARALLEL DATA STRUCTURES

Consider again the explicit central difference algorithm:

$$\begin{aligned} \dot{\mathbf{d}}^{n+1/2} &= \dot{\mathbf{d}}^{n-1/2} + h\mathbf{M}^{-1}(\mathbf{F}^{ex}(t^n) - \mathbf{F}^{in}(\dot{\mathbf{d}}^n, \mathbf{d}^n)) \\ \mathbf{d}^{n+1} &= \mathbf{d}^n + h\dot{\mathbf{d}}^{n+1/2} \end{aligned} \tag{4}$$

The global mass matrix M is assembled once. At each time step t^n, the computations are dominated by the evaluation of the internal forces:

$$\mathbf{F}^{in} = \sum_{e=1}^{e=n_{el}} \int_{\Omega^{(e)}} [LS]^T \sigma \, d\Omega$$

where σ is the stress vector, S are the shape functions, L is a partial derivative operator and $\Omega^{(e)}$ is the area of the $e-th$ finite element. Clearly, the parallel computation of $\mathbf{F}^{in}$ is best done element-by-element. Thus, equation (1) can be efficiently integrated in parallel if the CM_2 virtual processors are mapped onto the elements of the mesh. This is a departure from the grid point massively parallel computations advocated by Thinking Machines Corporation for the CM_2 [14]. First, all processors compute concurrently the local forces $\mathbf{F}^{ex(e)}(t^n)$ and $\mathbf{F}^{in(e)}(\dot{\mathbf{d}}^n, \mathbf{d}^n)$. Next, these contributions are accumulated through communications among processors that are mapped onto neighboring elements.

In this section, we describe the finite element data structures which we have selected to drive the massively parallel computations on the CM_2. These are element oriented, while similar data structures proposed for other hypercubes are subdomain oriented (see Farhat, Wilson and Powell [15] and Fox et al. [16]). In Section VIII, we give further comments on this difference. We group these data structures into two sets.

The first set of data structures deals with element-level parallel computations. To be able to perform locally its assigned element-level computations —that is, to perform these computations without interacting with the front-end machine—each processor must store in its own memory its element type (truss, beam, shell, ..., number of Gauss points, ...), its element material properties (density, parameters and coefficients for constitutive equations, damping characteristics, thickness, ...), its nodal geometry (nodal coordinates, number of nodes per element), and its boundary conditions (fixed/free degrees of freedom at each

node, prescribed forces at each node). This information is compacted in one-dimensional arrays. In addition, each processor must also store in its memory a set of scalars corresponding to computational parameters such as the fixed time step h, and a scalar or one-dimensional buffer for the temporary storage of messages to be passed to neighboring processors.

The second set of data structures provides the *router* with the mechanism for parallel interprocessor communication. The inability of the *NEWS grid* to handle irregular communication patterns has been addressed in Section II.2. Let p denote a virtual processor and e_p its assigned finite element. In order to exchange $\mathbf{F}^{in(e)}(\dot{\mathbf{d}}^n)$ and $\mathbf{F}^{ex(e)}(t^n)$, virtual processor p must be able to identify at run time:

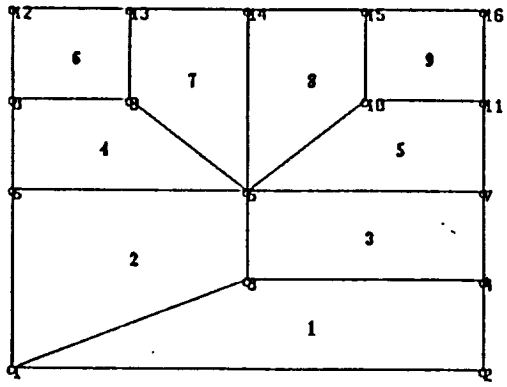
- the set of processors mapped onto elements ajacent to e_p
- the nodes that e_p shares with these elements
- at each shared node, the degrees of freedom which need to be assembled. This particular information is vital for meshes with different types of elements. It guarantees that, for example, a moment is not accumulated with a force, or that a force in the x direction is not accumulated with a force in the y direction.

If the above information is gathered in a global form on the front-end machine, most of the execution time which elapses during the accumulation phase would be due to message-passing between the CM_2 processors and the front-end computer. On the other hand, if this information is decentralized—that is, if the memory of processor p is loaded only with the subset of that information which is relevant to the connectivity of e_p—the accumulation phase can be performed without any message-passing between the CM_2 and the front-end computer. Consequently, prior to any computation, the memory of processor p is loaded with the following one-dimensional arrays:

Proc_att_to_node	For each node connected to e_p, it contains the identification of the processors that are mapped onto elements which are also connected to this node. These are stored in a stacked fashion.
Pointer	This is a pointer array. It stores in position i, the location in *Proc_att_to_node* of the list of vps that are attached to the node in the $i-th$ local position.
Location	For each entry in *Proc_att_to_node*, this array specifies the local position of the shared node in the processor that is mapped onto an element adjacent to e_p

The above arrays are set up by the dedicated finite element mesh analyzer which was presented by Farhat, Wilson and Powell [15]. They require about

80 integer words per processor. Clearly, this is a very small overhead. The mechanism of these arrays is depicted in figure 9 for element 1. The mesh patch is composed of shell and beam elements.



Element 1

Proc_att_to_node	[2,3,3,2]
Pointer	[1,2,2,3,5]
Location	[1,2,1,2]

FIG. 9. A Distributed Data Structure for Interprocessor Communication

There is however one penalty associated with assigning one element to each vp. The nodes which are common to several elements are duplicated in their corresponding processors. As a result, about 11% of the total memory available on the CM_2 is wasted. This is a small price for the highly parallel computations that are achieved. Given the low cost of memory nowadays, this seems a worthwhile trade. Moreover, this assignment allows I/O manipulations and graphic post-processing to be trivially parallelized. At each time step, after the nodal displacements are found, all of the processors draw concurrently the outline of their assigned elements on the frame buffer and send back the results to the front end in parallel.

V. THE DECOMPOSITION/MAPPING STRATEGY

Since the mesh irregularities inhibit the exploitation of the *NEWS grid*, we rely on the data structures of Section IV to guide the router during interprocessor communication. However, there is still one additional problem to resolve. Efficiency in massively parallel computations requires the minimization of both the distance that information must travel and, more importantly, the "hammering" on the router. In the case of finite element computations, this implies that adjacent elements must be assigned, as much as possible, to directly connected processors, and contention for the wire connecting neighboring chips must be reduced. This defines the mapping problem - that is, it defines which hardware processor is to be mapped onto which finite element of a given mesh.

Farhat [19] developed a heuristic algorithm for mapping massively parallel processors onto finite element graphs and presented some analytical results for corresponding efficiency improvement. Basically, the algorithm searches iteratively for a better mapping candidate through a two-step procedure for the minimization of the communication costs associated with a specific parallel processor topology. Because it seeks a very fast solution for a machine with thousands of processors, this algorithm does not guarantee "the" optimal mapping. However, it has produced very encouraging results on a variety of non-uniform two and three-dimensional meshes.

In this work, we adapt the mapping algorithm of [19] to our target parallel processor, the CM_2. The 65536 processors of this machine are packaged into 4096 16-processor chips, each having its own router node. The 4096 router nodes are arranged in a hypercube of dimension 12. To cope with this topology, we proceed in two steps. First, we decompose the given mesh into 4096 submeshes,

each containing 16 connected finite elements. Next, we apply the mapper given in [19] to identify which hardware chip is to be mapped onto which submesh. Finally, within each submesh, the elements are numbered randomly between the chip number and the chip number + 15.

Given a finite element mesh, there are several ways to decompose it into 16-element submeshes (see for example Farhat [17] and Malone [18]). Here, each submesh is to be assigned to one chip of the CM_2. In figures 10, 11 and 12, we show two different decompositions for a discretized square domain, D_1 and D_2.

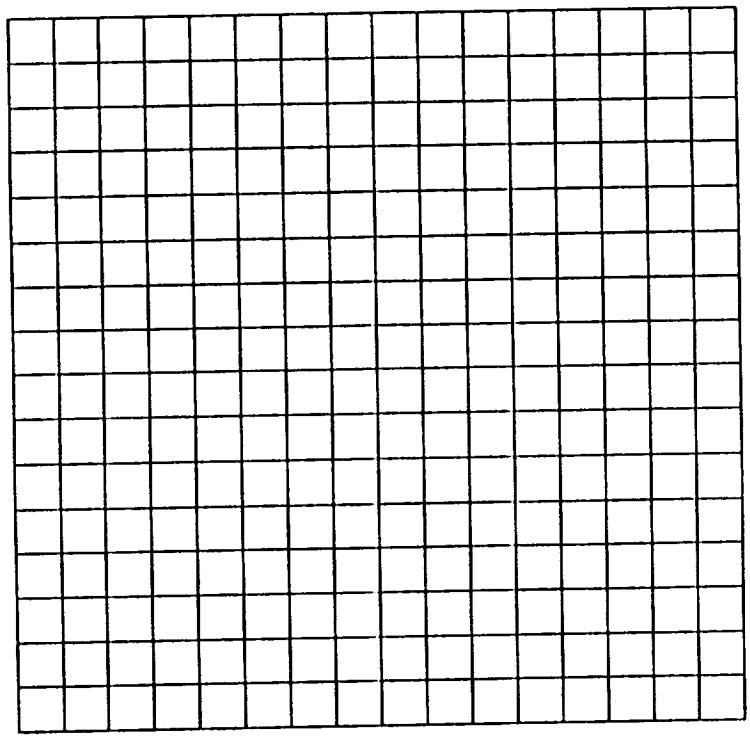


FIG. 10. Domain to be Decomposed

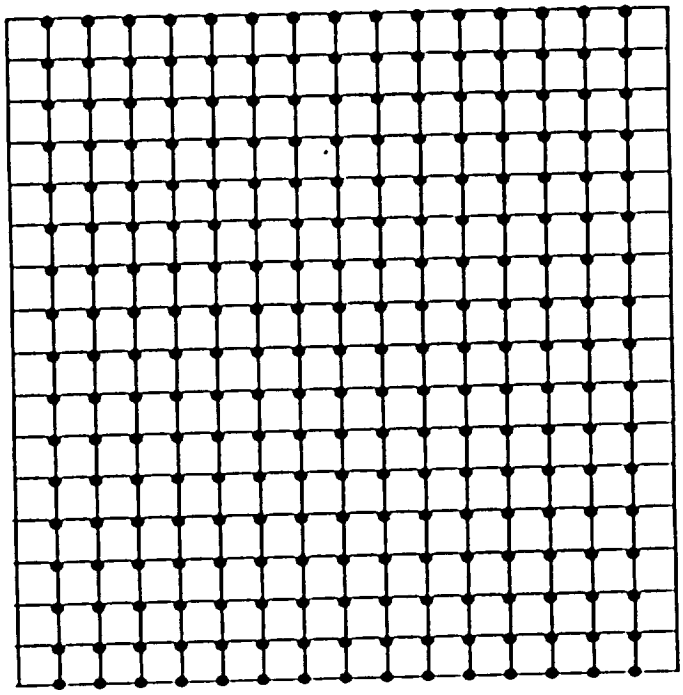


FIG. 11. Decomposition D1 - Bandwith Minimization

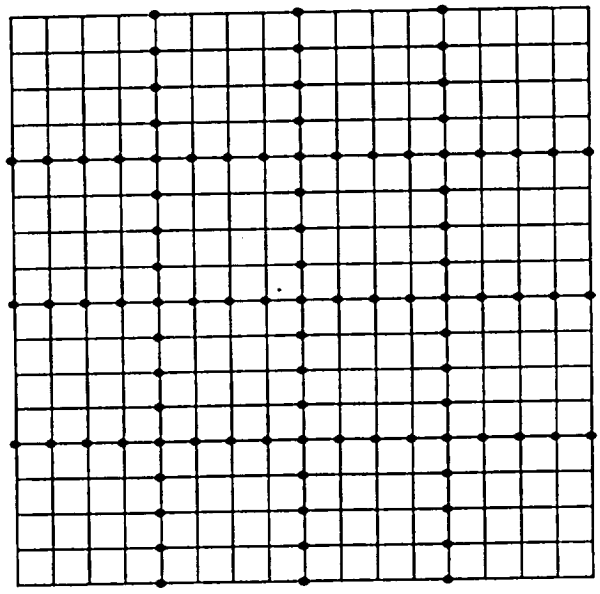


FIG. 12. Decomposition D2 - Interface Minimization

Both decompositions yield 16 submeshes, each with 16 adjacent elements. Decomposition D_1 was designed to minimize the communication bandwidth - that is, the maximum number of different chips with which any chip need to communicate. It can be seen (fig. 13) that for D_1 the bandwidth equals 2, while for D_2 it equals 8.

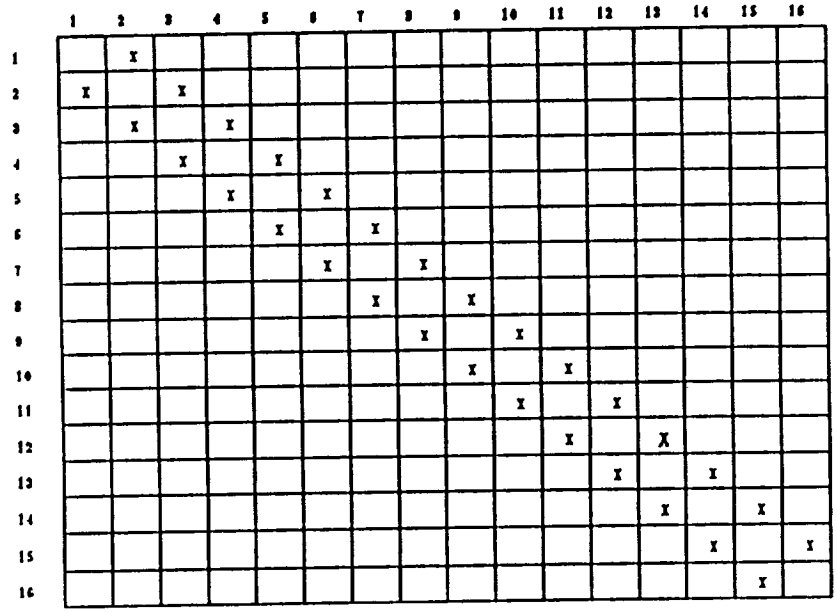


FIG. 13-a. Interchip Communication Pattern for D1

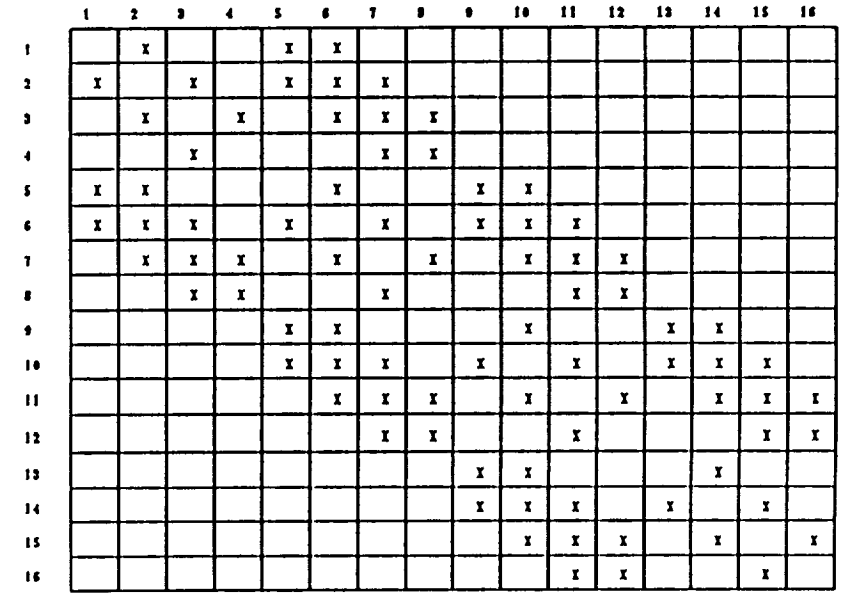


FIG. 13-b. Interchip Communication Pattern for D2

It should be remarked that, if the substructuring approach [15, 16] had been chosen — that is assigning a subdomain to a physical processor, D_1 would have been more efficient than D_2. For this decomposition, each chip would buffer the contributions of its interface nodes and send only two messages, one to the chip at its left and another to the chip at its right. The decomposition D_2 requires the same chip to send up to 8 buffered messages. These messages would eventually be shorter, but would still render D_2 more expensive because of message start-up costs. However, we have opted for a virtual processor approach — that is assigning one element to a virtual processor, for reasons that are given in Section VIII. For this case, processors exchange information one node at a time, so that the number of interface nodes associated with a decomposition is more important than its bandwidth. The reader can confirm that decomposition D_1 delivers 255 interface nodes, while D_2 delivers only 93. Indeed, there is another equally, if not more important, reason why D_2 is better for the CM-2 than D_1. In the case of D_1, all of the 16 processors of any chip communicate simultaneously with a set of processors which are on the same neighboring chip (fig. 14). This generates a significant amount of contention for the single wire that connects these two chips. In the case of D_2 however, one can observe (fig. 15) that:

- for each chip, only 12 out of the 16 processors communicate with processors onto another chip
- only 3 processors out of these 12 communicate simultaneously with the same neighboring chip, so that much less contention occurs for the wire connecting the two chips. We recall that each chip is connected with up to 12 other ones using 12 different wires which can operate in parallel.

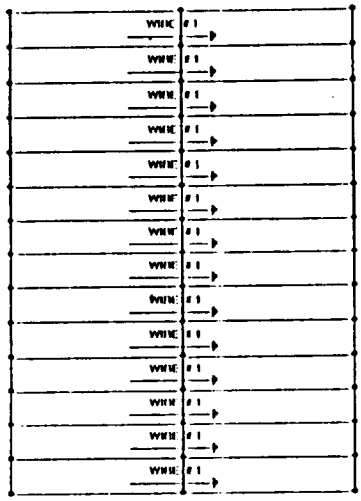


FIG. 14. Wire Contention Induced by Decomposition D1

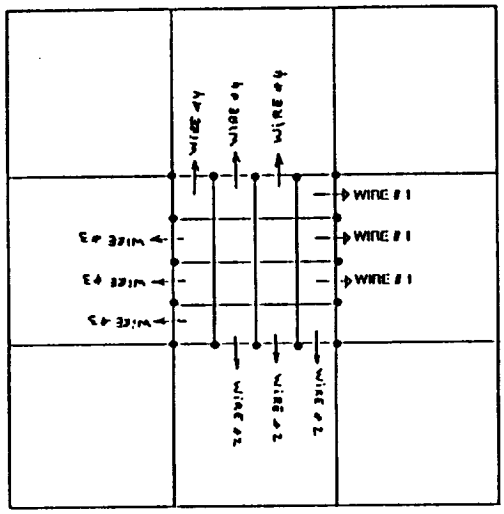


FIG. 15. Wire Traffic for Decomposition D2

The decomposition D_1 was obtained using a general purpose finite element decomposer presented by the first author in reference [17]. We advocate its use in conjunction with the mapper given in reference [19] for massively parallel computations on the CM_2. The efficiency improvement potential of this preprocessing phase is demonstrated with the following finite element wave propagation problem. Figure 16 shows the discretization of a tapered cantilever beam. The beam is modeled with 4-node isoparametric elements and linearly elastic plane stress constitutive equations. It is fixed at one end and subjected at the tip of the other to an impact point loading. The wave propagation nature of the problem dictates the meshing technique to create elements which are, as far as possible, of equal size. Since the beam is tapered, transition zones with irregular elements had to be introduced. Other mesh irregularities are due to the presence of a region with a hole. The complete mesh contains 8192 elements, which corresponds to an 8K CM_2. The use of a naive mapping (element i into processor $i-1$) would have resulted in a maximum routing distance between adjacent elements equal to 9. Our decomposer/mapper reduces this distance to 5. If EFF denotes the efficiency (speed-up per processor) of the parallel computations using a naive mapping, and

f is the factor by which the decomposer/mapper reduces the maximum routing distance between adjacent elements, the theoretical improved efficiency (Farhat, [19]) is given by:

$$EFF^* = \frac{1}{(1-\frac{1}{f})+\frac{1}{fE}} \tag{5}$$

For this problem, we have measured an efficiency $EFF = 40\%$ on an 8K CM-2. Since $f = 9/5$, the predicted improved efficiency is $EFF^* = 54\%$. A second run of the problem using the decomposer/mapper has revealed a measured improved efficiency $EFF^* = 60\%$. The discrepancy between the predicted and measured improved efficiencies is due to the fact that (5) does not account for the wire contention problem.

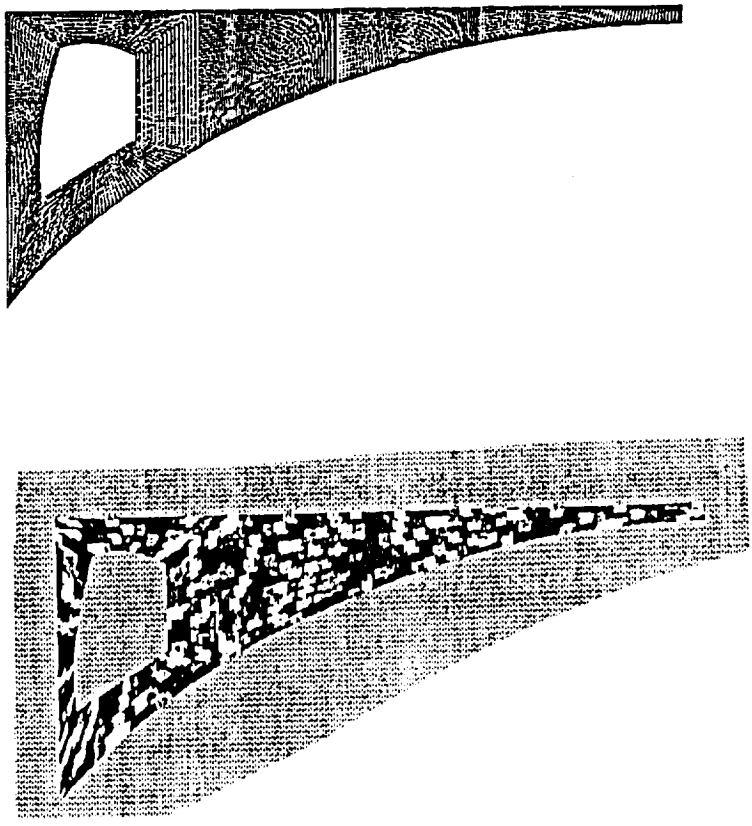


FIG. 16. Discretization and Decomposition of a Tapered Beam

VI. FLOWCHART OF THE MASSIVELY PARALLEL TRANSIENT SIMULATION

The overall organization of the solution on the CM_2 of a transient dynamic problem using the explicit central difference algorithm is depicted in figure 17. It consists of four phases, namely: mesh preprocessing, data loading, number crunching, and data unloading.

Read Input File (Front End)
Decompose Mesh and Form Parallel Data Structure (Front End)

Load Parallel Data Structure (Front End - CM_2)

Compute Lumped Mass Matrix (CM_2)
Compute Critical Time Step (CM_2)
Loop on Time Steps (Front End)
{
Compute Internal and External Local Forces (CM_2)
Assemble Global Forces (Interprocessor Communication)
Compute Velocities, Displacements, Strains and Stresses (CM_2)

Visualize Results (CM_2 - Frame Buffer)
Archive Results (CM_2 - Data Vault)
}

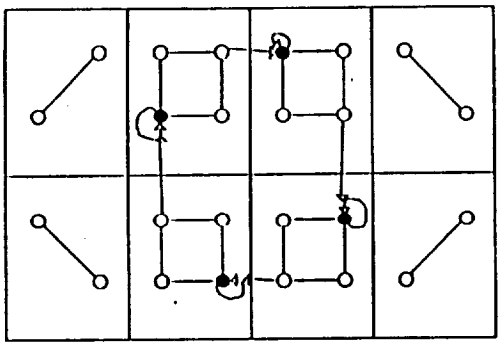
FIG. 17. Solution of a Transient Problem on the CM_2

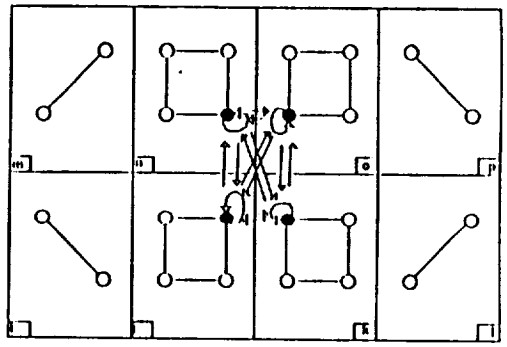
A conservative stable time step for the central difference algorithm is given by

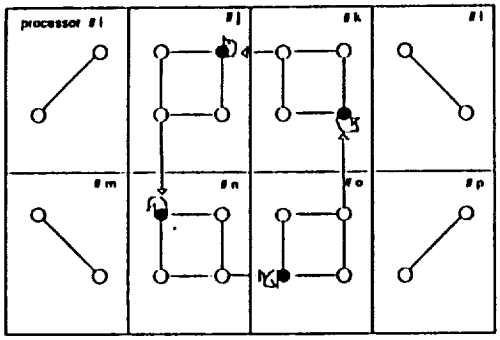
$$h \leq \frac{2}{\omega_{max}^{(e)}} \tag{6}$$

where $\omega_{max}^{(e)}$ is the maximum element frequency of the undamped dynamic problem. Belytschko has pointed out that it is in fact usually not practical to compute the maximum eigenvalues of the element directly, for this would increase the cost of computation considerably [20]. Instead, formulas for upper bounds on $\omega_{max}^{(e)}$ have been recommended. However, on massively parallel processors such as the CM_2, the parallelism inherent in the computation of $\omega_{max}^{(e)}$ is such that this frequency is obtained at the cost of the frequency of one single element.

The interprocessor communication mechanism for a mesh with more than one type of element is illustrated in figure 18. For the example shown, the 4-node elements are activated first. They communicate in four steps, one node at a time. Next, the 4-node elements are de-activated and the truss elements are selected. These communicate in two steps. As explained in Section II.2, the serialization between different types of elements is due to the SIMD nature of the CM_2.







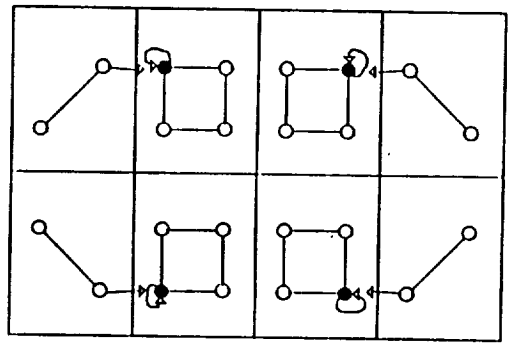


FIG. 18. Interprocessor Communication For a Hybrid Patch

VII. EXAMPLES

In this section, we apply our approach to massively parallel finite element explicit computations to the solution of various transient problems on an 8K CM_2 with Weitek accelerators. We analyze performance results in detail. We assess the efficiency of our decomposition/mapping strategy at reducing communication time. We highlight the impact on machine performance of variations in mesh topology, finite element modeling, and problem nonlinearities. We also report on the performance of the Data Vault system for problems that are I/O bound.

For each example, two simulations were carried out. The first one assumed a linear elastic material. In the second simulation, the material was assumed to have an elastoplastic behavior governed by a Von Mises yield condition.

VII.1 E1: Transient Response of a Cracked Aluminium Plate

The quarter of a mesh in figure 19 was generated to study the dynamic response of a cracked aluminum plate under a uniform time varying loading. The full mesh contained a total of 4008 plane stress elements and 4073 nodes. Mesh irregularities were induced by transition zones. The *NEWS grid* could not be used.

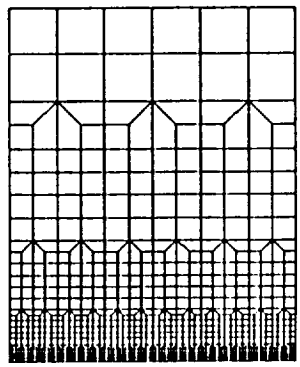


FIG. 19. A Quarter of a Mesh for a Cracked Plate

VII.2 E2: Wave Propagation in a Three-Dimensional Bar

The second example considered was the impact of a metallic ball on an unsupported glassy bar. The bar was discretized using 8160 brick elements (fig. 20). The finite element mesh contained 13500 nodes and 40500 degrees of freedom. Given the regularity of the discretization, the *NEWS grid* was used for interprocessor communication. This example was also re-run using the router for performance comparison.

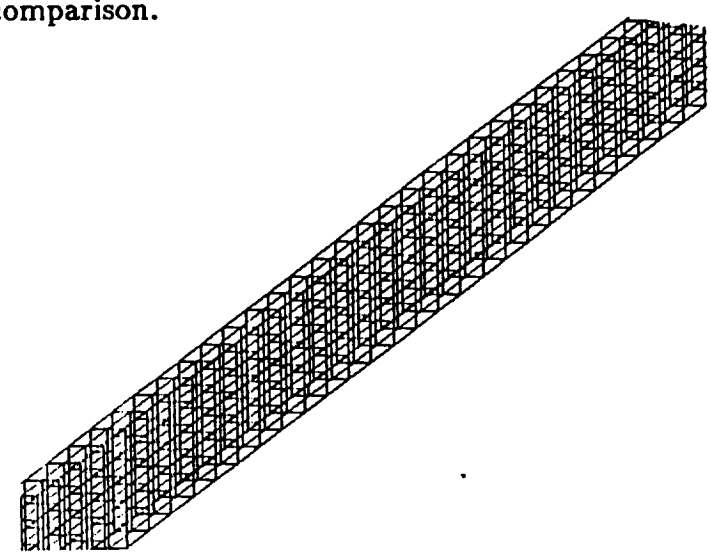


FIG. 20. Finite Element Discretization of a Glassy Bar

VII.3 E3: Shuttle Docking Induced Vibrations in a Space Station

This dynamic analysis was carried out to investigate the vibrations of a space station model assembled from 5-meter erectable struts. These vibrations were assumed to be induced by a shuttle docking. The finite element model (fig. 21) comprised 7584 three-dimensional truss elements and 2304 nodes. It was generated by aligning identical cells along various axes. However, each cell by itself was irregular (fig. 22) and did not allow the use of the *NEWS grid*.

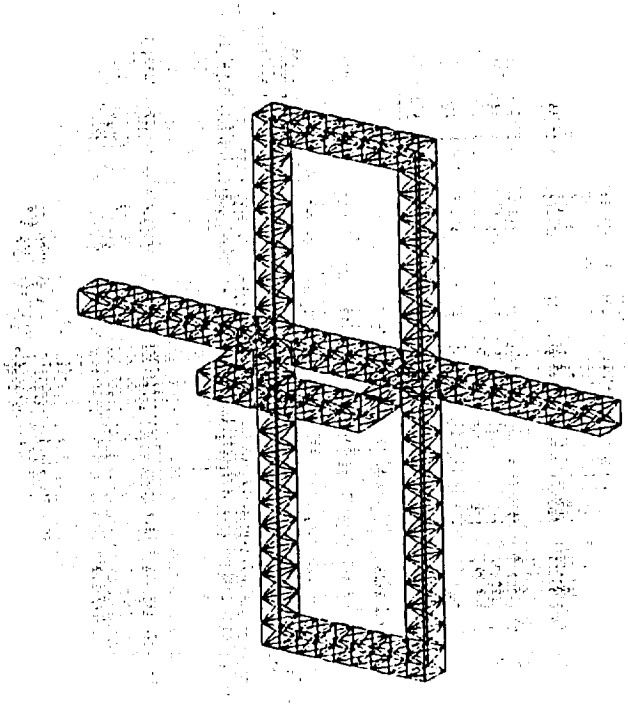


FIG. 21. A Space Station Model

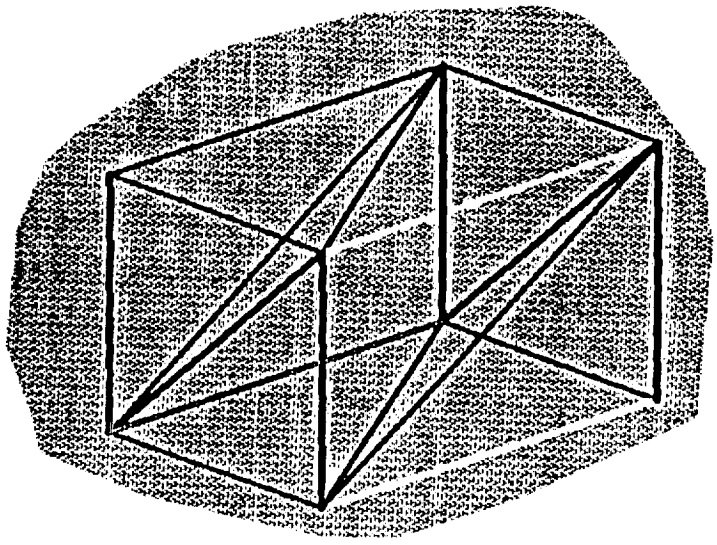


FIG. 22. Irregular Cell

VII.4 E4: Three-Dimensional Glassy Bar on an Elastic Foundation

The wave propagation example problem $E2$ was repeated with different boundary conditions. The glassy bar was assumed to be supported by a layer of foam. The mesh was comprised of a total of 8164 elements (which is very close to the number of elements in the former mesh), of which 1636 truss elements were used to model an elastic foundation.

VII.5 Performance Results and Analysis

All segments of code were written exclusively in C*. Floating-point arithmetic was performed in single precision (32 bit words). Measured performance results are gathered in tables 2, 3, 4, 5 and 6. Only example $E2$ could make use of the *NEWS grid.* However, all timings except those given in table 6 correspond to runs where communication was carried through the *router.* Execution times are given in seconds and correspond to a sample of 2000 time integration steps and a vp ratio equal to 1.

TABLE 2. Overall Measured Performance for Various Transient Finite Element Computations

Example	Mesh Preprocessing	Data Loading in the CM-2	Equation of Motion Solving	Sustained MFLOPS
$E1$ - elatic	1.04 secs	5.47 secs	861 secs	400
$E1$ - elastoplastic	1.04 secs	5.47 secs	1033 secs	480
$E2$ - elatic	1.98 secs	31.78 secs	4139 secs	392
$E2$ - elastoplastic	1.98 secs	31.78 secs	4718 secs	440
$E3$ - elatic	1.28 secs	13.56 secs	887 secs	254
$E3$ - elastoplastic	1.28 secs	13.56 secs	896 secs	256
$E4$ - elatic	2.11 secs	33.00 secs	4770 secs	340
$E4$ - elastoplastic	2.11 secs	33.00 secs	5440 secs	386

The mesh preprocessing phase corresponds to the decomposition of the finite element mesh as explained in Section V. It also includes the setup of the finite element parallel data structure, which is then distributed across the processors. Both of these phases are shown to require relatively very little computer time. It

can also be observed that in the worst case, the nonlinear computations consume only about 15% additional time. This is due to the explicit nature of the radial return mapping algorithm that was used. Because of "what you see is what you get", the reported mflop rates should be compared to those measured in Section III and not to the theoretical peak performance of the machine. It should also be noted that our C* code still leaves room for further optimizations.

TABLE 3. Data Vault System Performance

Example	Solving Equation of Motion	Unloading Results on Front End	Unloading Results on Data Vault
$E1$	861 secs	5340 secs	3.81 secs
$E2$	4139 secs	16400 secs	12.61 secs
$E3$	887 secs	9500 secs	7.04 secs

For examples $E1$, $E2$, and $E3$, the computed displacements, strains and stresses were archived on secondary storage after each time integration step. Two solutions were compared. In the first case, these results were brought back to the front end and stored in appropriate disk files. For that case, the measurements given in table 3 demonstrate that the amount of involved I/O dominated the simulation total time. In the second case, the results were transferred in parallel directly to a Data Vault System. The speed-up provided by the Data Vault is shown to be of the order of 1400! This parallel I/O capability is what was most lacking on earlier hypercubes [18].

TABLE 4. Computation vs. Communication

Example	Solving Equation of Motion	Computation Time	Communication Time
$E1$	861 secs	460 secs	401 secs
$E2$	4139 secs	1959 secs	2180 secs
$E3$	887 secs	260 secs	627 secs
$E4$	4770 secs	2340 secs	2430 secs

If T_{cp} and T_{cm} are respectively the computation parallel time and the communication parallel time, and N_p is the number of available processors on a given parallel machine, the achieved efficiency (speed-up per processor) can be expressed as:

$$EFF = \frac{1}{N_p} \frac{N_p T_{cp}}{T_{cp} + T_{cm}} = \frac{1}{1 + \frac{T_{cm}}{T_{cp}}}$$

The results given in table 4 indicate that efficiencies of 53%, 47%, 29% and 49% are achieved respectively for examples $E1$, $E2$, $E3$ and $E4$. If one refers to the performance results of Section III, it can be seen that the sustained mflop rates reported in table 2 are consistent with these efficiencies. At the first glance, these efficiency results appear to be very pessimistic. However, they are well above the 10% often obtained on current vector supercomputers [21]. The reader can observe that the timing results for example $E4$ are very close to the cumulative timings of examples $E2$ and $E3$, which illustrates the impact of the SIMD nature of the CM-2 on the MIMD nature of finite element computations. It should also be noted that while the communication time is fixed for a given mesh, the computation time increases with the complexity of the analysis. Thus, highly nonlinear formulations which include large deformations are expected to yield higher efficiencies than those deduced from table 4.

At this point, we give further details regarding interprocessor communication in the context of finite element explicit computations. As outlined in Section V, the finite elements of a mesh exchange their local contributions one node at a time. For a given finite element, this information exchange procedure is organized around two nested loops. The outer loop is carried over the nodes that are connected to this element. The inner loop is carried over the neighboring elements that are attached to each local node. Using a C notation, this is written as:

```
for (node = 1; node ≤ my_nodes; node++)   (7)
    {
      start = pointer[node]; stop = pointer[node + 1] - 1;
      for (position = start; position ≤ stop; position++)   (8)
        {
          neighbor = proc_att_to_node[position];
          exchange(variable,myself,neighbor);
        }
    }
```

where *my_nodes* is the total number of nodes that are connected to a given finite element and *proc_att_to_node* is the array containing the identification of the neighboring elements. Clearly, these variables are element dependent. The total number of communications to be performed by one processor is determined by the product $P_{cm}^{(e)} = d * (pointer[my\_nodes + 1] - 1)$ which is both element and mesh dependent. The CM_2 being an SIMD machine, the communication time is determined by $max_e\{P_{cm}^{(e)}\}$. For a regular mesh composed of three-dimensional truss elements ($d = 3$) or 4-node plane elements ($d = 2$), every node is attached to 4 elements, so that 24 communication instructions per time integration step are required for the truss element and 32 for the 4-node plane element. However, table 4 indicates that the space station example exhibits a longer communication time than the aluminum plate problem. The reason is that in the mesh of example $E3$, some truss elements are connected to 12 other elements. Because of the SIMD nature of the CM_2, the element with the highest degree of connectivity determines the communication time. For a regular mesh with 8-node solid elements ($d = 3$) each time integration step is followed by 192 communication steps, since each node can be attached up to eight different elements. This is reflected in table 4 where example $E2$ is shown to possess by far the longest communication time (2180 secs). In summary, the amount of communication involved in finite element explicit computations on the CM_2 is determined by the element topology and order, and the mesh irregularities. Because only d nodal information are exchanged at a time among the CM_2 processors, three-dimensional and high order

elements substantially increase the communication time. Mesh irregularities also adversely affect the amount of communication because of the SIMD nature of the CM_2. It is interesting to note that elements which transmit physical information across edges and faces such as those proposed by De Veubeke, [22] would require much less communication than traditional elements. These elements should be revisited for computations on massively parallel processors such as the CM_2.

An in-depth investigation of the communication phase was carried out. It was found that most of the communication time was elapsed in the header of loop (8). This loop header involves the quantities *start* and *stop* which differ from one processor to another in the presence of mesh irregularities and different element types. Consequently, the front end computer has to process and manage several different loops rather than a unique one, which is not very efficient on an SIMD machine. The time associated with the headers of loops (7) and (8) is referred to as software overhead in table 5. The true time that is elapsed in effective communication among the processors is shown to be only a fraction of the overall communication time (see table 5).

TABLE 5. True Communication Time

Example	Computation Time	Effective Coomunication Time	Software Overhead
$E1$	460 secs	81 secs	320 secs
$E2$	1959 secs	1380 secs	1280 secs
$E3$	260 secs	146 secs	481 secs

Because it was designed to handle arbitrary meshes, our C* code did not make use of the *NEWS grid* package. However, a special module that incorporated calls to the *NEWS grid* was written specifically for the regular mesh of example $E2$. Execution times for this example using both the *NEWS grid* and the *router* are shown in table 6. Clearly, a high price is paid for the handling of eventual mesh irregularities.

However, the irregular pattern of communication is fixed in time. Thus, a considerable improvement can be achieved if this pattern is evaluated at the first time step, then somehow stored in the CM_2 for use during subsequent time steps. We believe that this is an issue that massively parallel computer architects should investigate.

TABLE 6. Router vs. NEWS Grid

Example	Computation Time	Communication Time Using the *NEWS grid*	Communication Time Using the Router
$E2$	4139 secs	560 secs	2660 secs

In order to assess the performance of the decomposer/mapper module, examples $E1$, $E2$ and $E3$ were re-run with the naive shifted identity mapping (element i in processor $i-1$). Figure 23 demonstrates that the true communication time can be reduced by as much as 60 %. Unfortunately, the total execution time is reduced only between 10% and 17% because of the communication software overhead associated with mesh irregularities.

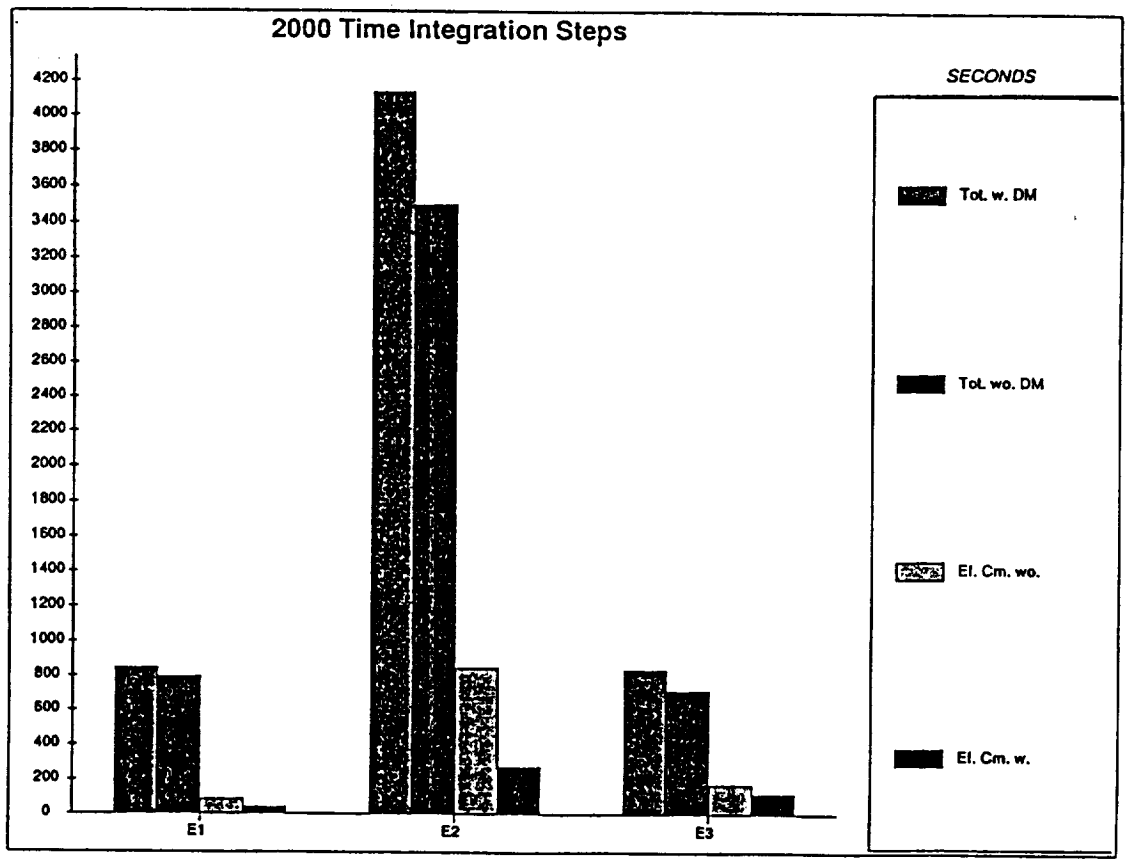


FIG. 23. The Decomposer/Mapper Performance

VIII. CONCLUDING REMARKS

We have reported herein on our experience in performing transient finite element computations on the CM_2. We have presented the architectural features of this parallel processor and discussed their impact on finite element computational strategies. In particular, those features which distinguish the CM_2 from earlier hypercubes have been emphasized. These include the virtual processor concept and the fast parallel I/O capabilities. The processor memory size of 64 Kbits has been shown to penalize high order elements. We have also described and discussed a domain decomposition strategy and a mapping algorithm which are suitable for massively parallel processors such as the Connection Machine. The main idea behind the decomposition technique is the minimization of both the amount of wire contention within a chip, and the amount of communications between different chips. A given finite element mesh is partitioned into 16-element

subdomains which correspond to the 16-processor chips of the Connection Machine. This partitioning is carried out in a way that minimizes the number of nodes at the interface between the subdomains. As a result, only those processors which are mapped onto finite elements at the periphery of a subdomain communicate with processors packaged on different chips. Moreover, this partitioning is such that the connectivity bandwidth of the resulting subdomains is large enough to allow an efficient use of the interchip wires. The mapping algorithm attempts at reducing the distance information has to travel throug the communication network. In essence, it searches iteratively for an optimal mapping through a two-step minimization of the communication costs associated with a candidate mapping. Various issues related to the single instruction multiple data stream nature of the CM_2 and pertinent to computational mechanics have been addressed. Measured performance results for realistic two and three dimensional transient problems have been reported. Three-dimensional and high order elements have been shown to induce longer communication times. Mesh irregularities have been shown to slow down the computation speed in many ways. The Data Vault has been demonstrated to be very effective at reducing the I/O time.

Now, we briefly highlight some additional implementational and theoretical issues that we hope will materially advance the application ranges of finite element computations on this highly parallel processor.

Virtual Processor Ratio vs. Substructuring

In this work, we have assigned when possible more than one finite element to a single processor using the virtual processor feature of the CM_2. However, another way to obtain the same result is to assign a substructure to an individual processor (Farhat, Wilson and Powell, [15] and Fox et al., [16]). From a numerical point of view, both approaches are equivalent. However, these two distinct approaches differ in their implementations and may perform differently. The substructure approach requires each processor to work with both external and internal data structures. The set of external data structures stores information about substructure interconnections. These are similar to the ones described in this paper. The set of internal data structures stores the connectivity table of the elements within a substructure. The computations within each substructure are carried out by looping over the elements of that substructure. The advantage of this approach is a saving in storage since the substructure internal nodes are uniquely defined, and a faster computation of the results associated with these nodes. Moreover, the global results at the internal nodes can be accumulated without any explicit call to a message-passing function. The global quantities at the boundary nodes are accumulated using the router and the external data

structures. However, the substructuring approach requires that the sequencer broadcast the same instruction several times, once for each element of the substructure, which increases the overall wall clock execution time. Moreover, this approach does not allow the Weitek chip to pipeline the computations over the elements of the substructure.

On the other hand, the virtual processor approach requires that each element communicate explicitly with its neighbors, even if these are assigned to the same processor. Of course, this communication is virtual since it is within the processor itself and generates minimal additional overhead. On the positive side, the virtual processor approach utilizes only one type of data structure and exploits the pipelining capabilities of the Weitek chip. The latter feature significantly enhances overall performance, as demonstrated in Section III. Consequently, we advocate the use of the virtual processor ratio rather than the substructuring technique, especially if the processor memory size is to be increased in the future.

Implicit Algorithms and the CM_2

In this report, no attempt has been made to design a novel parallel algorithm for the solution of the differential equation of motion. We have selected the central difference algorithm because of its inherent parallelism, which allowed us to focus on implementational issues and to fully explore the multiprocessig capabilities of the CM_2. Our experience suggests that a whole class of explicit and semi-implicit dynamic and static algorithms can be implemented on the CM_2 in a very similar way. Among others, we cite the EBE algorithms (Hughes et al., [23]), the EBE preconditioners (Hughes, Ferencz and Hallquist, [24]), and the Jacobi preconditioned conjugate gradient algorithm (Golub and Van Loan, [25]). However, the solution of some static and transient problems may necessitate the use of an implicit algorithm, which usually implies the solution of a set of simultaneous banded equations. If the global symmetric stiffness matrix $\mathbf{K}$ is banded, with semi-bandwidth b, then it is well known (see for example Ortega and Voigt, [26]) that Gaussian elimination methods for solving $\mathbf{Kd} = \mathbf{F}$ allow at each step on the order of $\frac{b^2}{2}$ pairs of (+,x) to be processed concurrently, but require significant communication because the b entries of the pivot column must be made available to all other processors. Several parallel algorithms based on these elimination methods were designed for finite element applications and were implemented on ealier hypercubes (see for example, Farhat and Wilson [27] and Utku, Salama and Melosh, [28]). Typically, a processor was assigned to a set of matrix columns. Results from our previous experience with the early version of Intel's iPSC suggest that direct solvers are feasible on hypercubes only when the number of available processors, N_p, is much smaller than the bandwidth b

of the given finite element problem, so that communications do not dominate computations. On the iPSC-1, a message that was sent from one extreme corner of a 5-dimensional cube to the other would result in an elapsed time 475 times longer than the time to perform a floating point multiplication (see Rudell, [29]). However, on a 10-dimensional subcube of the CM_2 we have measured the ratio of a broadcast to a floating point computation to be only about 2.87. This observation suggests that for problems with $b > 360$, a processor could be mapped onto a few matrix entries and a parallel direct solver could be feasible on the CM_2. For problems with smaller bandwidth, direct solvers which operate on more than one pivot at a time (Alaghband and Jordan, [30]; Peters, [31]) should also be investigated for implementation on massively parallel processors.

There is an additional issue which has to be examined before attempting to solve finite element equations on the CM_2 with a parallel direct solver. This issue is related to the balance on massively parallel processors between the number of available processors, N_p, and the processor memory size. Let M^n denote a two-dimensional regular n by n finite element mesh, where n is the number of elements along one side. If d is the number of degrees of freedom at a given node, the semi-bandwidth of M^n is $b = d(n+3)$ and the total number of mathematical unknowns is $N = d(n+1)^2$. For this mesh, the storage cost of $\mathbf{K}$ amounts to $Nb = d^2(n+3)(n+1)^2$ words. The total amount of storage available on the CM_2 is $S = N_p * m_p$, where N_p is the number of available processors and $m_p = 8$ Kbytes is the current size of the processor memory. Let $NE = n^2_{max}$ be the maximum number of elements for which M^n has a banded stiffness matrix that can be factored in-core on the CM_2. Table 7 below gives the values of NE for different values of d and for the case of a fully configured Connection Machine ($N_p = 65536$). Values of NE are shown for both single precision (32 bit words) and double precision (64 bit words) floating-point arithmetic.

TABLE 7. Number of Allowable Elements vs. DOF/Node for the Two-Dimensional Case

$N_p = 65536$		d=2	d=3	d=4	d=5	d=6
Single Precision	NE	102400	59536	40401	29929	23409
Double Precision	NE	64009	37249	25281	18769	14884

Clearly, except for the case where $d = 2$ and floating-point arithmetic is done in single precision, NE is smaller than N_p. Similarly, the case where M^n is an n by n by n three-dimensional regular mesh is assessed in table 8 below for various values of d.

TABLE 8. Number of Allowable Elements vs. DOF/Node for the Three-Dimensional Case

$N_p = 65536$		d=2	d=3	d=4	d=5	d=6
Single Precision	NE	29791	19683	13824	10648	8000
Double Precision	NE	19683	12167	9261	6859	4913

For this case, NE is much smaller than N_p, even for $d = 2$ and for single precision floating-point arithmetic. For $d = 6$ (some shell elements), only 8000 elements (4000 elements) can be included in M^n when computations are carried out using single precision (doubl precision) floating-point arithmetic.

It is noted that the eventual solution of a system of equations is only one phase of several finite element computational sequences. In linear three-dimensional analysis, this phase dominates the computer execution time. However, in the nonlinear analysis of flexible space structures most of the computational time is usually spent in modules that perform element level computations [32]. These include the evaluation of generalized nodal internal forces and/or elemental stiffness matrices. Consider now a mesh M^n where the number of elements NE is chosen so that the upper part of the banded stiffness matrix K fills the N_p processor memories completely. The preceding complexity analysis demonstrates that the balance on the CM_2 between the number of processors and the memory size of each processor is such that NE is much smaller than N_p. Hence, if a direct algorithm is used to solve a finite element system of equations, the N_p processors will be active during the solution phase, but $N_p - E$ processors will remain idle during the rest of the phases which involve element level computations. Consequently, an in-core direct solution strategy would not efficiently utilize the computational power of the CM_2 in a highly nonlinear finite element analysis.

ACKNOWLEDGMENTS

The authors wish to acknowledge the support by the Naval Research Laboratory (NRL) under Grant DOD N00014-87K-2018, with Dr. Hank Dardy and Dr. Louise Schuetz as technical monitors. The first author also acknowledges the support by the National Science Foundation under grant ASC-8717773. Both CM_2s at NRL and at CAPP were utilized to develop this work. The parallel I/O experiments were done using the Data Vault System at NRL. We also thank Drs. Carlos Felippa and Eddy Pramono for their front end fast I/O software, which was supported under the Computational Mechanics Initiative (CSM), grant NAG-1-756 from teh NASA Langley Research Center. Special thanks to Bob Whaley (Thinking Machines Corporation and NRL), Eric Hoffman (NRL) and Roldan Pozo (CAPP).

REFERENCES

[1] C. Farhat and E. Wilson, "A New Finite Element Concurrent Computer Program Architecture", *International Journal for Numerical Methods in Engineering*, Vol. 24, No. 9, (1987) pp. 1771-1792.

[2] G. A. Lyzenga, A. Raefsky, and B. H. Hager, "Finite Elements and the Method of Conjugate Gradient on Concurrent Processors", *CalTech/JPL Rept. C3P-119*, California Institute of Technology, Pasadena CA, 1984.

[3] T. Belytschko and N. Gilbersten, "Concurrent and Vectorized Mixed Time, Explicit Nonlinear Structural Dynamics Algorithms", *Parallel Computations and Their Impact on Mechanics*, ed. by A. K. Noor, American Society of Mechanical Engineers, New York, (1987) pp. 279-290.

[4] C. Farhat and L. Crivelli, "A General Approach to Nonlinear FE Computations on Shared Memory Multiprocessors", *Computer Methods in Applied Mechanics and Engineering*, (in press).

[5] M. Benten, C. Farhat and H. Jordan, "The Force for Efficient Multitasking on the CRAY Series of Supermultiprocessors", *Proceedings of the Fourth International Symposium on Science and Engineering on CRAY Supercomputers*, Minneapolis, Minnesota, Oct. 12-14, (1988) pp. 389-406.

[6] D. W. White and J. F. Abel, "Bibliography on Finite Elements and Supercomputing", *Communications in Applied Numerical Methods*, Vol. 4, No. 2, (1988) pp. 279-294.

[7] A. K. Noor, "Parallel Processing in Finite Element Structural Analysis", *Parallel Computations and Their Impact on Mechanics*, ed. by A. K. Noor, American Society of Mechanical Engineers, New York, (1987) pp. 253-277.

[8] C. Farhat, "Parallel Computational Strategies for Large Space and Aerospace Flexible Structures: Algorithms, Implementations and Performance", *Proceedings Supercomputing in Engineering Structures*, IBM Europe Institute 1988, July 11-15, 1988, Oberlech, Austria.

[9] O. McBryan, "New Architectures: Performance Highlights and New Algorithms", *Parallel Computing*, Vol. 7, No. 3, (1988) pp. 477-499.

[10] J. L. Gustafson, G. R. Montry, and R. E. Benner, "Development of Parallel Methods for a 1024-Processor Hypercube", *SIAM Journal on Scientific and Statistical Computing*, Vol. 9, No. 4, (1988) pp. 609-638.

[11] R. D. Krieg, "Unconditional Stability in Numerical Integration Methods", *Journal of Applied Mechanics*, Vol. 40, (1973) pp. 417-521.

[12] K. C. Park, "Practical Aspects of Numerical Time Integration", *Computers & Structures*, Vol. 7, (1977) pp. 343-353.

[13] W. Daniel Hillis, "The Connection Machine", *MIT Press*, Cambridge, Mass, 1987.

[14] Thinking Machines Corporation, "Connection Machine Model CM_2 Technical Summary", *Technical Report Series*, HA87-4.

[15] C. Farhat, E. Wilson and G. Powell, "Solution of Finite Element Systems on Concurrent Processing Computers", *Engineering With Computers*, Vol. 2, No. 3, (1987) pp. 157-165.

[16] Fox et. al, "Solving Problems on Concurrent Processors", *Prentice Hall*, N. J., 1988.

[17] C. Farhat, "A Simple and Efficient Automatic FEM Domain Decomposer", *Computers & Structures*, Vol. 28, No. 5, (1988) pp. 579-602.

[18] J. G. Malone, "Automated Mesh Decomposition and Concurrent Finite Element Analysis for Hypercube Multiprocessors Computers", *Comp. Meth. Appl. Mech. Eng.*, Vol. 70, No. 1, (1988) pp. 27-58.

[19] C. Farhat, "On the Mapping of Massively Parallel Processors Onto Finite Element Graphs", *Computers & Structures*, (in press).

[20] T. Belytschko, "Overview of Semidiscretization", *Computational Methods for Transient Analysis*, ed. by T. Belytschko and T. J. R. Hughes, North-Holland, (1983) pp. 1-63.

[21] Kuck *et.al*, "The Effects of Program Restructuring, Algorithm Change, and Architecture Choice on Program Performance", *IEEE*, (1984) pp. 129-138.

[22] "B. M. Fraejis De Veubeke Memorial Volume of Selected Papers", ed. by M. Geradin, Sijthoff & Noordhoff, 1980.

[23] T. J. R. Hughes *et al.*, "Element-by Element Implicit Algorithms For Heat Conduction", *J. Eng. Mech.*, Vol. 109, No. 2, (1983) pp. 576-585.

[24] T. J. R. Hughes, R. M. Ferencz and J. O. Hallquist, "Large-scale Vectorized Implicit Calculations in Solid Mechanics on a Cray X-MP/48 Utilizing EBE Preconditioned Conjugate Gradients", *Computer Methods in Applied Mechanics and Engineering*, Vol. 61, No. 2, (1987) pp. 215-248.

[25] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The John Hopkins University Press, 1983.

[26] J. M. Ortega and R. G. Voigt, "Solution of Partial Differential Equations on Vector and Parallel Computers", *SIAM Review*, Vol. 27, No. 2, (1985) pp. 149-240.

[27] C. Farhat and E. Wilson, "A Parallel Active Column Equation Solver", *Computers & Structures*, Vol. 28, No. 4, (1988) pp. 289-304.

[28] S. Utku, M. Salama and R. Melosh, "Concurrent Factorization of Positive Definite Banded Hermitian Matrices", *Int. J. Num. Meth. Eng*, Vol. 23, (1986) pp. 2137-2152.

[29] R. Rudell, "Parallel Processing Efficiency on the Hypercube", *CS252 Project Report*, The University of California at Berkeley, 1985.

[30] G. Alaghband and H. F. Jordan, "Multiprocessor Sparse L/U Decomposition with Controlled Fill-in", *ICASE Report No. 85-48*, NASA Langley Research Center, Hampton, Virginia 23665, 1985.

[31] F. J. Peters, "Parallel Pivoting Algorithms for Sparse Symmetric Matrices", *Parallel Computing*, Vol. 1, (1984) pp. 99-110.

[32] N. F. Knight et al., "CSM Testbed Development and Large Scale Structural Applications", *Proceedings of the 4th International Symposium*, Minneapolis, Minnesota, (1988), pp. 359-388.

[33] B. Nour-Omid and K. C. Park, "Solving Structural Mechanics Problems on the CALTECH Hypercube Machine", *Comp. Meth. Appl. Mech. Eng.*, Vol. 61, No. 2, (1987) pp. 161-176.

WHICH PARALLEL FINITE ELEMENT ALGORITHM FOR WHICH ARCHITECTURE AND WHICH PROBLEM?

C. Farhat
Department of Aerospace Engineering Sciences and Center for Space Structures and Controls
University of Colorado at Boulder
Boulder, Colorado

ABSTRACT

The various forms of parallel numerical algorithms that speed up finite element computations are as numerous as the number of researchers working on the problem. In this paper, we review some of these parallel computational strategies and assess their adequacy for a given architecture and a given problem. We also report on the performance of both extreme parallel hardware technologies on real-life structural problems.

I INTRODUCTION

The realistic simulation of the nonlinear dynamics of complex structural systems remains beyond the feasible range of traditional computers. It has been the author's experience that the simulation of the transient response of a space station model with 100,000 degrees of freedom to various loading configurations consumes over 10 CPU hours on a CRAY-2 supercomputer and that the simulation of the deployment of a space structure is even more computationally demanding, especially if the control/structure interaction problem is to be included. The aeroelastic response of a detailed wing-body configuration using a potential flow theory requires about 5 CPU hours using the same supercomputer. In order to establish the transonic flutter boundary for a given set of aeroelastic parameters, about 30 aeroelastic response analyses are required, which brings the total CPU time to 6 days. If the full Navier-Stokes equations are to be solved, it is estimated that the CPU time increases by two orders of magnitude. It is also clear that large amounts of data can be generated in a large-scale transient structural analysis or a large-scale computational fluid dynamic solution. This raw data has to be interpreted, in real-time if possible, in order to be understood.

Clearly, the true potential for execution improvement lies in massively parallel and/or parallel/vector supercomputing. The commercial supercomputer manufacturers of the last decade have extended their products into configurations that use a few vector processors coupled around a massive shared memory (CRAY-2, CRAY X-MP, CRAY Y-MP). Supercomputers with a larger number of vector processors are also under development (CRAY3). Concurrent multiprocessors with much finer granularity and a wide range of interconnection strategies are now appearing. Recently, massively parallel computers such as the CONNECTION MACHINE have demonstrated their potential to be the fastest supercomputers, a trend that may accelerate in the future (McBryan [1]). The advent of advanced frame buffers and high performance workstations such as Ardent's TITAN now makes real-time visualization possible.

Moving engineering applications to concurrent processors faces significant obstacles that will have to be resolved as such machines become more and more available. The obstacles center on algorithms, methods, languages, and education. In this paper, we address some of these issues in the context of finite element computations.

The various forms of parallel numerical algorithms that speed up finite element computations are as numerous as the number of researchers working on the problem. Extensive lists of references on this topic may be found in the surveys of Noor [2], White and Abel [3], and Ortega, Voigt and Romine [4]. Throughout this paper, we discuss the adequacy of a set of parallel finite element computational strategies (mesh preprocessing, solution algorithms, I/O manipulations) for a given parallel processor and a given structural and/or mechanical problem. This leads us to the introduction of the notion of *algorithmic portability* in addition to the problem of *language portability*.

The remainder of this paper is organized as follows. In Section II, we present an overview of the present status of parallel computers that is pertinent to finite element computations. Through the examples of SIMD (Single Instruction Multiple Data), MIMD (Multiple Instruction Multiple Data), local memory and shared memory multiprocessors, we address the impact of hardware architecture on the design and implementation of parallel algorithms and parallel data structures. Section III focuses

on local memory MIMD hypercubes and Section IV on shared memory multiprocessors. Section V summarizes the author's experience with massively parallel finite element computations on the CONNECTION MACHINE. Performance results and concluding remarks are offered in Section VI and Section VII.

Because of space limitations, algorithmic details and formulas are avoided. The paper emphasizes major results and conclusions. For specific details, the reader is urged to consult the references.

II WHAT ONE MUST KNOW ABOUT PARALLEL PROCESSORS

Several parallel computers have already been marketed commercially. Rather than discuss these individually, we here focus on presenting an overview of their architecture and emphasize the impact of their hardware features on the design and implementation of parallel computational strategies for finite element simulations. A review of some of the commercially available parallel systems can be found in Babb [5], where programming examples are also provided.

Multiprocessors can be generally described by three essential elements: granularity, topology and control.

Granularity relates to the number of processors and involves the size of these processors. A fine-grain multiprocessor features a large number of usually very small and simple processors. The CONNECTION MACHINE (65,536 processors) is such a massively parallel supercomputer. NCUBE's 1024-node and iPSC's 128-node models are comparatively medium-grain machines. On the other hand, a coarse-grain multiprocessor is typically built by interconnecting a small number of large, powerful processors, — usually but not necessarily vector processors. ALLIANT FX/8 (8 processors), IBM 3090-VF (6 processors), CRAY X-MP (4 processors), CRAY-2 (4 processors) and the ETA-10 (8 processors) are examples of such multiprocessors and supermultiprocessors. Granularity directly affects the parallel computational strategy. On a coarse-grain multiprocessor, finite element computations can be parallelized at the subdomain level. On a fine-grain machine, they are best parallelized at the element and sometimes at the degree of freedom level. When designing parallel algorithms for finite element computations on coarse grained vector supermultiprocessors, one should preserve vectorization. This is because the potential speed-up due to interconnecting a few vector processors cannot compete with the speed-up due to the vector capabilities of a single processor. This matter is addressed and emphasized in Section IV.

Topology refers to the pattern in which the processors are connected and reflects how data will flow. Currently available designs include hypercube arrangement, network of busses, and banyan networks. Usually, the interconnection topology is related to the memory organization. For example, iPSC, NCUBE and the CONNECTION MACHINE are local memory multiprocessors with a hypercube topology. On these systems, a processor is assigned its own (local) memory and can only access this memory. Independent processors communicate by sending each other messages. Efficient solution of finite element simulations on these machines requires minimizing the interprocessor communication bandwidth, especially when the communication hardware/software is relatively slow. This requires the mapping of adjacent elements as much as possible onto directly connected processors, which may be no trivial problem. On the other hand, the processors on a shared memory system such as ALLIANT FX/8 are connected through a common memory bus and can access the same (global) large memory system. Adequate finite element parallel data structures are crucial for efficient computations on both shared and local memory multiprocessors. On a local memory machine, one has to introduce the concept of distributed data base and data structure. Each local memory is loaded only with the data relevant to the computational task assigned to its attached processor. For a system with thousands of processors, the total amount of available memory can be very large. Yet, it is the storage capacity of each local memory which really matters. Different finite elements require different amounts of data to be stored. For each finite element in the mesh, a material and geometrical nonlinear high order shell element may require an amount of data storage two orders of magnitude higher than a simple linear truss element. Hence, one may be able to assign one or several finite elements of a certain type to one processor but may fail in the attempt to assign one or several elements of another type to a similar processor. Also, in the case of MIMD machines such as iPSC and NCUBE, one has to ensure that the compiled subroutines can be accommodated on the local memory. Consider the case where a processor is mapped onto a submesh containing different types of elements. In this situation, one has to load into the processor's local memory all the element libraries for the types encountered in the assigned submesh. Generally, one can overcome these problems by devising an intelligent partitioning scheme and a compact data structure. Careful data structures must also be designed for shared memory multiprocessors to avoid potential serializations due to memory conflicts.

Control describes the way the work is divided up and synchronized. Of particular interest are the SIMD and MIMD machines. The CRAY-2 (4 processors) and iPSC (128 processors) are respectively a shared memory MIMD supermultiprocessor and a local memory MIMD hypercube. They can simultaneously execute multiple instructions which can operate on multiple data. The CONNECTION MACHINE is an SIMD system where a single instruction is executed at a time, — an instruction which can operate on multiple data. Typically, on an SIMD machine a single program executes on the front end and its parallel instructions are submitted to the processors. On an MIMD parallel processor separate program copies execute on separate processors.

Practically, local memory parallel processors are more difficult to program than shared memory multiprocessors. However, this does not imply that optimal performance is easily achieved on shared memory machines, especially when vector processors are interconnected. It is believed that local memory systems are easier to scale to a large number of processors. Shared memory multiprocessors are usually coarse grained because the bus to memory saturates and/or becomes prohibitively expensive above a few processors. However, machines such as Evans and Sutherlands' ES-1 and MYRIAS are considered as shared memory multiprocessors and can be configured with several hundreds of processors. Note also that on SIMD machines, one has to devise special tricks to be able to process parallel finite elements of different types, since these do not involve the same instructions and only one instruction can be executed at a time.

III FE COMPUTATIONS ON MIMD LOCAL MEMORY MULTIPROCESSORS

Several solution algorithms have been designed for static, modal and transient finite element analyses on MIMD local memory multiprocessors. Examples of these can be found in Farhat and Wilson [6] (Intel's iPSC), Lyzenga, Raefsky and Hager [7] and Nour-Omid, Raefsky and Lyzenga [8] (JPL/Caltech's MARK III). Typically, these algorithms stem from the *divide and conquer* paradigm.

Consider the finite element discretization of the mechanical joint shown in figure 1. If the complete finite element system is subdivided into N_s subdomains, each group of elements within a subdomain can be processed in parallel. The data structure for such an approach is very simple. On local memory multiprocessors, only the storage for the node geometry and element properties within the substructure need be stored within the RAM (Random Access Memory) of the processor assigned to that subdomain. In addition, concurrent formation and reduction of the mass, damping and stiffness matrices for that region require no interprocessor communication. Message passing occurs only when transfering solutions between subdomain interconnected boundaries. The latter phase often determines the efficiency of the parallel computational approach. While load balancing is an important criterion for automatically subdividing a mesh into as many submeshes as there are available processors, $N_p = N_s$, it is not sufficient by itself to determine the partitioning algorithm.

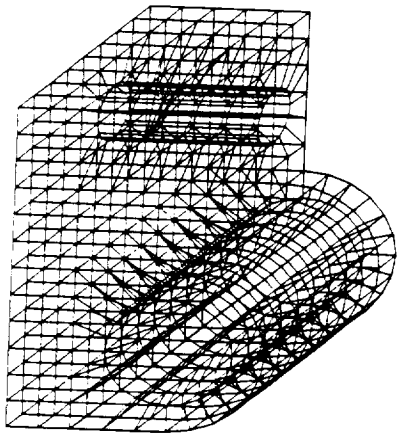


Fig. 1 Discretization of a mechanical joint

Suppose that a parallel explicit or explicit-like algorithm is to be implemented on an MIMD local memory multiprocessor. It could be, for example, an iterative solver for the linearized static problem, or a time integration explicit algorithm for the transient response analysis. Typically, these computations involve matrix-vector products Ku and inner products $u^T u$ which can be evaluated in parallel as:

$$Ku = \sum_{j=1}^{j=N_s} K_j u_j$$

$$u^T u = \sum_{j=1}^{j=N_s} u_j^T u_j$$

where K_j denotes the stiffness of the $j-th$ subdomain and u_j is the localization of the displacement vector to the $j-th$ subdomain. In this case, only neighboring subdomains need to exchange boundary information. Hence, an optimal decomposition is the one which minimizes the communication bandwidth of the problem, — that is, the subdomain connectivity. This strategy is discussed by Malone in [9]. When applied to the above problem, for $N_p = 32$, it delivers the partitioning shown in figures 2a-2b. The average and maximum communication bandwidths are 5 and 8 respectively, and the number of interface nodes is 718.

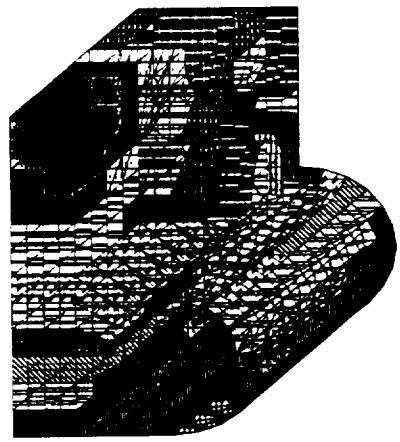


Fig. 2a Decomposition with $N_p = 32$

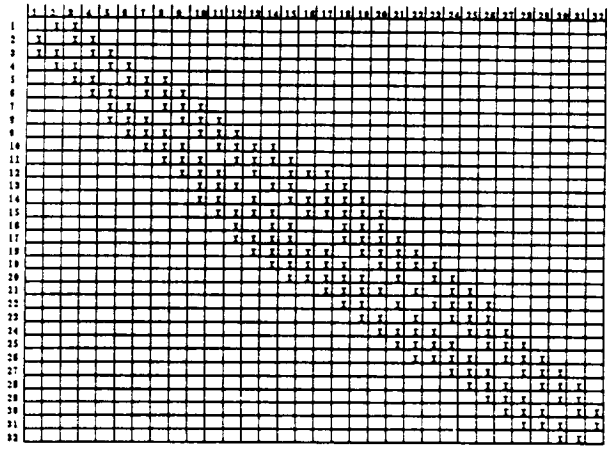


Fig. 2b Interprocessor communication pattern

Suppose now that parallel implicit static or dynamic computations are to be invoked. In this case, a higher level of parallelism is obtained by treating the interface nodes as a separate entity, and numbering the unknowns so that, for example, the stiffness matrix has the pattern shown in figure 3b. The submatrices K_{jj}, K_{II} and K_{jI} denote respectively the subdomain and interface stiffnesses, and the coupling term. Clearly, all subdomains can be processed in parallel after the interface problem

$$(K_{II} - \sum_{j=1}^{j=N_p} K_{jI}^T K_{jj}^{-1} K_{jI}) u_I = f_I - \sum_{j=1}^{j=N_p} K_{jI}^T K_{jj}^{-1} f_j \qquad (2)$$

has been solved. In equation (2), u_I and f_I are respectively the generalized displacements and forces at the interface nodes. The size of the interface problem determines the efficiency of this parallel stratagem. An optimal decomposition for this approach which minimizes the number of interface nodes is presented by Farhat in [10]. When applied to the above mechanical joint (fig. 3a), for $N_p = 32$, it delivers 356 interface nodes only.

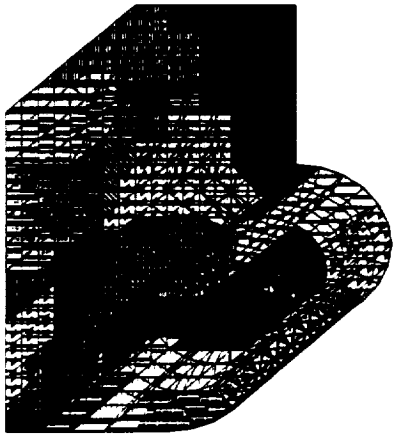


Fig. 3a Decomposition: $N_p = 32$ and interface minimization

$$\begin{bmatrix} K_{11} & 0 & 0 & 0 & 0 & K_{1I} \\ 0 & \ddots & 0 & 0 & 0 & \vdots \\ 0 & 0 & K_{jj} & 0 & 0 & K_{jI} \\ 0 & 0 & 0 & \ddots & 0 & \vdots \\ 0 & 0 & 0 & 0 & \ddots & \vdots \\ K_{1I}^T & \cdots & K_{jI}^T & \cdots & \cdots & K_{II} \end{bmatrix}$$

Fig. 3b Pattern for stiffness matrix

Equation (2) can be solved using a direct method (Farhat, Wilson and Powell [11]), or an iterative one (Farhat and Wilson [12], Nour-Omid, Raefsky and Lyzenga [8]). Let n_I^* denote the average number of interface nodes per subdomain, and d the number of degrees of freedom per node. If the interface problem is treated with a direct solver, the formation of Schur's complement $K_{II} - \sum_{j=1}^{j=N_p} K_{jI}^T K_{jj}^{-1} K_{jI}$ requires $2N_s n_I^* d$ solutions of sparse triangular systems. On the other hand, each conjugate gradient iteration involves N_s matrix-vector products of the form $K_{jI}^T K_{jj}^{-1} K_{jI} u_I^{(k)}$, which require $2N_s$ solutions of sparse triangular systems. If memory is an issue, the fill-in of (2) can be such that an iterative solver, for example the preconditioned conjugate gradient method, is recommended for the solution of the interface problem. If the coupling between subdomains is very strong, a preconditioned conjugate gradient algorithm may require more than $n_I^* d$ iterations to achieve convergence, so that a direct method becomes more advantageous.

Parallel modal and transient analyses using both approaches have been experimented on Intel's iPSC (Farhat and Wilson [13], Malone [9]).

The reader should note that when using implicit computations, the substructuring technique introduces a high level of parallelism, however sometimes at the cost of additional floating point computations. On the other hand, parallel direct solvers do not increase the computational complexity, but on local memory multiprocessors, they may suffer from interprocessor communication costs. For this reason, and because the degree of parallelism direct parallel solvers offer is limited by the mesh bandwidth, the author does not recommend their use for the solution of the entire finite element system on currently available local memory multiprocessors, especially if the number of processors is large, say $N_p \geq 128$. However, they have been successfully combined with the substructuring technique to solve the interface problem only (see Farhat, Wilson and Powell [11]).

IV FE COMPUTATIONS ON MIMD SHARED MEMORY MULTIPROCESSORS

In principle, parallel algorithms which are developed for local memory multiprocessors can be used on shared memory machines. However, a much higher performance can be achieved if the special features of these machines are fully exploited. In particular, if the multiprocessor offers a vector capability, the algorithms outlined in Section III must be revisited.

For explicit computations on a shared memory multiprocessor, the substructuring approach advocated in Section III may be also utilized. Interface data may be either duplicated in the shared memory, or treated as a *Critical Section* (see Benten, Farhat and Jordan [15]), — that is, a portion of a code where a processor needs to store into a memory location used concurrently by another processor. In the latter case, the processors are serialized when processing the interface degrees of freedom. For example, while in (1) the quantities $K_j u_j$ and $u_j^T u_j$ can be evaluated in parallel for all j, the assembly of the results at the interface nodes is recursive and requires serialization. A more efficient approach on shared memory machines is described by Farhat and Crivelli in [16]. Explicit computations are parallelized at the element level. Memory contention, and therefore *Critical Sections*, are avoided by processing the elements in an order dictated by a graph coloring algorithm [16]. Basically, the mesh is partitioned into sets of internally disjoint elements, so that vectorization and parallelization are optimized. For example, when

applied to the problem shown in figure 1, the coloring scheme creates 8 sets of internally disjoint elements. Figure 4 shows the elements in set 3 for this example. Within a set of elements, explicit computations are performed asynchronously. Synchronization points are required only between the processing of two different sets of elements (8 synchronization points in this case).

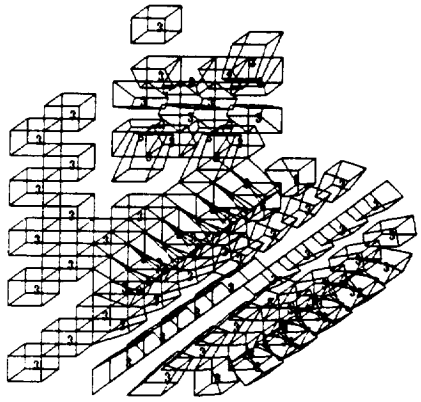


Fig. 4 Internally disjointed elements in set #3

Contrary to popular belief, implicit computations are more difficult to optimize on shared memory multiprocessors. To illustrate this fact, we consider the static solutions of the mechanical joint and of the Solid Rocket Booster (SRB) (fig. 5) problems, for a prescribed loading. Moreover, we assume that $N_p = 4$ processors are available. The subdivision of both meshes into balanced subdomains with a minimum number of interface nodes are depicted in figures 6a-6b.

Fig. 5 Solid rocket booster

The discretized mechanical joint contains 456 elements and 852 nodes. After node-renumbering, the average profile bandwidth is 168. The optimized average profile bandwidth for each subdomain is 93. Therefore, the parallel reduction of each subdomain benefits not only from a lesser number of equations to be reduced, but also from a smaller bandwidth. For this problem, the computational savings due to a lower subdomain bandwidth offs the computational requirements of the interface problem (2). that the parallel algorithm based on substructuring is faster, ev in serial mode, than a global Choleski decomposition.

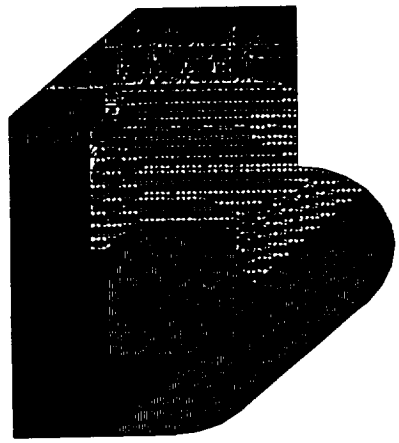


Fig. 6a Decomposition with $N_p = 4$ and interface minimization

Fig. 6b Decomposition with $N_p = 4$ and interface minimization

The discretized SRB model has 10,453 elements, 9,206 nodes and 54,870 degrees of freedom. The number of interface nodes corresponding to its subdivision into 4 subdomains is 165. After node-renumbering, the average profile bandwidth is 310. The optimized average profile bandwidth for the 4 subdomains is 365. Clearly, for this problem, reducing the 4 subdomain stiffnesses requires a little more floating point operations than reducing the global stiffness. Consequently, all the manipulations involved in the solution of the interface problem (2) are additional computations generated by the substructuring method. Fortunately, the interface problem size is only about 2% of the size of the entire problem, so that the parallel method is still feasible. However, for this problem, and especially if vector processing is available, a better performance is achieved with a parallel highly vectorized direct solver. Indeed, the effectiveness of the solution of the interface problem (2) comes from sophisticated implementations

whose details cannot be described here. For example, the algebraic manipulations involved in the evaluation of the quantity

$$K_{jI}^T K_{jj}^{-1} K_{jI} \qquad (3)$$

do not vectorize well if the sparse data structures and computational techniques described by George and Liu in [17], and advocated by the author for local memory multiprocessors [11] are used, unless special tricks are invoked. On the other hand, for the SRB problem, a parallel direct global solver such as the parallel active column solver presented by Farhat and Wilson in [18], or a parallel version of the highly vectorized variable band solver described by Poole and Overman in [19] are very efficient on a parallel/vector supercomputer (CRAY Y-MP, 8 processors). For the SRB problem, this is especially true because the bandwidth to number of processors ratio is 310/8 = 38.75.

Clearly, the above examples demonstrate that the optimal efficiency of a parallel algorithm depends on the underlying hardware architecture and on the topological characteristics of the problem to be solved.

With the advent of hardware gather-scatter on most recent vector supercomputers, significant progress has been made in implementing sparse linear equation solvers on these machines (see Lewis and Simon [20]). Recently, Aschcraft, Grimes, Lewis, Peyton and Simon [21] have used the new algorithmic concept of a supernodal sparse factorization for implementing a superfast sparse linear solver on the CRAY X-MP. The key ideas behind the high level of vectorization come from the graph theory model of the sparse elimination process which can be found in the book of George and Liu [17]. In [22], Simon, Vu, and Yang describe a parallel implementation of the supernodal sparse code which delivers a performance rate as high as 1.682 GIGAFLOPS. However, sparse solvers require a preliminary nodal re-ordering (*i.e.* minimal degree ordering) and symbolic factorization which can consume an important amount of CPU time. Therefore, they are most effective in nonlinear problems or problems with several right hand sides, where the preprocessing phase is done once.

Parallel I/O developments for finite element simulations, and performance measurements on shared memory multiprocessors can be found in Farhat, Pramono and Felippa [14].

V FE COMPUTATIONS ON A MASSIVELY PARALLEL PROCESSOR

The CONNECTION MACHINE is probably the only massively parallel processor that is now commercially available. It consists of two parts: a front end computer (VAX, SYMBOLICS, SUN), and a 64K processor hypercube (65,536 single bit processors). The front end computer provides instruction sequencing and program development and has the ability to address any location in the hypercube distributed memory. The hypercube system provides number crunching power.

Recently, Farhat, Sobh and Park [23, 24] have investigated massively parallel transient finite element explicit computations on the CONNECTION MACHINE. Preliminary results can be found in [23] and more detailed information in [24]. In general, it has been found that this highly parallel processor can outperform vector supercomputers on explicit computations, but not on implicit ones. Several features distinguish the CONNECTION MACHINE from earlier hypercubes. On the hardware side, we note the impressive number crunching power and the fast parallel I/O capabilities. On the software side, we note the virtual processor concept, which is somehow the dual of the well-known virtual memory concept. Mesh decomposition and processor-to-element mapping are the two fundamental keys for efficient massively parallel finite element computations. A given finite element mesh is partitioned into 16-element subdomains which correspond to the 16-processor chips of the CONNECTION MACHINE. This partitioning is carried out in a way that minimizes the number of nodes at the interface between the subdomains. As a result, only those processors which are mapped onto finite elements at the periphery of a subdomain communicate with processors packaged on different chips. Moreover, this partitioning is such that the connectivity bandwidth of the resulting subdomains is large enough to allow an efficient use of the 12 interchip wires. The mapping algorithm attempts at reducing the distance information has to travel through the communication network. In essence, it searches iteratively for an optimal mapping through a two-step minimization of the communication costs associated with a candidate mapping (see Farhat [25]). We summarize herein the basic conclusions reported in [23, 24]. The processor memory size of 64 Kbits penalizes high order elements. Three-dimensional and high order elements induce longer communication times. Mesh irregularities slow down the computation speed in many ways. The *Data Vault* is very effective at reducing I/O time. The *Frame Buffer* is ideal for real-time visualization. Finally, the virtual processor concept outperforms the substructuring technique on the CONNECTION MACHINE.

VI PERFORMANCE EXAMPLES

The speed-up and MFLOP rates reported in this section include all phases of the finite element analyses. A pair of (+,*) is counted as 2 flops.

To illustrate the *surgeon* approach to parallel/vector finite element computations, we report on the solution of three different problems on three different multiprocessors. First, we consider a modal analysis of the simplified space station model shown in figure 7. The finite element mesh comprises 384 nodes, 1264 beam elements and 2304 degrees of freedom. Since this is rather a small problem, we consider the use of an Intel iPSC with 16 processors and 4 Mbytes of available memory. After node-renumbering, the average profile bandwidth for this problem is 90. We select not to use a global parallel direct solver to carry out implicit computations, because it would allow only 6 columns of the stiffness matrix to be assigned to one processor, which would make interproccessor communications dominate local computations. Therefore, we select an approach based on the substructuring technique outlined in Section III. The mesh is decomposed into 16 balanced subdomains, each containing approximately 79 elements. The size of the interface problem is 672 (112 nodes). Our parallel algorithm for eigenvalue extraction and modal superposition on a hypercube

architecture is described in [13]. The number of extracted modes is 200. The performance results for this analysis on the iPSC are reported in table 1.

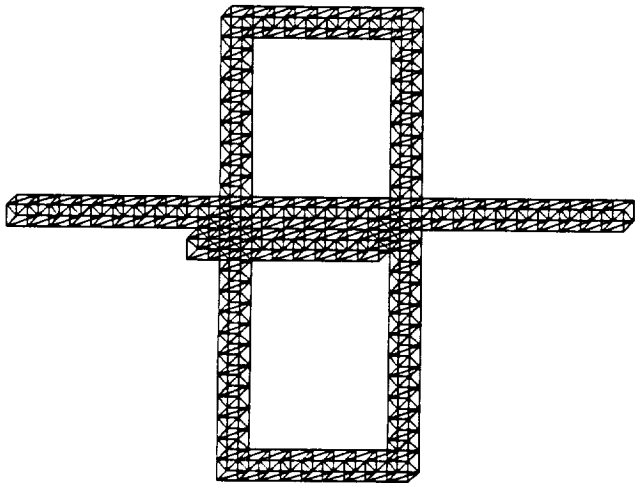


Fig. 7 Space station structural model

Table 1 Modal analysis on a 16-processor iPSC

Space station structural model
2,304 d.o.f - 200 modes

Phase	Speed-up	MFLOPS
Forming K and M	15	0.6
Factoring K	12	0.5
Generating Lanczos vectors	12	0.5
Extracting 200 frequencies	14	0.4
Computing 200 mode shapes	13	0.5

Next, we consider the transient response of a more detailed space station model to perturbations induced by shuttle docking. The finite element model for this analysis incorporates 7596 2-node beam elements, 572 4-node shell elements, 24 3-node rigid elements, 9802 nodes and 58,812 degrees of freedom (fig. 8). Given the size of this problem, we select to run it on an 8K CONNECTION MACHINE using the parallel central difference algorithm [20]. Table 2 summarizes the measured performances for computations and I/O manipulations. The latter correspond to dumping at each time step the computed displacements, velocities, accelerations, stresses and strains onto the front-end. The reported performances are scaled to the full 64K processor configuration (see [1] for justifications). For this problem, the Data Vault improves I/O by a factor of 1307!

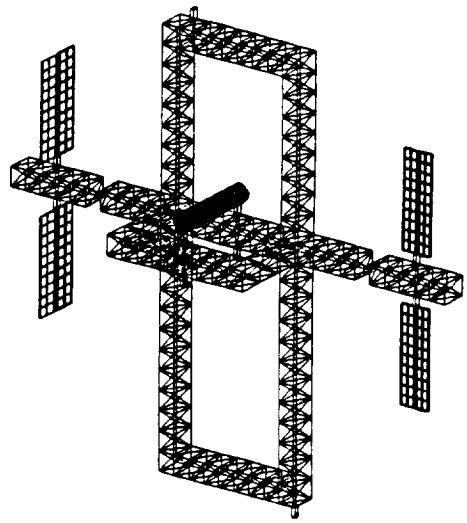


Fig. 8 Detailed space station model

Table 2 Transient analysis on the Connection Machine

Detailed space station structural model
58,812 d.o.f - 2000 time integration steps

Phase	CPU time	MFLOPS (using C^*)
Mesh decomposition	3 secs	-
Data loading in the CM2	41 secs	-
Equation of Motion Solving	4500 secs	340
Computation time	2500 secs	665
Communication time	2000 secs	-
I/O through front end	18,300 secs	-
I/O through data vault	14 secs	-

Finally, we consider the static analysis of the SRB on a CRAY Y-MP with 8 processors. Following the reasoning of Section IV, we select to perform the factorization of the stiffness matrix using a global parallel direct algorithm. For this purpose, we have developed a parallel/vector version of the highly vectorized direct solver described in [19]. The measured performances for 1, 2 and 4 processors are tabulated below (table 3). No results are available for the case $N_p = 8$ because the author could not arrange for a dedicated time on the CRAY Y-MP

The SRB problem was also solved in [22] using the supernodal sparse factorization. The corresponding results are displayed in table 4. It is interesting to note that while the sparse factorization is twice as fast as the variable band solver on a single CPU, both algorithms become comparable on 4 CPUs. Note also that for the SRB problem, it appears that the supernodal code does not parallelize well.

Table 3 Static analysis on the CRAY Y-MP

SRB structural model - 54,870 d.o.f.

Number of processors	CPU time	Speed-up	MFLOPS
1	39 secs	1	235
2	19.79 secs	1.97	464
4	10 secs	3.90	918
8	NA	NA	NA

Table 4 Static analysis on the CRAY Y-MP

SRB structural model - 54,870 d.o.f.

Supernodal Sparse Factorization

Number of processors	CPU time	Speed-up	MFLOPS
1	20.21 secs	1	231.71
2	13.12 secs	1.54	355.79
4	9.53 secs	2.12	491.45
6	8.53 secs	2.37	548.90
8	8.12 secs	2.49	578.08

CONCLUSIONS

In summary, the choice of a parallel finite element algorithm should be dictated by the multiprocessor to be used and the problem to be solved. On local memory MIMD (Multiple Instruction Multiple Data) parallel processors, the substructuring technique is recommended for both implicit and explicit computations. On shared memory multiprocessors, the decision is more difficult. If the bandwidth of the problem is small, say only 5 times the number of available processors, the substructuring technique is still recommended, unless the bandwidth of each subdomain is not lower than that of the global problem. Otherwise, a global parallel solver is advocated. In the case where vector processing is available, special data structures and computational orderings must be used in order to fully exploit the vectorization capabilities. The analyst must realize that the potential speed-up due to interconnecting a few vector processors cannot compete with the speed-up due to the vector capabilities of a single processor. Finally, massively parallel processors are just emerging. The CONNECTION MACHINE can outperform vector supercomputers when explicit computations are utilized.

While most portability problems on serial machines are due to subtleties in compilers and high-level languages, parallel computers will face the additional burden of *algorithmic portability*. Currently, the only portable parallel code is the one which is driven by an analyzer which takes for input the problem to be solved and the multiprocessor to be used, and outputs the switch for the right parallel algorithm to be invoked.

ACKNOWLEDGMENTS

The author would like to thank Dr. N. Knight (NASA Langley) for providing the SRB finite element model and Ms. A. Overman (NASA Langley) and Dr. E. Poole (Awesome Computing and NASA Langley) for communicating their valuable work [19], and Dr. H. Simon (Boeing and NASA Ames) for his enlightening discussion on sparse linear solvers. He also wishes to acknowledge partial support by NSF under Grant 87-17773, and partial support by NASA Langley under Grant NAG1-756.

REFERENCES

[1] O. McBryan, "New Architectures: Performance Highlights and New Algorithms", *Parallel Computing*, Vol. 7, No. 3, (1988) pp. 477-499.

[2] A. K. Noor, "Parallel Processing in Finite Element Structural Analysis", *Parallel Computations and Their Impact on Mechanics*, ed. by A. K. Noor, American Society of Mechanical Engineers, New York, (1987) pp. 253-277.

[3] D. W. White and J. F. Abel, "Bibliography on Finite Elements and Supercomputing", *Communications in Applied Numerical Methods*, Vol. 4, No. 2, (1988) pp. 279-294.

[4] J. Ortega, R. Voigt and C. Romine, "A Bibliography on Parallel and Vector Numerical Algorithms", *NASA Contractor Report 181764, ICASE Interim Report 6*, 1988.

[5] R. G. Babb II, (Ed.), "Programming Parallel Processors", Addison-Wesley Publishing, Inc., 1988.

[6] C. Farhat and E. Wilson, "A New Finite Element Concurrent Computer Program Architecture", *Int. J. Num. Meth. Eng.*, Vol. 24, (1987) pp. 1771-1792.

[7] G. A. Lyzenga, A. Raefsky, and B. H. Hager, "Finite Elements and the Method of Conjugate Gradient on Concurrent Processors", *CalTech/JPL Rept. C3P-119*, California Institute of Technology, Pasadena CA, 1984.

[8] B. Nour-Omid, A. Raefsky and G. Lyzenga, "Solving Finite Element Equations on Concurrent Computers", *Parallel Computations and Their Impact on Mechanics*, ed. by A. K. Noor, American Society of Mechanical Engineers, New York, (1987) pp. 209-228.

[9] J. G. Malone, "Automated Mesh Decomposition and Concurrent Finite Element Analysis for Hypercube Multiprocessors Computers", *Comp. Meth. Appl. Mech. Eng.*, Vol. 70, No. 1, (1988) pp. 27-58.

[10] C. Farhat, "A Simple and Efficient Automatic FEM Domain Decomposer", *Computers & Structures*, Vol. 28, No. 5, (1988) pp. 579-602.

[11] C. Farhat, E. Wilson and G. Powell, "Solution of Finite Element Systems on Concurrent Processing Computers", *Engineering With Computers*, Vol. 2, No. 3, (1987) pp. 157-165.

[12] C. Farhat and E. Wilson, "Concurrent Iterative Solution of Large Finite Element Systems", *Communications in Applied Numerical Methods*, Vol. 3, No. 4, (1987) pp. 319-326.

[13] C. Farhat and E. Wilson, "Modal Superposition Analysis on Concurrent Multiprocessors", *Engineering Computation*, Vol. 3, No. 4, (1986) pp. 305-311.

[14] C. Farhat, E. Pramono and C. Felippa, "Towards Parallel I/O in Finite Element Simulations", *Int. J. Num. Meth. Eng.*, Vol. 28, (1989)

[15] M. Benten, C. Farhat and H. Jordan, "The Force for Efficient Multitasking on the CRAY Series of Supermultiprocessors", *Proceedings of the Fourth International Symposium on Science and Engineering on CRAY Supercomputers*, Minneapolis, Minnesota, Oct. 12-14, (1988) pp. 389-406.

[16] C. Farhat and L. Crivelli, "A General Approach to Nonlinear FE Computations on Shared Memory Multiprocessors", *Computer Methods in Applied Mechanics and Engineering*, Vol. 72, No. 2, (1989) pp. 153-172.

[17] A. George and J. Liu, "Computer Solution of Large Sparse Positive Definite Systems", Prentice Hall, Inc., Englewood Cliffs, N. J., 1981.

[18] C. Farhat and E. Wilson, "A Parallel Active Column Equation Solver", *Computers & Structures*, Vol. 28, No. 4, (1988) pp. 289-304.

[19] E. Poole and A. Overman, "The Solution of Linear Systems of Equations with a Structural Analysis Code on the NAS CRAY-2", *NASA CR 4159*, December, 1988.

[20] J.G. Lewis and H.D. Simon, "The Impact of Hardware Gather/Scatter on Sparse Gaussian Elimination", *SIAM j. Sci. Stat. Comp.*, Vol. 9, No. 2, (1988) pp. 304-311.

[21] C. Ashcraft, R. Grimes, J. Lewis, B. Peyton, and H. Simon, " Recent Progress in Sparse Matrix Methods for large linear systems", *International Journal on Supercomputer Applications*, Vol. 1, No. 4, (1987) pp. 10-30.

[22] H. Simon, P. Vu and C. Yang, "Performance of a Supernodal General Sparse Solver on the CRAY Y-MP: 1.68 GFLOPS with Autotasking", *Applied Mathematic Technical Report SCA-TR-117*, Boeing Computer Services, March, 1989.

[23] C. Farhat, N. Sobh and K. C. Park, "Dynamic Finite Element Simulations on the Connection Machine", *Proceedings of the Conference on Scientific Applications of the Connection Machine*, ed. by H. Simon, World Scientific, (1989) pp. 217-233.

[24] C. Farhat, N. Sobh and K. C. Park, "Transient Finite Element Computations on 65,536 Processors: The Connection Machine", *Center for Space Structures & Controls, Rept. CU-CSSC-89-06*, University of Colorado at Boulder, Boulder CO, 1989.

[25] C. Farhat, "On the Mapping of Massively Parallel Processors Onto Finite Element Graphs", *Computers & Structures*, Vol. 32, No. 2, (1989) pp. 347-354.

INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 28, 2541–2553 (1989)

TOWARDS PARALLEL I/O IN FINITE ELEMENT SIMULATIONS

CHARBEL FARHAT, EDDY PRAMONO AND CARLOS FELIPPA

Department of Aerospace Engineering, and Center for Space Structures and Controls, University of Colorado at Boulder, Boulder, CO 80309-0429, U.S.A.

SUMMARY

I/O issues in finite element analysis on parallel processors are addressed. Viable solutions for both local and shared memory multiprocessors are presented. The approach is simple but limited by currently available hardware and software systems. Implementation is carried out on a CRAY-2 system. Performance results are reported.

1. INTRODUCTION

Several parallel processor projects have already resulted in commercial multiprocessors (iPSC, AMETEK, NCUBE, Connection Machine, Encore Multimax, Sequent, ALLIANT FX/8, CRAY X-MP, CRAY-2, etc.). These machines cover a broad spectrum in terms of three factors: (a) granularity, ranging from 2 to 65,536 processors, (b) peak performance, from 0·9 to 20,000 Mflops and (c) cost, from \$0·125 M to \$10 M. Other projects are still under development worldwide (GF-11, NYU/IBM, SUPRENUM, Myrias, etc., see Reference 1 for details). Some numerical algorithms have been revised, and some completely redesigned, for implementation on these multiprocessors.[2]

Solid mechanics and structural analysis are important major application areas for parallel computing. This is reflected by the continuously increasing number of publications on this topic over the last few years. An extensive list of references on finite element computations and supercomputing may be found in Reference 3. In these references various aspects of the subject, such as parallel element-by-element procedures and linear solvers have been investigated, and implementation schemes have been proposed and assessed. However, no attempt has been made to address, investigate and/or experiment on parallel I/O.

It is very well known that I/O manipulations can easily dominate the execution time of a finite element code. Hence, speeding up these manipulations through parallel processing should be of primary concern. This paper attempts to achieve this goal. Section 2 summarizes the occurrence of I/O in finite element computations. Section 3 reviews the basic features of parallel processors and emphasizes their I/O capabilities and limitations. In Section 4, two simple approaches for handling parallel I/O on multiprocessors are proposed. Section 5 specializes our views to the CRAY-2 supermulticomputer and reports on our 'hands on' experience with it. Remarks and conclusions are offered in Section 6.

2. I/O IN FINITE ELEMENT COMPUTATIONS

Realistic finite element modelling of real engineering systems involves the handling of very large data spaces which can amount to several gigabytes of memory. To cope with this, many programs

0029–5981/89/122541–13$06.50

Received 7 November 1988
Revised 23 January 1989

in the general area of solid mechanics and structural analysis use out-of-core data base management systems. However, I/O traffic between the disk and the processor memory slows down the computations significantly and increases even more significantly the overall cost of the analysis.

In a typical finite element analysis, nodal and element data are retrieved from a storage disk before their processing, then stored back on the same storage disk after their processing has been completed. Examples include the transfer of nodal point co-ordinates, elemental mass and stiffness matrices in element-by-element computational procedures, and of history response arrays in time-stepping algorithms for linear and non-linear dynamics. Other examples include the movement, into core and out of core, of blocks of an assembled stiffness or mass matrix in original or factored form, and the output on disk of the final results of an analysis. Table I is borrowed from Reference 4. It summarizes the comparative elapsed times for CPU and I/O on a Vax 11/780 of an analysis of a cylindrical tube with a viscoplastic behaviour. The frontal method,[5] which is known to be I/O bound, was used for the solution phase. Data transfers were carried out through Fortran I/O.

Clearly, the performance results reported in Table I underline the potential of I/O for bottlenecks in finite element computations. Speeding up all the computational phases through parallel processing is certainly an important issue. However, reducing the amount of time spent in data transfers can become even more of an issue.

3. ARCHITECTURE AND HARDWARE

Recently, several parallel computers have arrived on the scene with a variety of different architectures. These generally can be described through three essential elements, namely, granularity, topology and control:

- Granularity relates to the number of processors and involves the size of these processors. A fine-grain multiprocessor features a large number of usually very small and simple processors. The Connection Machine (65,536 processors) is such a massively parallel supercomputer. NCUBE's 1024-node model is a comparatively medium-grain machine. On the other hand, a coarse-grain supermultiprocessor is typically built by interconnecting a small number of large, powerful processors—usually vector processors. CRAY X-MP (4 processors), CRAY-2 (4 processors) and ETA-10 (8 processors) are examples of such supermultiprocessors.
- Topology refers to the pattern in which the processors are connected and reflects how data will flow. Currently available designs include hypercube arrangements, networks of busses and banyan networks.
- Finally, control describes the way the work is divided up and synchronized.

Table I. Comparison of CPU and I/O costs for an FE analysis on a Vax 11/780

Phase	CPU (sec)	I/O (sec)
Integration of constitutive equations	18·28	41·00
Assembly of external forces	0·05	0·00
Assembly of viscoplastic forces	13·90	100·00
Solution	2·75	36·00
Overall	35·18	177·00

Another important architectural distinction, and one that is most relevant to our effort in this paper, is that which characterizes memory organization. In shared memory systems, all processors access the same (global) large memory system. These multiprocessors are usually coarse-grained because the bus to memory saturates and/or becomes prohibitively expensive above a few processors. On the other hand, in local memory systems each processor can access only its own (local) memory. Independent processors communicate by sending each other messages. It appears that parallel computers in this class are easier to scale to a large number of processors.

Distinguishing only between shared and local memory systems does not give a complete picture of the problems that one may face when programming parallel processors. Granularity and control also have their influence. The Connection Machine (65,536 processors) and Intel's hypercube iPSC (128 processors) are both local memory systems. However, the former is an SIMD (single instruction multiple data streams) machine where a single program executes on the front end and its parallel instructions are submitted to the processors. The latter is an MIMD (multiple instruction multiple data streams) parallel processor where separate program copies execute on separate processors. The granularity of a parallel processor, which seems to affect other architectural elements, substantially affects the computational strategy and parallel I/O, as will be shown.

Multiprocessors with any of the above architectures have the capability to substantially speed up operations in scientific applications. However, I/O is still their Achilles heel. Before discussing parallel I/O strategies and their implementations, we mention that, at the time of writing this paper and to our best knowledge, only a few systems offer parallel I/O capabilities. These include NCUBE at one extreme, with up to 1024 processors and their small local memories, and CRAY-2 at the other, with four vector processors and a large shared memory. Parallel disk I/O capabilities are also available on the Connection Machine.

On NCUBE, each node (processor) has a direct connection to an I/O board through one of the system I/O channels, so that parallel disk access is possible. Generally speaking, on local memory multiprocessors a bundle of processors may be assigned a local disk through a dedicated I/O channel.

On CRAY-2, multitasking I/O is possible on a limited basis.[6] Different tasks can perform I/O simultaneously on different files. This is primarily for the following two reasons.

1. The non-deterministic nature of task execution limits I/O on the same file by different tasks. In other words, problems may arise not only from mapping two distinct hardware processors on to the same file, but also from mapping two logical processes on to the same file. Our experience has shown that the latter situation complicates even sequential I/O on most shared memory multiprocessors (ALLI-ANT FX/8, Encore Multimax, Sequent Balance), mainly because of the problem of maintaining consistency in the buffer sizes between distinct processes.
2. The fact that parts of the support library are *critical* regions that are protected from simultaneous access, and therefore limit the parallelism that one could otherwise exploit.

The next section presents two simple approaches for parallel disk I/O that are viable within the limitations of the currently available hardware and system software for local memory and shared memory multiprocessors.

4. TWO SIMPLE APPROACHES

Most of the computational strategies recently proposed for parallel finite element computations are based on the principle of *divide and conquer*: that is, divide the computing task into a number of

subtasks that are either independent or only loosely coupled, so that computations can be made on distinct processors with little communication and sharing. For example, if using this strategy the structure shown in Figure 1 is to be analysed using N_p processors, it is first automatically subdivided into a set of N_p (or a multiple of N_p) balanced substructures.[7]

Depending on the size of the problem and the granularity of the parallel processor, a substructure would contain anywhere from a single element to several thousand of them. Then, each processor is assigned the task of analysing one—or several—substructure(s). While this approach is feasible on most parallel computers, it is especially interesting for local memory multiprocessors. Each processor is attributed a simple data structure. Only the node geometry and element properties associated with its assigned substructure are stored within its RAM. In addition, formation and reduction of the stiffness matrix for that region require no interprocessor communication. Finally, after the displacements have been found, the postprocessing of subdomain stresses can be done concurrently.[8]

Local memory approach

It is very natural to extend this substructuring idea to achieve parallel I/O in the finite element analysis. For example, on local memory multiprocessors, it is tempting to imagine that, in the same way that a processor is assigned its own memory, it could be attributed its own set of I/O devices (I/O controller, disk drive, etc.) and its own files. Then, each processor would read/write the data for its subdomain from its own files and through its own data base, in parallel with the

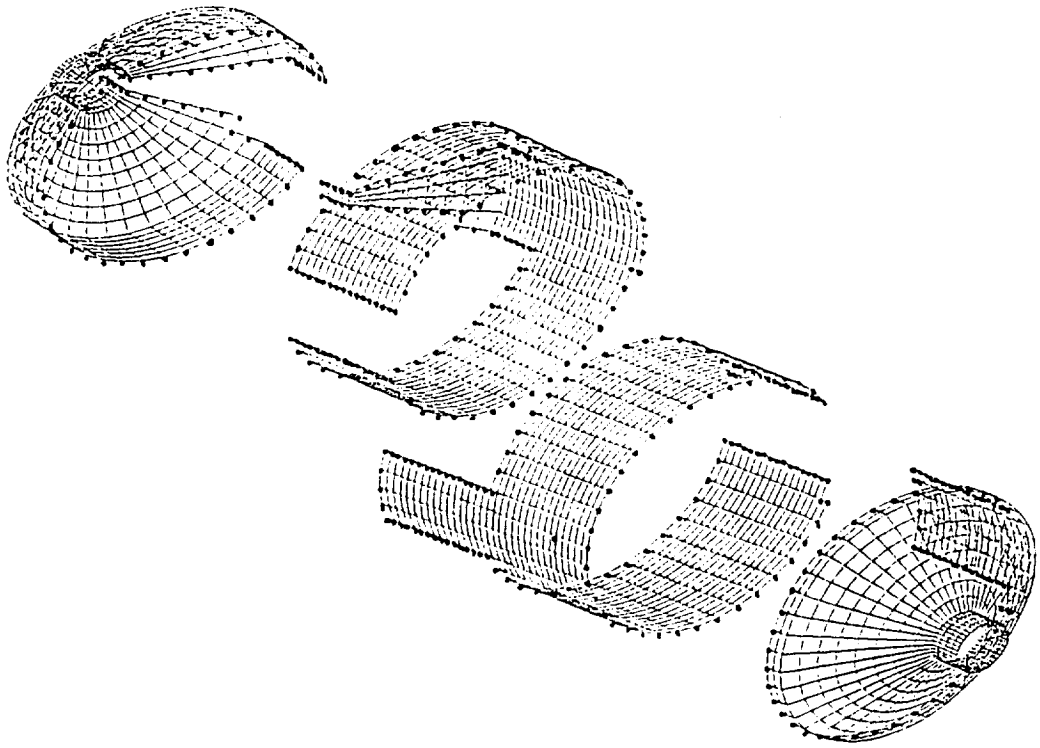


Figure 1. Dividing and conquering a mesh

other processors. If assigning an I/O controller and/or a disk drive to each processor is impractical and/or impossible, as is probably the case for a fine-grain system, for a cluster of processors it is possible. For concreteness, we overview NCUBE's I/O subsystem for a configuration with 1024 processors (Figure 2).

The 1024 computational nodes can be thought of as eight groups of 128 processors each. Each group consists of 16 clusters of eight directly connected computational nodes. (Recall that where 2^d processors are arranged in a hypercube pattern, d of them are directly connected.) Within a cluster, each computational node has 22 direct memory access (DMA) channels. Twenty of these are paired into 10 bi-directional communication links and are used for messages (data transfer) to and from direct neighbours. The remaining pair of channels is bundled together with the 127 other pairs of the same group and brought through the backplane to one of the I/O slots. This results in what is called a system I/O channel. Clearly, an NCUBE system with 1024 computational nodes has 8 system I/O channels. Next, an I/O board is interfaced to a backplane to serve the 128 processors organized into 16 clusters of 8 directly connected computational nodes. Another cube with 16 nodes is connected to the other side of the I/O board. Each of these nodes has direct access to a disk through a private controller. Hence, each of these 16 nodes can directly serve one of the 16 clusters of computational nodes. In other words, each computational node within a cluster of eight directly connected processors has a direct access to a disk through a dedicated node connected to the other side of an I/O board. In summary, the I/O subsystem outlined above supports 1024 processors with 8 system I/O channels, 128 controllers and 128 disks. It has the potential for a minimum I/O speed-up of 128.

In the following, the words 'host' and 'DBM' denote respectively the collection of processors serving an I/O board and a generic sequential data base manager. After a given finite element

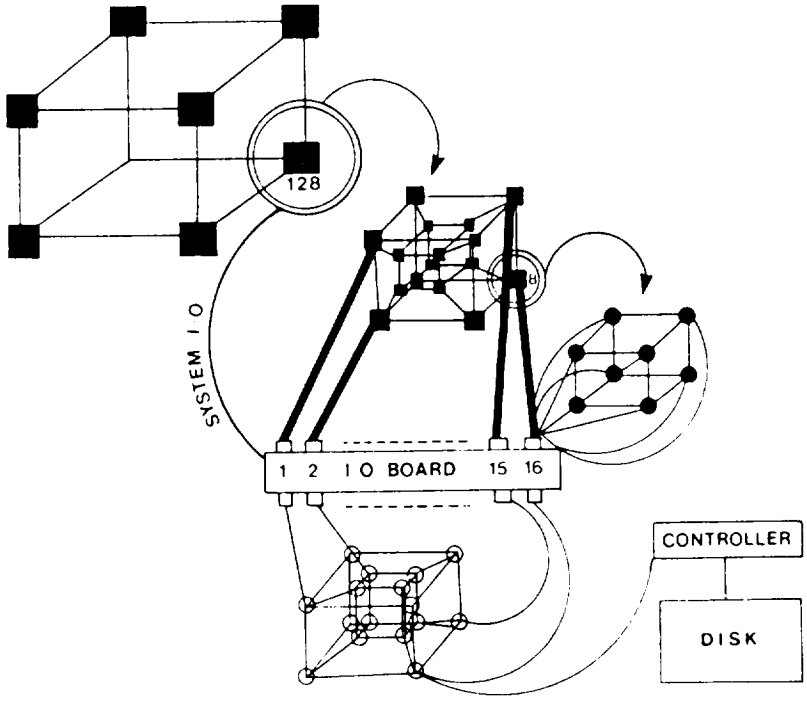


Figure 2. NCUBE's I/O subsystem

domain is decomposed, it is grouped into regions R_i, $i=1, \ldots, 128$, each containing eight (preferably adjacent) subdomains $D_j^{R_i}$, $j=1, \ldots, 8$. A host processor p_i^h is uniquely mapped onto each region R_i. It is assigned the task of handling I/O manipulations associated with computations performed primarily in the eight subdomains within R_i. Basically, since p_i^h is directly connected from one side to each of the eight processors p_j assigned to subdomains $D_j^{R_i}$, and from the other to its dedicated disk, it can directly transfer data from p_j's RAM, $j=1, \ldots, 8$, to the disk and vice versa. This is implemented as follows. Each host processor p_i^h is loaded with the same program driver, which we will call the *listener*, and the same copy of DBM. The main task of the listener is to listen to processor p_j's requests for I/O, $j=1, \ldots 8$. These requests may be:

- receive data from p_j and store it in disk using DBM;
- retrieve data from disk through DBM and send it to p_j;
- retrieve data from disk through DBM, send it to another host processor p_j^h together with the instruction of broadcasting it to a specified number of computational nodes that are directly connected to p_j^h; this particular operation implements potential exchange of data between subdomains.

Consequently, only a small amount of RAM is required on a host processor. It corresponds to the storage requirements of an executable listener with its buffer for data transfer and of an executable code of a DBM system. Note that the size of a message is not limited by the amount of buffer memory available on the host processor but by the amount of memory allocated by the operating system for a message passing operation. Hence, a large record of data may need to be split and transferred via more than one message.

On most local memory multiprocessors, a node sends a message to another node (or set of nodes) by typically executing a 'send' system call with the following parameters: (a) a set of destination nodes, (b) a process id, (c) a message type, (d) a buffered message or a pointer to the message buffer and (e) the length of the message (usually in bytes). Similarly, a node initiates the receipt of a message from another process by issuing a 'receive' system call with parameters corresponding to the 'send' call. In many cases, 'send' and 'receive' cannot be coordinated. This is the case, for example, when a host processor does not have *a priori* the schedule of the messages that computational nodes will issue during the finite element analysis. In such situations, a host processor can 'probe' for all pending messages of a specific type and act when a message of a given type is available for reception. A computational node program transmits its instructions and data to the listener via a message buffer denoted here by BUFFER, and formatted as indicated below:

KEY 1	KEY 2	KEY 3	INSTRUCTION	TAGGED DATA	
				TAG	DATA

Example:

4	23	4	STORE IN FILE 'STIFF'

BUFFER[1] points to the location in BUFFER of the instruction stream to be processed then delivered to DBM.

BUFFER[2] points to the location in BUFFER of the data stream to be processed then delivered to DBM.

BUFFER[3] contains the number of continuing messages.

On most local memory multiprocessors, messages issued by a node to a same other processor are received in the same order that they are sent. Hence, if computational node p_k^c sends to a host processor p_i^h an instruction and/or data message followed by two other messages containing the remaining of the data, p_i^h receives first the instruction tailed with the first part of the data, then the rest of the data. However, problems may occur if two different computational nodes p_j^c and p_k^c each send a set of continued messages to the same host processor p_i^h. In this case, the host processor might receive the messages in disorder. To eliminate ambiguity, the logic of the listener is implemented as following:

- it receives a first message, identifies its type and the number of continuation messages;
- it probes for pending continuation messages of the same type, receives and processes them (pending messages of a different type are queued by the operating system);
- it listens to another starting message.

Next, we describe another approach, this one for multiprocessors with shared memory.

Shared memory approach

It is possible to simulate a local memory multiprocessor with a shared memory one, by partitioning the global memory into locations each fetched always by the same processor. Consequently, the approach presented above for parallel I/O on local memory multiprocessors identically applies to shared memory machines. However, we see three reasons for adopting a different approach on shared memory parallel processors.

1. Mimicking a local memory system on a shared memory one defeats the purpose of sharing information.
2. As described previously, the local memory approach ties a given processor indefinitely to the I/O needs of a specific region of the finite element domain. We refer to this as a static mapping of a processor onto a subdomain. A key issue in performance of parallel processing is load balancing. When the amount of work (computations + I/O) to be performed can be predicted for each region of a mesh, it can be evenly distributed among the processors through a careful partitioning of the geometrical domain and an adequate mapping of the processors onto the resulting subdomains. When such predictions are not possible, a dynamical load balancing algorithm is necessary for optimal performance on parallel processors. Local mesh refinements in adaptive computations and local material properties changes in elastoplastic analyses are examples of situations where the mapping of a processor onto a subdomain needs to be re-defined at each computational step. Note that on local memory multiprocessors re-mapping of the processors on the finite element domain implies a substantial amount of data transfer between the processors, and what is gained with the even redistribution of computations and I/O is lost with interprocessor communications. On the other hand, the dynamical re-mapping of the processors of a shared memory system for complex finite element computations can be achieved at almost zero overhead cost.[9]
3. Because of the ability of a processor to reference any location in the global memory, shared memory multiprocessors provide the programer with a wider variety of parallel strategies than do local memory systems. One ought to take advantage of this fact. It will be shown that our approach for parallel I/O in finite element computations on shared memory multiprocessors embeds our approach on local memory machines as a particular case.

Unlike the previous approach, a single executable version of a sequential DBM is stored in the global memory of the multiprocessor. Moreover, there is no need for a listener since all processors

can directly access DBM, the I/O library and the disks. However, the core of the computational routines needs to be slightly modified to distinguish between global variables, which are shared by all the defined processes, and local variables, which have a single name to ease programming but a distinct value for each process. Using parallel constructs such as those of *The Force*[10] reduces the nature and amount of modifications to one: that of preceding each Fortran declaration of a variable by either the word SHARED or the word PRIVATE. While our approach for parallel I/O on local memory machines is subdomain oriented, it is purely data oriented on shared memory multiprocessors. In the following, we distinguish between four classes of parallel I/O requests.

1. *Synchronous request with private variables [SRPV].* All processes request I/O operations simultaneously, each with a private buffer area. Typically, this happens in an SIMD programming style, even when the multiprocessor is of the MIMD type. For example, suppose that all of the processes have to perform the same amount of identical computations but on distinct sets of data, and suppose that these computations are such that out-of-core temporary storage associated with each set of data is needed. Here, identity in the instructions calls for synchronous parallel I/O, and independence in the data sets calls for private temporary storage.

2. *Synchronous request with shared variables [SRSV].* All processes request I/O operations simultaneously using a common buffer area. For example, consider the previous case with the additional assumption that the nature of the computations requires shuffling of the temporary data between processes.

3. *Asynchronous request with private variables [ARPV].* A process requests I/O operations independently of another process and with a private buffer area. These requests are identical to those on MIMD local memory multiprocessors. For example, the entire approach described earlier for local memory multiprocessors fits into this class of I/O requests.

4. *Asynchronous request with shared variables [ARSV].* A process requests I/O operations independently of another process using a shared buffer area.

Clearly, the four classes of I/O request described above cover all the possibilities on a shared memory multiprocessor. At this point we introduce the following remarks.

1. Synchronous and asynchronous refer to the initiation of the processes and not to their execution. Two processes can be initiated at the same time but executed at two different times, for example, if one processor were tied up by a previous process.
2. [ARSV] requires that a pointer to the location in the buffer of the starting address for storage and/or retrieval of data be carefully computed by its owner process, in order not to destroy the information by overlapping the data.
3. The multiprocessor will take no responsibility for automatically generating synchronization. It is entirely the responsibility of the user to make sure that the shared data to be created by one process and to be read by another process are available before an [ARSV] is issued. Typically, one invokes an explicit synchronization instruction for that purpose.

Next, we describe a simple parallel I/O manager, PIOM, which copes with our four defined I/O requests. First, note that PIOM can handle [ARPV] and [ARSV] exactly as in the sequential case. Hence, [SRPV] and [SRSV] are the requests which call for a modification of a basic sequential I/O manager. Moreover, after PIOM recognizes that [SRPV] deals with private variables, it can treat it exactly as [ARPV], with the difference that calling processes are responded to in parallel.

In other words, [SRPV] is treated as a set of simultaneous [ARPV]. Consequently, the treatment of [SRSM] is PIOM's major task.

For each file related to an [SRSV], PIOM consults an I/O table. If the request is for storing data, PIOM's logic is as follows:

(S1) it partitions the information into a number of contiguous subsets equal to the number of calling processes, each subset containing an equal amount of data.
(S2) for each subset, it computes a pointer to the location in the shared buffer where the subset data stream begins.
(S3) for each calling process, it creates a corresponding 'S' I/O process. Each 'S' I/O process is assigned a subset of the data with its pointer.
(S4) it reports in the I/O table the total number of created 'S' I/O processes. For each 'S' I/O process, it specifies the length of its assigned data and their destination on a hardware device.
(S5) it fires the 'S' I/O processes. Each 'S' I/O process re-partitions its assigned data into a number of records that is a multiple of the total number of available processors on the machine, then calls DBM independently of another 'S' I/O process. The reason for the internal partitioning will become clearer in the remarks which follow.

On the other hand, if the request is for retrieving data, PIOM's logic becomes:

(R1) it retrieves the I/O table corresponding to the file. If the number of calling processes is equal to the number of processes registered in the table (the 'S' processes which originally stored the file), the inverse logic to the 'store' case is followed and the data are retrieved in parallel. If not:
(R2) for each registered 'S' I/O process, it partitions its subset of information into a number of contiguous blocks of data equal to the number of calling processes, each block containing an equal amount of data.
(R3) for each block, it computes a pointer to the location in the shared buffer where the block data stream begins.
(R4) for each registered 'S' I/O process, it creates a number of 'R' I/O processes equal to the number of calling processes. Each 'R' I/O process is assigned a block of the subset data with its pointer.
(R5) it fires the 'R' I/O processes.
(R6) it follows with the next 'S' process to be retrieved.

Clearly, step (S5) and steps (R2) to (R6) allow for a file that was written in parallel using p processes to be read in parallel using p^* processes, where p^* is different from p. In this case, the retrieval of the file is carried out in p waves, each of a degree of parallelism equal to p^*. The overall logic is summarized in Figure 3.

In order to illustrate the flexibility of this approach, we describe two simple examples. Example 1 illustrates the distinction between the mapping of the processors on the data during I/O and during computations. Example 2 illustrates the ability of the approach to handle dynamical load balancing algorithms.

Example 1. A block of the stiffness matrix is to be retrieved from disk and factored using four processors. An [SRSV] is issued to read the block of the stiffness matrix. The partitioning of the data by PIOM into contiguous subsets is shown in Figure 4(a). After the entire data are retrieved in parallel, the processors are mapped onto the stiffness matrix block in an interleaved fashion (Figure 4(b)). Next a call for a parallel active column solver[11] is issued.

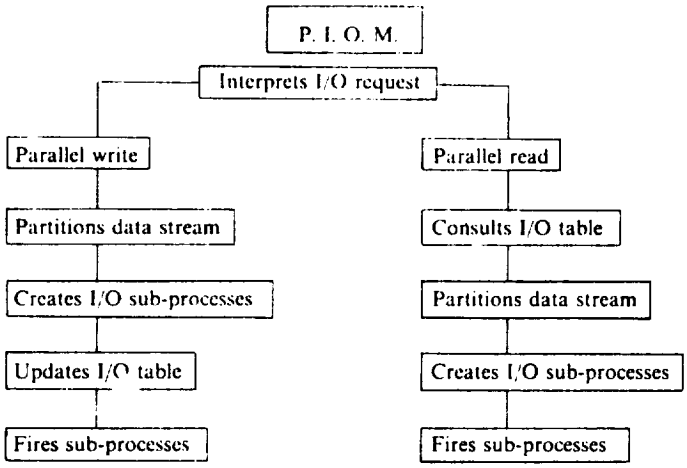


Figure 3. A parallel I/O manager

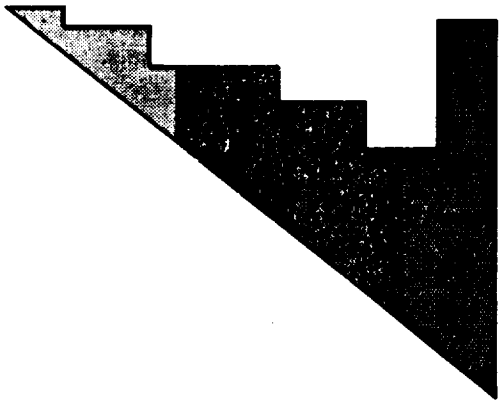


Figure 4(a). Mapping for data retrieval

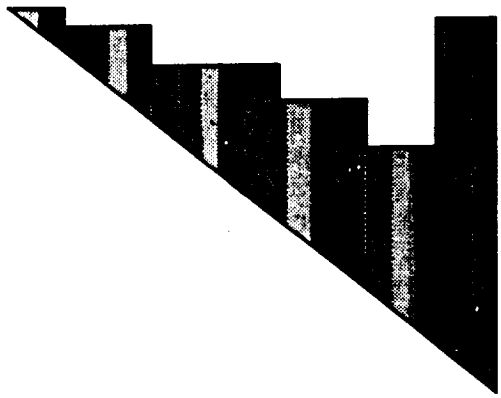


Figure 4(b). Mapping for computations

Example 2. During coloured element-by-element computations,[9] NEB elemental stiffness matrices have to be read from disk and processed for computation of residuals. Here again, an [SRSV] is issued for parallel retrieval of the data. The mapping of N_p processors on the elemental stiffnesses is initially prescribed only for the first N_p elements of NEB. After that, the elements are processed as soon as a processor becomes available. Hence, the question of which element turns out to be non-linear and which turns out to remain linear does not affect the load balancing. Moreover, another [SRSV] for another set of NEB elements to be read in parallel can be issued as soon as a processor is done with its computations and while all the others are still tied up with the last N_p-1 elements to be processed.

5. IMPLEMENTATION ON THE CRAY-2

The CRAY-2 supermultiprocessor is characterized by a global memory of 256 million 64-bit words, four background processors and a clock cycle of 4·1 nanoseconds. It is the target machine

for our first experiments with parallel I/O. The four background processors can operate independently on separate jobs or concurrently on a single problem (CRAY Research Inc. refers to this as multitasking). Each processor can independently coordinate the data flow between the system common memory and all the external devices across four high-speed I/O channels.

As stated in Section 3, multitasking I/O is possible on CRAY-2 with the restriction that different processes can simultaneously perform I/O only on separate files that are located on different disks. The shared memory approach presented in Section 4 is slightly modified to accommodate CRAY-2's limitations. Any file specified by the user is automatically partitioned by PIOM into a number of 'sub-files' equal to the number of I/O processes. The partitioning and the sub-files names are transparent to the user. They are recorded in the I/O table for further I/O processing. Three algorithms—chunking, interleaving and interleaving with buffering—are considered for mapping the data onto the sub-files.

The blocking algorithm is a straightforward implementation of steps (S1) and (S2) described in Section 4. The data to be transferred are partitioned into a number of subsets of contiguous data equal either to the number of available disks, or to the number of calling processes, whichever is smaller. This algorithm is very fast, but has two main drawbacks:

- it may not utilize all the available processors for some I/O read requests. For example, consider the case where the information to be read corresponds to data that were previously written by PIOM on the same physical disk.
- appending to an existing file may not be efficient.

The reader should note that the words 'a file' refer to what is in the user's mind. PIOM always splits 'the' file into as many sub-files as there are available disks. Appending an existing file, and reading from an arbitrary location in a file, are two operations which are better handled by the interleaving algorithm. Basically, if N_d denotes the number of available disks, and D denotes the data stream to be processed, this algorithm partitions D into a set of segments S_i of arbitrary sizes, and assigns each segment S_i to disk $mod\ (i, N_d)$ (Figure 5).

The interleaving algorithm above requires the I/O manager to be invoked a number of times; that number is equal to the ratio of the number of segments divided by the number of disks, N_d. Each time the I/O manager is invoked, it conveys the information segment directly from main memory to auxiliary storage or vice versa. Another approach consists of first buffering the segments of a given parallel I/O process in an order that reflects their layout in their assigned disk, then invoking only once the I/O manager to execute the parallel I/O request.

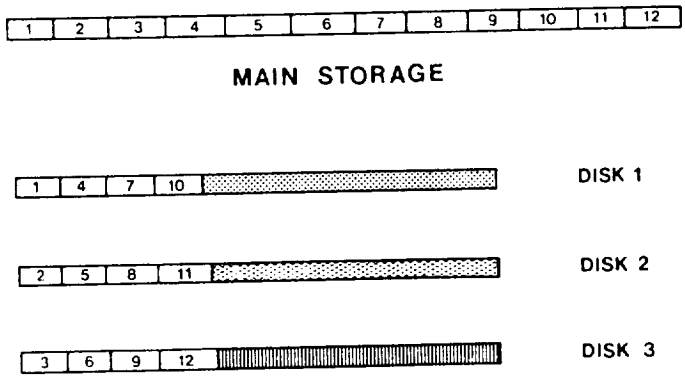


Figure 5. Interleaving data on disks

The practical implementation of the three algorithms described above is carried out with *The Force*,[10] a preprocessor which provides a FORTRAN style parallel programming language utilizing a set of parallel constructs. [ARPV] and [ARSV] are implemented with regular CALL statements to PIOM: each process executes independently of the other its call to a subroutine, delivering a different data buffer. [SRPV] and [SRSV] are implemented with the FORCECALL executable statement to PIOM: this construct causes the entire processes to jump and execute parallel calls to PIOM. In the latter case, the processes' ids are automatically passed to PIOM. Performance results for the three algorithms are reported in Tables II, III and IV. Tables II and III are associated with a segment size equal respectively to 1 sector (1024 bytes) and 1 track (65536 bytes). They compare the performance of the three algorithms for a parallel read request consisting of retrieving a 24 Mbytes data stream using 2 CPU's. Wall-clock, system time and user time are reported. System time corresponds to the time elapsed in PIOM managing parallelism. User time is associated with I/O overhead.

Table II. Performance results

Parallel read—Information size = 24 Mb—Segment size = 1 sector			
	Clock (sec)	System (sec)	User (sec)
Chunking	1·222	7·777E − 5	3·482E − 2
Interleaving (buffering)	4·885	2·0565	2·922E − 2
Interleaving	6·604	0·5916	2·523

Table III. Performance results

Parallel read—Information size = 24 Mb—Segment size = 1 track			
	Clock (sec)	System (sec)	User (sec)
Chunking	1·222	7·777E − 5	3·482E − 2
Interleaving (buffering)	4·120	1·965	3·917E − 2
Interleaving	1·159	9·30E − 3	7·643E − 2

Table IV. Speed-up

Chunking algorithm—Information size = 200 Mb			
	1 Process (sec)	2 Processes (sec)	3 Processes (sec)
Write	21·603	12·208	10·036
Speed-up	1·0	1·77	2·15
Read	1·165	0·584	0·390
Speed-up	1.0	1·99	2·99

For a segment size equal to 1 sector, the chunking algorithm is by far the fastest. For this example, the number of segments to be processed, which is given by the ratio information size/segment size, is such that the interleaving algorithm has a high overhead associated with I/O instructions, and the interleaving with buffering algorithm has a high overhead associated with PIOM's instructions.

However, for a segment size equal to 1 track, the interleaving algorithm performs best. This is because for the given segment size, fewer segments need to be processed and less time is elapsed in I/O instructions.

The above results provide the user with a guidance for the selection of any of the three implemented parallel algorithms.

Table IV reports the wall-clock time and measured speed-up for parallel read/write requests using the chunking algorithm. Only three out of the four available CRAY-2 CPU's were activated because only three different disks were available. For each case, the size of the data stream to be processed was fixed to 200 Mbytes.

Clearly, very high speed-ups are achieved for both read/write parallel requests. Note, however, the pathological performance for the write case with three processors. We have not yet been able to justify this particular result.

6. CONCLUSION

Finite element analyses are known to be I/O bounded. In this paper, two approaches are presented to speed I/O manipulations through parallel processing. The first approach deals with local memory MIMD multiprocessors and is based on a substructuring technique. The second approach is dedicated to shared memory multiprocessors. It has been implemented and tested on a CRAY-2 system with four CPU's. The obtained performance results confirm the potential of parallel processing in I/O manipulations. Future work will address I/O operations on the data vaults of the Connection Machine (65,536 processors).

ACKNOWLEDGEMENT

The first author wishes to acknowledge the partial support of a CDC PACER Fellowship Award, with Dr R. F. Woodruff as technical monitor. The second and third authors acknowledge partial support by NASA Langley under Grant NAG-1-756, with Drs J. Housner and J. Stroud as technical monitors.

REFERENCES

1. O. A. McBryan, 'State of the art in highly parallel computer systems', in A. K. Noor (ed.), *Parallel Computations and Their Impact on Mechanics*, American Society of Mechanical Engineers, New York, 1987.
2. J. M. Ortega and R. G. Voigt, 'A bibliography on parallel and vector numerical algorithms', *NASA CR-178335*, 1987.
3. D. W. White and J. F. Abel, 'Bibliography on finite elements and supercomputing, *Commun. Appl. Numer. Methods*, **4**, 279–294 (1988).
4. J.-C. Golinval, 'Calcul de la response d'une structure en viscoplasticite cyclique', *Rapport SF-133*, Aerospace Research Laboratory, University of Liege, 1985.
5. B. Irons, 'A frontal solution program for finite element analysis', *Int. j. numer. methods eng.*, **2**, 5–32 (1970).
6. *CRAY-2 Multitasking Programmer's Manual—SN-2026*.
7. C. Farhat, 'A simple and efficient automatic FEM domain decomposer', *Comp. Struct.*, **28**, 579–602 (1988).
8. C. Farhat and E. Wilson, 'A new finite element concurrent computer program architecture', *Int. j. numer. methods eng.*, **24**, 1771–1792 (1987).
9. C. Farhat and L. Crivelli, 'A general approach to nonlinear FE computations on shared memory multiprocessors', *Comp. Methods Appl. Mech. Eng.*, **72**, 153–171 (1989).
10. H. Jordan, M. Benten and N. Arenstorf, *Force User's Manual*, Department of Electrical and Computer Engineering, University of Colorado, Boulder, Colorado.
11. C. Farhat and E. Wilson, 'A parallel active column equation solver', *Comp. Struct.*, **28**, 289–304 (1988).

A METHOD OF FINITE ELEMENT TEARING AND INTERCONNECTING AND ITS PARALLEL SOLUTION ALGORITHM

Charbel Farhat
Department of Aerospace Engineering Sciences
and Center for Space Structures and Controls
University of Colorado at Boulder
Boulder, CO 80309-0429, U. S. A.

and

Francois-Xavier Roux
O. N. E. R. A. Groupe Calcul Parallele
29 Av. de la Division Leclerc
BP72 92322 CHATILLON Cedex, FRANCE

Abstract. A novel domain decomposition approach for the parallel finite element solution of equilibrium equations is presented. The spatial domain is partitioned into a set of totally disconnected subdomains, each assigned to an individual processor. Lagrange multipliers are introduced to enforce compatibility at the interface nodes. In the static case, each floating subdomain induces a local singularity that is resolved in two phases. First, the rigid body modes are eliminated in parallel from each local problem and a direct scheme is applied concurrently to all subdomains in order to recover each partial local solution. Next, the contributions of these modes are related to the Lagrange multipliers through an orthogonality condition. A parallel conjugate *projected* gradient algorithm is developed for the solution of the coupled system of local rigid modes components and Lagrange multipliers, which completes the solution of the problem. When implemented on local memory multiprocessors, this proposed method of tearing and interconnecting requires less interprocessor communications than the classical method of substructuring. It is also suitable for parallel/vector computers with shared memory. Moreover, unlike parallel direct solvers, it exhibits a degree of parallelism that is not limited by the bandwidth of the finite element system of equations.

1. Introduction

A number of methods based on domain decomposition procedures have been proposed in recent years for the parallel solution of both static and dynamic finite element equations of equilibrium. Most of these methods are derived from the popular substructuring technique. Typically, the finite element domain is decomposed into a set of subdomains and each of these is assigned to an individual processor. The solution of the local problems is trivially parallelized and usually a direct method is preferred for this purpose. Parallel implementations of both a direct (Farhat, Wilson [1]) and an iterative (Nour-Omid, Raefsky and Lyzenga [2]) solution of the resulting interface problem have been reported in the literature. A number of more original approaches have also been spurred by the advent of new parallel processors. Ortiz and Nour-Omid [3] have developed a family of unconditionally stable concurrent procedures for transient finite element analysis and Farhat [4] has designed a multigrid-like algorithm for the massively parallel finite element solution of static problems. Both of these developments relate to the divide and conquer paradigm but depart from classical substructuring.

In this paper, we present a parallel finite element computational method for the solution of static problems that is also a departure from the classical method of substructures. The unique feature about the proposed procedure is that it requires fewer interprocessor communication than traditional domain decomposition algorithms, while it still offers the same amount of parallelism. Roux [5, 6] has presented an early version of this work that is limited to a very special class of problems where a finite element domain can be partitioned into a set of disconnected but non-floating subdomains. Here, we generalize the method for arbitrary finite element problems and arbitrary mesh partitions. We denote the resulting computational strategy by "finite element tearing and interconnecting" because of its resemblance with the very early work of Kron [7] on tearing methods for electric circuit models. In Section 2, we partition the finite element domain into a set of totally disconnected subdomains and derive a computational strategy from a hybrid variational principle where the inter-subdomain continuity constraint is removed by the introduction of a Lagrange multiplier. An arbitrary mesh partition typically contains a set of floating subdomains which induce local singularities. The handling of these singularities is treated in Section 3. First, the rigid body modes are eliminated in parallel from each local problem and a direct scheme is applied concurrently to all subdomains in order to recover each partial local solution. Next, the contributions of these modes are related to the Lagrange multipliers through an orthogonality condition. A parallel conjugate *projected* gradient algorithm is developed in Section 4 for the solution of the

coupled system of local rigid modes components and Lagrange multipliers, which completes the solution of the problem. Section 5 deals with the preconditioning of the interface problem in order to speed up the recovery of the Lagrange multipliers. Section 6 emphasizes the parallel characteristics of the proposed method and Section 7 contrasts it with the method of substructures. Section 8 discusses some important issues related to the partitioning of a given finite element mesh. Finally Section 9 illustrates the method with structural examples on the distributed memory hypercube iPSC (32 processors) and the shared memory parallel/vector CRAY-2 system (4 processors), and Section 10 concludes the paper.

2. Finite element tearing and interconnecting

Here we present a domain decomposition based algorithm associated with a hybrid formulation for the parallel finite element solution of the linear elastostatic problem. However, the method is equally applicable to the finite element solution of any self-adjoint elliptic partial differential equation. For the sake of clarity, we consider first the case of two subdomains, then generalize the method for an arbitrary number of subdomains.

The variational form of the three-dimensional boundary-value problem to be solved goes as follows. Given f and h, find the displacement function u which is a stationary point of the energy functional:

$$J(v) = \frac{1}{2}a(v,v) - (v,f) - (v,h)_\Gamma$$

where

$$\begin{aligned} a(v,w) &= \int_\Omega v_{(i,j)} c_{ijkl} w_{(k,l)}\, d\Omega \\ (v,f) &= \int_\Omega v_i f_i\, d\Omega \\ (v,h)_\Gamma &= \int_{\Gamma_h} v_i h_i\, d\Gamma \end{aligned} \tag{1}$$

In the above, the indices i,j,k take the value 1 to 3, $v_{(i,j)} = (v_{i,j} + v_{j,i})/2$ and $v_{i,j}$ denotes the partial derivative of the $i-th$ component of v with respect to the $j-th$ spatial variable, c_{ijkl} are the elastic coefficients, Ω denotes the volume of the elastostatic body, Γ its piecewise smooth boundary, and Γ_h the piece of Γ where the tractions h_i are prescribed.

If Ω is subdivided into two regions Ω_1 and Ω_2 (fig. 1), solving the above elastostatic problem is equivalent to finding the two displacements functions u_1 and u_2 which are stationary points of the energy functionals:

$$
\begin{aligned}
J_1(v_1) &= \frac{1}{2}a(v_1,v_1)_{\Omega_1} - (v_1,f)_{\Omega_1} - (v_1,h)_{\Gamma_1} \\
J_2(v_2) &= \frac{1}{2}a(v_2,v_2)_{\Omega_2} - (v_2,f)_{\Omega_2} - (v_2,h)_{\Gamma_2}
\end{aligned}
$$

where

$$
\begin{aligned}
a(v_1,w_1)_{\Omega_1} &= \int_{\Omega_1} v_{1(i,j)} c_{ijkl} w_{1(k,l)}\, d\Omega \\
a(v_2,w_2)_{\Omega_2} &= \int_{\Omega_2} v_{2(i,j)} c_{ijkl} w_{2(k,l)}\, d\Omega \\
(v_1,f)_{\Omega_1} &= \int_{\Omega_1} v_{1i} f_i\, d\Omega \\
(v_2,f)_{\Omega_2} &= \int_{\Omega_2} v_{2i} f_i\, d\Omega \\
(v_1,h)_{\Gamma_1} &= \int_{\Gamma_{h1}} v_{1i} h_i\, d\Gamma \\
(v_2,h)_{\Gamma_2} &= \int_{\Gamma_{h2}} v_{2i} h_i\, d\Gamma
\end{aligned}
\tag{2}
$$

and which satisfy on the interface boundary Γ_I the continuity constraint:

$$
u_1 = u_2 \text{ on } \Gamma_I \tag{3}
$$

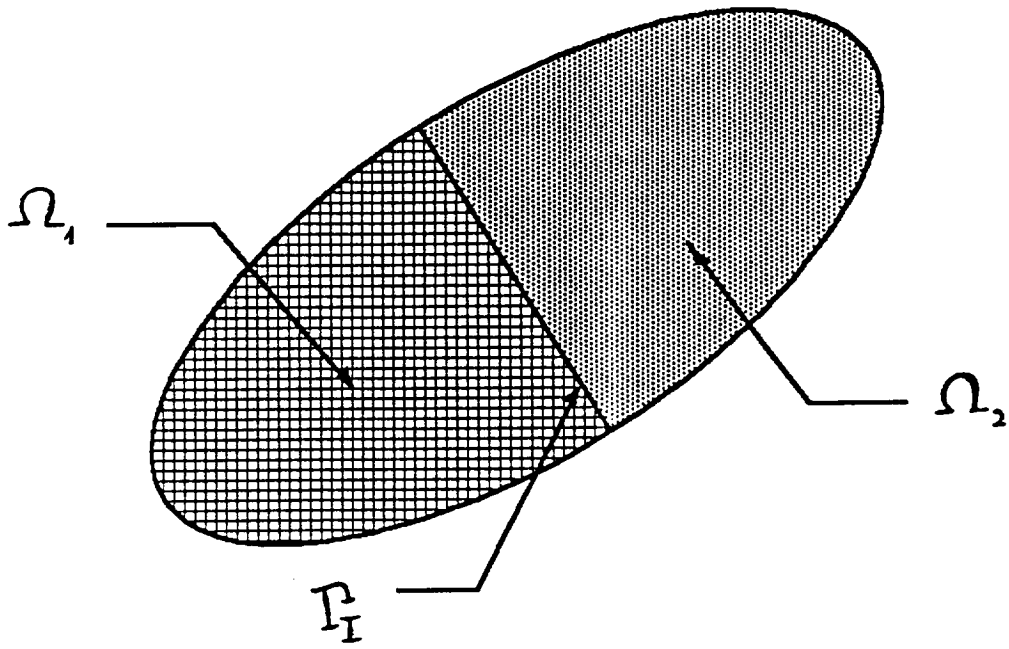


FIG. 1 *Decomposition in two subdomains*

Solving the two above variational problems (2) with the subsidiary continuity condition (3) is equivalent to finding the saddle point of the Lagrangian:

$$J^*(v_1, v_2, \mu) = J_1(v_1) + J_2(v_1) + (v_1 - v_2, \mu)_{\Gamma_I}$$

where

$$(v_1 - v_2, \mu)_{\Gamma_I} = \int_{\Gamma_I} \mu(v_1 - v_2)\, d\Gamma \tag{4}$$

— that is, finding the two displacement fields u_1 and u_2 and the Lagrange multiplier λ which satisfy:

$$J^*(u_1, u_2, \mu) \leq J^*(u_1, u_2, \lambda) \leq J^*(v_1, v_2, \lambda) \tag{5}$$

for any admissible v_1, v_2 and μ. Clearly, the left inequality in (5) implies that $(u_1 - u_2, \mu)_{\Gamma_I} \leq (u_1 - u_2, \lambda)_{\Gamma_I}$, which imposes that $(u_1 - u_2, \mu)_{\Gamma_I} = 0$ for any admissible μ and therefore $u_1 = u_2$ on Γ_I. The right inequality in (5) imposes that $J_1(u_1) + J_2(u_2) \leq J_1(v_1) + J_2(v_2)$ for any pair of admissible functions (v_1, v_2).

This implies that among all admissible pairs (v_1, v_2) which satisfy the continuity condition (3), the pair (u_1, u_2) minimizes the sum of the energy functionals J_1 and J_2 defined respectively on Ω_1 and Ω_2. Therefore, u_1 and u_2 are the restriction of the solution u of the non-partitioned problem (1) to respectively Ω_1 and Ω_2. Indeed, equations (4) and (5) correspond to a hybrid variational principle where the inter-subdomain continuity constraint (3) is removed by the introduction of a Lagrange multiplier (see, for example, Pian [8]).

If now the displacement fields u_1 and u_2 are expressed by suitable shape functions as:

$$u_1 = \mathbf{N}\mathbf{u}_1 \quad \text{and} \quad u_2 = \mathbf{N}\mathbf{u}_2$$

and the continuity equation is enforced for the discrete problem, a standard Galerkin procedure transforms the hybrid variational principle (4) in the following algebraic system:

$$\begin{aligned} \mathbf{K}_1\mathbf{u}_1 &= \mathbf{f}_1 + \mathbf{B}_1^T\boldsymbol{\lambda} \\ \mathbf{K}_2\mathbf{u}_2 &= \mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda} \\ \mathbf{B}_1\mathbf{u}_1 &= \mathbf{B}_2\mathbf{u}_2 \end{aligned} \tag{6}$$

where $\mathbf{K}_j$, $\mathbf{u}_j$, and $\mathbf{f}_j$, $j = 1, 2$, are respectively the stiffness matrix, the displacement vector, and the prescribed force vector associated with the finite element discretization of Ω_j. The vector of Lagrange multipliers $\boldsymbol{\lambda}$ represents the interaction forces between the two subdomains Ω_1 and Ω_2 along their common boundary Γ_I. Within each subdomain Ω_j, we denote the number of interior nodal unknowns by n_j^s and the number of interface nodal unknowns by n_j^I. The total number of interface nodal unknowns is denoted by n_I. Note that $n_I = n_1^I = n_2^I$ in the particular case of two subdomains. If the interior degrees of freedom are numbered first and the interface ones are numbered last, each of the two connectivity matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ takes the form:

$$\mathbf{B}_j = [\mathbf{O}_j \quad \mathbf{I}_j] \quad j = 1, 2$$

where $\mathbf{O}_j$ is an $n_j^I \times n_j^s$ null matrix and $\mathbf{I}_j$ is the $n_j^I \times n_j^I$ identity matrix. The vector of Lagrange multipliers $\boldsymbol{\lambda}$ is n_I long.

If both $\mathbf{K}_1$ and $\mathbf{K}_2$ are non-singular, equations (6) can be written as:

$$
\begin{aligned}
(\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{B}_1^T + \mathbf{B}_2\mathbf{K}_2^{-1}\mathbf{B}_2^T)\boldsymbol{\lambda} &= \mathbf{B}_2\mathbf{K}_2^{-1}\mathbf{f}_2 - \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 \\
\mathbf{u}_1 &= \mathbf{K}_1^{-1}(\mathbf{f}_1 + \mathbf{B}_1^T\boldsymbol{\lambda}) \\
\mathbf{u}_2 &= \mathbf{K}_2^{-1}(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda})
\end{aligned}
\tag{7}
$$

and the solution of (6) is obtained by solving the first of equations (7) for the Lagrange multipliers $\boldsymbol{\lambda}$, then substituting these in the second of (7) and back-solving for $\mathbf{u}_1$ and $\mathbf{u}_2$.

For an arbitrary number of subdomains Ω_j, the method goes as follows. First, the finite element mesh is "torn" into a set of totally disconnected meshes (fig. 2).

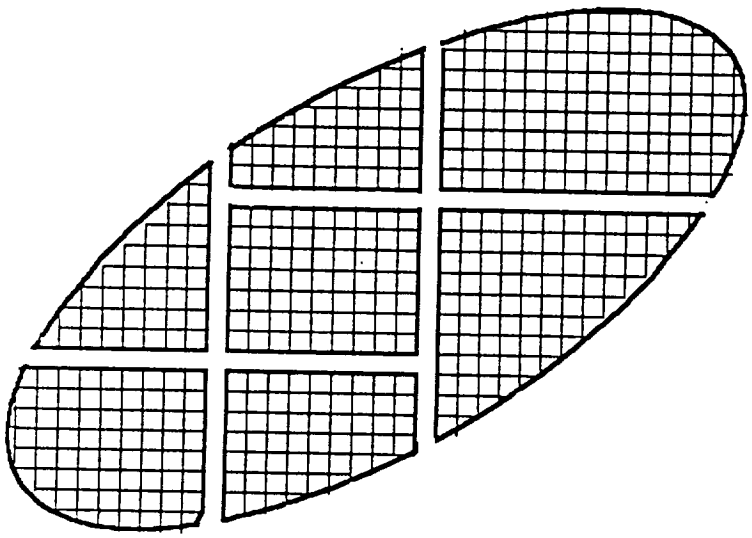


FIG. 2 *Finite Element Tearing*

For each mesh, the stiffness matrix $\mathbf{K}_j$ and the vector of prescribed forces $\mathbf{f}_j$ are formed. Next, for each Ω_j, a set of boolean symbolic matrices $\mathbf{B}_j^k$ are set up to interconnect the mesh of Ω_j with those of its neighbors Ω_k. In general, $\mathbf{B}_j^k$ is $n_I \times (n_j^s + n_j^I)$ and has the following pattern:

$$\mathbf{B}_j^k = \begin{bmatrix} \mathbf{O}_1(j,k) \\ \mathbf{C}_j^k \\ \mathbf{O}_2(j,k) \end{bmatrix}$$

where $\mathbf{O}_1(j,k)$ is an $m_1(j,k) \times (n_j^s + n_j^I)$ zero matrix, $\mathbf{O}_2(j,k)$ is another $m_2(j,k) \times (n_j^s + n_j^I)$ zero matrix and $\mathbf{C}_j^k$ is an $m_c(j,k) \times (n_j^s + n_j^I)$ connectivity matrix, $m_c(j,k)$ is the number of Lagrange multipliers that interconnect Ω_j with its neighbor Ω_k, and $m_1(j,k)$ and $m_2(j,k)$ are two non-negative integers which satisfy $m_1(j,k) + m_c(j,k) + m_2(j,k) = n_I$. The connectivity matrix $\mathbf{C}_j^k$ can be written as:

$$\mathbf{C}_j^k = [\mathbf{O}_3(j,k) \quad \mathbf{I}_j^k \quad \mathbf{O}_4(j,k)]$$

where $\mathbf{O}_3(j,k)$ is an $m_c(j,k) \times m_3(j,k)$ zero matrix, $\mathbf{I}_j^k$ is the $m_c(j,k) \times m_c(j,k)$ identity matrix, $\mathbf{O}_4(j,k)$ is another $m_c(j,k) \times m_4(j,k)$ zero matrix, and $m_3(j,k)$ and $m_4(j,k)$ are two non-negative integers which verify $m_3(j,k) + m_c(j,k) + m_4(j,k) = n_j^s + n_j^I$. If a_j and N_s denote respectively the number of subdomains Ω_k that are adjacent to Ω_j and the total number of subdomains, the finite element variational interpretation of the saddle-point problem (4) generates the following algebraic system:

$$\begin{aligned} \mathbf{K}_j \mathbf{u}_j &= \mathbf{f}_j + \sum_{k=1}^{k=a_j} \mathbf{B}_j^{k\,T} \boldsymbol{\lambda} \qquad j = 1, N_s \\ \mathbf{B}_j^k \mathbf{u}_j &= \mathbf{B}_k^j \mathbf{u}_k \qquad\quad j = 1, N_s \text{ and } \Omega_k \text{ connected to } \Omega_j \end{aligned} \tag{8}$$

If $\mathbf{K}_j$ is non-singular for all $j = 1, N_s$, the solution procedure (7) can be extended to the case of an arbitrary number of subdomains. However, the finite element tearing process described in this section may produce some "floating" subdomains Ω_f which are characterized by a singular stiffness matrix $\mathbf{K}_f$. When this happens, the above solution algorithm (7) breaks down and a special computational strategy is required to handle the local singularities.

We refer to the computational procedure presented herein as the method of finite element tearing and interconnecting because of its resemblance with Kron's tearing method [7] for electric circuit models. We also note that the utility of Lagrange multipliers specifically for domain decomposition has also

been previously recognized by other investigators (Dihn, Glowinsky and Periaux [9], Dorr [10]).

3. Handling local singularities

Again, we focus on the two-subdomain tearing. The extrapolation to $N_s > 2$ is straightforward. For example, suppose that Ω corresponds to a cantilever beam and that Ω_1 and Ω_2 are the result of a vertical partitioning (fig. 3).

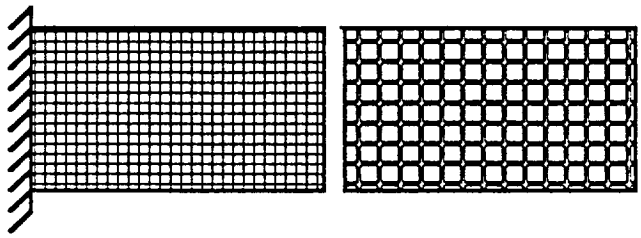


FIG. 3 *Decomposition resulting in a singular subdomain*

In this case, $\mathbf{K}_1$ is positive definite and $\mathbf{K}_2$ is positive semi-definite since no boundary condition is specified over Ω_2. Therefore, the second of equations (6):

$$\mathbf{K}_2\mathbf{u}_2 \;=\; \mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda} \tag{9}$$

requires special attention. If the singular system (9) is consistent, a pseudo-inverse of $\mathbf{K}_2$ can be found, — that is a matrix $\mathbf{K}_2^+$ which verifies $\mathbf{K}_2\mathbf{K}_2^+\mathbf{K}_2 = \mathbf{K}_2$, and the general solution of (9) is given by

$$\mathbf{u}_2 \;=\; \mathbf{K}_2^+(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda}) + \mathbf{R}_2\boldsymbol{\alpha} \tag{10}$$

where $\mathbf{R}_2$ is an $(n_2^s+n_2^I)\times n_2^r$ rectangular matrix whose columns form a basis of the null space of $\mathbf{K}_2$, and $\boldsymbol{\alpha}$ is a vector of length n_2^r. Physically, $\mathbf{R}_2$ represents the rigid body modes of Ω_2 and $\boldsymbol{\alpha}$ specifies a linear combination of these. Consequently, we have $n_2^r \leq 6$ for three-dimensional problems, and $n_2^r \leq 3$ for two-dimensional problems. Substituting (10) into (7) leads to:

$$\begin{aligned}(\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{B}_1^T + \mathbf{B}_2\mathbf{K}_2^+\mathbf{B}_2^T)\boldsymbol{\lambda} \;&=\; -\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 + \mathbf{B}_2(\mathbf{K}_2^+\mathbf{f}_2 + \mathbf{R}_2\boldsymbol{\alpha}) \\ \mathbf{u}_1 \;&=\; \mathbf{K}_1^{-1}(\mathbf{f}_1 + \mathbf{B}_1^T\boldsymbol{\lambda}) \\ \mathbf{u}_2 \;&=\; \mathbf{K}_2^+(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda}) + \mathbf{R}_2\boldsymbol{\alpha}\end{aligned} \tag{11}$$

It should be noted that:

1. because $\mathbf{B}_j$ is a boolean operator, the result of its application to a matrix or vector quantity should be interpreted as an extraction process rather than a matrix-matrix or matrix-vector product. For example, $\mathbf{B}_2\mathbf{R}_2$ is the restriction of the local rigid modes $\mathbf{R}_2$ of Ω_2 to the interface unknowns. In the sequel we adopt the notation:

$$\mathbf{R}_2^I \;=\; \mathbf{B}_2\mathbf{R}_2$$

2. the pseudo-inverse $\mathbf{K}_2^+$ does not need to be explicitly computed. For a given input vector $\mathbf{v}$, the output vector $\mathbf{K}_2^+\mathbf{v}$ and the rigid modes $\mathbf{R}_2$ can be obtained at almost the same computational cost as the response vector $\mathbf{K}_1^{-1}\mathbf{v}$, where $\mathbf{K}_1$ is non-singular (see *appendix A*).

3. system (11) is under-determined. Both $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ need to be determined before $\mathbf{u}_1$ and $\mathbf{u}_2$ can be found, but only three equations are available so far.

Since $\mathbf{K}_2$ is symmetric, the singular equation (9) admits at least one solution if and only if the right hand side $(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda})$ has no component in the null space of $\mathbf{K}_2$. This can be expressed as:

$$\mathbf{R}_2^T(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda}) \;=\; 0 \tag{12}$$

The above orthogonality condition provides the missing equation for the complete solution of (11). Combining (11) and (12) yields after some algebraic manipulations:

$$\begin{bmatrix} \mathbf{F}_I & -\mathbf{R}_2^I \\ -\mathbf{R}_2^{IT} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_2\mathbf{K}_2^+\mathbf{f}_2 - \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 \\ -\mathbf{R}_2^T\mathbf{f}_2 \end{bmatrix}$$

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{K}_1^{-1}(\mathbf{f}_1 + \mathbf{B}_1^T\boldsymbol{\lambda}) \\ \mathbf{u}_2 &= \mathbf{K}_2^+(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda}) + \mathbf{R}_2\boldsymbol{\alpha} \end{aligned} \tag{13}$$

where

$$\mathbf{F}_I = (\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{B}_1^T + \mathbf{B}_2\mathbf{K}_2^+\mathbf{B}_2^T)$$

Clearly, $\mathbf{F}_I$ is symmetric positive definite and $\mathbf{R}_2^I$ has full column rank. Therefore, the system of equations in $(\boldsymbol{\lambda}, \boldsymbol{\alpha})$ is symmetric and non-singular. It admits a unique solution $(\boldsymbol{\lambda}, \boldsymbol{\alpha})$ which uniquely determines $\mathbf{u}_1$ and $\mathbf{u}_2$.

It is important to note that since $n_2^r \leq 6$, systems (13) and (7) have almost the same size. For an arbitrary number of subdomains N_s of which N_f are floating, the additional number of equations introduced by the handling of local singularities is bounded by $6N_f$. For large-scale problems and relatively coarse mesh partitions, this number is a very small fraction of the size of the global system. On the other hand, if a given tearing process does not result in any floating subdomain, $\boldsymbol{\alpha}$ is zero and the systems of equations (13) and (7) are identical.

Next, we present a numerical algorithm for the solution of (13).

4. A preconditioned conjugate projected gradient algorithm

Here we focus on the solution of the non-singular system of equations:

$$\begin{bmatrix} \mathbf{F}_I & -\mathbf{R}_2^I \\ -\mathbf{R}_2^{IT} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_2\mathbf{K}_2^+\mathbf{f}_2 - \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 \\ -\mathbf{R}_2^T\mathbf{f}_2 \end{bmatrix} \tag{14}$$

where

$$\mathbf{F}_I = \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{B}_1^T + \mathbf{B}_2\mathbf{K}_2^+\mathbf{B}_2^T$$

We seek an efficient solution algorithm that does not require the explicit assembly of $\mathbf{F}_I$.

The solution to the above problem can be expressed as:

$$\begin{aligned} \boldsymbol{\lambda} &= -\mathbf{H}(\mathbf{B}_2\mathbf{K}_2^{+}\mathbf{f}_2 - \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1) + \mathbf{T}\mathbf{R}_2^{+}\mathbf{f}_2 \\ \boldsymbol{\alpha} &= \mathbf{T}^T(\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 - \mathbf{B}_2\mathbf{K}_2^{+}\mathbf{f}_2) - \mathbf{U}\mathbf{R}_2^T\mathbf{f}_2 \end{aligned}$$

where

$$\begin{aligned} \mathbf{H} &= \mathbf{F}_I^{-1} - \mathbf{F}_I^{-1}\mathbf{R}_2^I\mathbf{U}^{-1}\mathbf{R}_2^{I^T}\mathbf{F}_I^{-1} \\ \mathbf{T} &= \mathbf{F}_I^{-1}\mathbf{R}_2^I\mathbf{U}^{-1} \\ \mathbf{U} &= -\mathbf{R}_2^{I^T}\mathbf{F}_I^{-1}\mathbf{R}_2^I \end{aligned} \tag{15}$$

As written in (15), this solution procedure is not recommended because it requires either the evaluation of the inverse of $\mathbf{F}_I$, or the nested solutions of two linear systems involving $\mathbf{F}_I$ and $\mathbf{R}_2^{I^T}\mathbf{F}_I^{-1}\mathbf{R}_2^I$. It is noted by Fletcher [11] that if two matrices $\mathbf{S}$ and $\mathbf{Z}$ are computed such that:

$$\begin{aligned} \mathbf{S}^T\mathbf{R}_2^I &= \mathbf{I} \\ \mathbf{Z}^T\mathbf{R}_2^I &= \mathbf{O} \end{aligned} \tag{16}$$

an alternative representation of the solution to (14) is given by:

$$\begin{aligned} \boldsymbol{\lambda} &= -\mathbf{H}(\mathbf{B}_2\mathbf{K}_2^{+}\mathbf{f}_2 - \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1) + \mathbf{T}\mathbf{R}_2^{+}\mathbf{f}_2 \\ \boldsymbol{\alpha} &= \mathbf{T}^T(\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 - \mathbf{B}_2\mathbf{K}_2^{+}\mathbf{f}_2) - \mathbf{U}\mathbf{R}_2^T\mathbf{f}_2 \end{aligned}$$

where

$$\begin{aligned} \mathbf{H} &= \mathbf{Z}(\mathbf{Z}^T\mathbf{F}_I\mathbf{Z})^{-1}\mathbf{Z}^T \\ \mathbf{T} &= \mathbf{S} - \mathbf{H}\mathbf{F}_I\mathbf{S} \\ \mathbf{U} &= \mathbf{S}^T\mathbf{F}_I\mathbf{H}\mathbf{F}_I\mathbf{S} - \mathbf{S}^T\mathbf{F}_I\mathbf{S} \end{aligned} \tag{17}$$

which does not require the explicit assembly of $\mathbf{F}_I$ if a suitable iterative scheme is chosen for solving all the temporary systems involving the quantity $(\mathbf{Z}^T\mathbf{F}_I\mathbf{Z})^{-1}$. Still, the above solution procedure is not feasible because it requires the computation of $\mathbf{S}$ and $\mathbf{Z}$ — typically via a $\mathbf{QR}$ factorization of some matrix involving

$\mathbf{R}_2^I$ [11], and the iterative solution of too many temporary systems before $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ can be obtained.

Clearly, the nature of $\mathbf{F}_I$ makes the solution of (14) inadequate by any technique which requires this submatrix explicitly. This implies that a direct method or an iterative method of the SOR type cannot be used. The only efficient method of solving (14) in the general sparse case is that of conjugate gradients, because once $\mathbf{K}_1$ and $\mathbf{K}_2$ have been factorized, matrix-vector products of the form $\mathbf{F}_I\mathbf{v}$ can be performed very efficiently using only forward and backward substitutions. Unfortunately, the Lagrangian matrix:

$$\mathbf{L} = \begin{bmatrix} \mathbf{F}_I & -\mathbf{R}_2^I \\ -\mathbf{R}_2^{I^T} & \mathbf{O} \end{bmatrix}$$

is indefinite so that a straightforward conjugate gradient algorithm cannot be directly applied to the solution of (14). However, the conjugate gradient iteration with the *projected* gradient (see, for example, Gill and Murray [12]) can be used to obtain the sought-after solution. In order to introduce the latter solution algorithm, we first note that solving (14) is equivalent to solving the equality constraint problem:

$$\begin{aligned} \textit{minimize} \quad & \Phi(\boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{\lambda}^T\mathbf{F}_I\boldsymbol{\lambda} + (\mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 - \mathbf{B}_2\mathbf{K}_2^{+}\mathbf{f}_2)^T\boldsymbol{\lambda} \\ \textit{subject to} \quad & \mathbf{R}_2^{I^T}\boldsymbol{\lambda} = \mathbf{R}_2^T\mathbf{f}_2 \end{aligned} \tag{18}$$

Since $\mathbf{F}_I$ is symmetric positive definite, a conjugate gradient algorithm is most suitable for computing the unique solution to the unconstrained problem. Therefore, this algorithm will converge to the solution to (18) if and only if it can be modified so that the constraint $\mathbf{R}_2^{I^T}\boldsymbol{\lambda} = \mathbf{R}_2^T\mathbf{f}_2$ is satisfied at each iteration. This can be achieved by projecting all the search directions onto the null space of $\mathbf{R}_2^I$.

The result is a conjugate gradient algorithm with the projected gradient. It is of the form:

Initialize

Pick $\boldsymbol{\lambda}^{(0)}$ such that $\mathbf{R}_2^{I^T}\boldsymbol{\lambda}^{(0)} = \mathbf{R}_2^T\mathbf{f}_2$

$$\mathbf{r}^{(0)} = (\mathbf{B}_2\mathbf{K}_2^{+}\mathbf{f}_2 - \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1) \tag{19}$$

Iterate k = 1,2, ... until convergence

$$\begin{aligned}
\beta^{(k)} &= \mathbf{r}^{(k-1)^T}\mathbf{r}^{(k-1)}/\mathbf{r}^{(k-2)^T}\mathbf{r}^{(k-2)} \quad (\beta^{(1)} = 0) \\
\mathbf{s}^{(k)} &= \mathbf{r}^{(k-1)} + \beta^{(k)}\mathbf{s}^{(k-1)} \quad (\mathbf{s}^{(1)} = \mathbf{r}^{(0)}) \\
\mathbf{s}^{(k)} &= [\mathbf{I} - \mathbf{R}_2^I(\mathbf{R}_2^{I^T}\mathbf{R}_2^I)^{-1}\mathbf{R}_2^{I^T}]\mathbf{s}^{(k)} \\
\gamma^{(k)} &= \mathbf{r}^{(k-1)^T}\mathbf{r}^{(k-1)}/\mathbf{s}^{(k)^T}\mathbf{F}_I\mathbf{s}^{(k)} \\
\boldsymbol{\lambda}^{(k)} &= \boldsymbol{\lambda}^{(k-1)} + \gamma^{(k)}\mathbf{s}^{(k)} \\
\mathbf{r}^{(k)} &= \mathbf{r}^{(k-1)} - \gamma^{(k)}\mathbf{F}_I\mathbf{s}^{(k)}
\end{aligned} \tag{20}$$

A fast scheme for finding a starting $\boldsymbol{\lambda}^{(0)}$ which satisfies the constraint $\mathbf{R}_2^{I^T}\boldsymbol{\lambda}^{(0)} = \mathbf{R}_2^T\mathbf{f}_2$ is given in *appendix B*. Clearly, $\mathbf{R}_2^{I^T}\mathbf{s}^{(k)} = 0$ for all $k \geq 1$. Therefore, $\mathbf{R}_2^{I^T}\boldsymbol{\lambda}^{(k)} = \mathbf{R}_2^{I^T}\boldsymbol{\lambda}^{(0)}$ which indicates that the approximate solution $\boldsymbol{\lambda}^{(k)}$ satisfies the linear equality constraint of problem (14) at each iteration k. It is also important to note that within each iteration, only one projection is performed. This projection is relatively inexpensive since the only implicit computations that are involved are associated with the matrix $\mathbf{R}_2^{I^T}\mathbf{R}_2^I$ which is at most 6×6. This matrix is factored once, before the first iteration begins. Except for this small overhead, algorithm (20) above has the same computational cost as the regular conjugate gradient method.

After $\boldsymbol{\lambda}$ is found, the rigid body mode coefficients are computed as:

$$\boldsymbol{\alpha} = (\mathbf{R}_2^{I^T}\mathbf{R}_2^I)^{-1}(\mathbf{F}_I\boldsymbol{\lambda} - \mathbf{B}_2\mathbf{K}_2^{-1}\mathbf{f}_2 + \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1)$$

For an arbitrary number of subdomains N_s of which N_f are floating, the equality constraint is:

$$\mathbf{R}^{I^T}\boldsymbol{\lambda} = \begin{bmatrix} \mathbf{R}_1^{I^T} \\ \cdots \\ \mathbf{R}_f^{I^T} \end{bmatrix} \boldsymbol{\lambda} = \mathbf{R}^T\mathbf{f} = \begin{bmatrix} \mathbf{R}_1{}^T & 0 & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & \mathbf{R}_f{}^T \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \cdots \\ \mathbf{f}_f \end{bmatrix}$$

Only those columns of $\mathbf{R}_j^T$ which operate on Lagrange multipliers that are associated with $\Gamma_I \cap \Omega_j$ are non-zero. The projection matrix is $\mathbf{P} = [\mathbf{I} - \mathbf{R}^I(\mathbf{R}^{I^T}\mathbf{R}^I)^{-1}\mathbf{R}^{I^T}]$ where $\mathbf{R}^{I^T}\mathbf{R}^I$ is generally banded of dimension at most equal to $6N_f$. The banded structure of $\mathbf{P}$ is determined by the subdomains interconnectivity. If for practical reasons this banded structure is not exploited, the number of three-dimensional floating subdomains should be kept as small as possible, say less than thirty two, which implies that the proposed computational method would be suitable only for coarse and medium grain multiprocessors.

5. Preconditioning the interface problem

As in the case of the conjugate gradient method, the conjugate projected gradient algorithm is most effective when applied to the preconditioned system of equations. It should be noted that even in the presence of floating subdomains, only $\mathbf{F}_I$ needs to be preconditioned and not the global Lagrangian matrix $\mathbf{L}$. In the case of two subdomains, $\mathbf{F}_I$ can be written in matrix form as:

$$\mathbf{F}_I = [\mathbf{B}_1 \quad \mathbf{B}_2] \begin{bmatrix} \mathbf{K}_1^{-1} & \mathbf{O} \\ \mathbf{O} & \mathbf{K}_2^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{B}_1^T \\ \mathbf{B}_2^T \end{bmatrix} \tag{21}$$

where $\mathbf{K}_j^{-1}$, $j = 1, 2$, is replaced by $\mathbf{K}_j^+$ if Ω_j is a floating subdomain. The objective is to find an approximate inverse $\mathbf{P}_I^{-1}$ of $\mathbf{F}_I$ that: (a) does not need to be explicitly assembled (especially since $\mathbf{F}_I$ is not explicitly assembled), and (b) that is amenable to parallel computations. The matrix $\mathbf{P}$ is then the preconditioner. Equation (21) above suggests the following choice for $\mathbf{P}_I^{-1}$:

$$\mathbf{P}_I^{-1} = [\mathbf{B}_1 \quad \mathbf{B}_2] \begin{bmatrix} \mathbf{K}_1 & \mathbf{O} \\ \mathbf{O} & \mathbf{K}_2 \end{bmatrix} \begin{bmatrix} \mathbf{B}_1^T \\ \mathbf{B}_2^T \end{bmatrix} \tag{22}$$

At each iteration k, the preconditioned conjugate projected gradient algorithm involves the solution of an auxiliary system of the form:

$$\mathbf{P}_I \mathbf{z}^{(k)} = \mathbf{r}^{(k)} \tag{23}$$

where $\mathbf{r}^{(k)}$ is the residual at the $k-th$ iteration. The particular choice of $\mathbf{P}_I^{-1}$ given in (22) offers the advantage of solving (23) explicitly without the need for any intermediate factorization.

For computational efficiency, $\mathbf{P}_I^{-1}$ is implemented as:

$$\mathbf{P}_I^{-1} = \mathbf{K}_1^I + \mathbf{K}_2^I \tag{24}$$

where $\mathbf{K}_1^I$ and $\mathbf{K}_2^I$ are the traces of $\mathbf{K}_1$ and $\mathbf{K}_2$ on Γ_I. Clearly, with this choice for the preconditioner, the auxiliary system (23) is "cheap", easy to solve and perfectly parallelizable on both local and shared memory parallel architectures.

Since we do not have a strong mathematical justification for this choice of the preconditioner, we have conducted a set of numerical experiments to assess *a priori* its performance. A fixed-fixed cylindrical panel was discretized with an N by M regular mesh and was modeled with 4 node shell elements (fig. 4). All test cases used $N_s = 2$ and a vertical slicing.

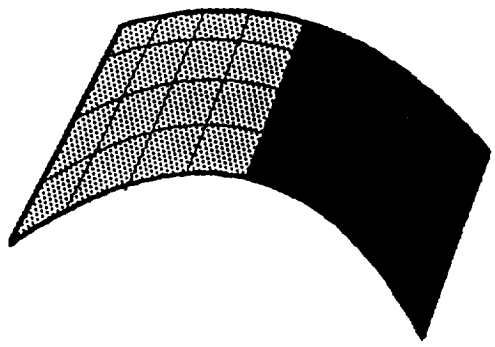


FIG. 4 *Cylindrical panel* - $N_s = 2$

Table 1 shown below reports the condition numbers of the global stiffness matrix $\mathbf{K}$, the subdomain stiffness matrices $\mathbf{K}_1$ and $\mathbf{K}_2$, and the original and preconditioned interface flexibility matrices $\mathbf{F}_I$ and $\mathbf{P}_I^{-1}\mathbf{F}_I$, for various values of N.

TABLE 1

Condition numbers

Cylindrical panel - N by M mesh - shell elements - 2 subdomains

N	M	$\kappa(\mathbf{K})$	$\kappa(\mathbf{K}_1)$	$\kappa(\mathbf{K}_2)$	$\kappa(\mathbf{F}_I)$	$\kappa(\mathbf{P}_I^{-1}\mathbf{F}_I)$
10	5	$2.5\ 10^4$	$5.6\ 10^3$	$5.6\ 10^3$	$1.4\ 10^4$	$4.9\ 10^2$
20	10	$3.4\ 10^5$	$2.1\ 10^4$	$2.1\ 10^4$	$2.8\ 10^4$	$3.8\ 10^3$
40	20	$5.4\ 10^6$	$9.1\ 10^4$	$9.1\ 10^4$	$1.2\ 10^5$	$3.1\ 10^4$

For this test problem, the condition number of the preconditioned interface is two order of magnitude lower than that of the global problem.

The extrapolation of (22) and (24) to $N_s > 2$ is straightforward. In order to reduce furthermore the number of preconditioned conjugate projected gradient iterations, the selective reorthogonalization procedure developed by Roux and reported in [13] is also utilized.

6. Parallel characteristics of the proposed method

Like most domain decomposition based algorithms, the proposed method of finite element tearing and interconnecting is perfectly suitable for parallel processing. If every subdomain Ω_j is assigned to an individual processor p_j, all local finite element computations can be performed in parallel. These include forming and assembling the stiffness matrix $\mathbf{K}_j$ and the forcing vector $\mathbf{f}_j$, factoring $\mathbf{K}_j$ and eventually computing the rigid modes $\mathbf{R}_j$, as well as backsolving for $\mathbf{u}_j$ after $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ have been determined. The conjugate projected gradient algorithm described in Section 4 is also amenable to parallel processing. For example, the matrix-vector product $\mathbf{F}_I\mathbf{s}^{(k)}$ can be computed in parallel by assigning to each processor p_j the task of evaluating $\mathbf{y}_j^{(k)} = \mathbf{B}_j^k\mathbf{K}_j^{-1}\mathbf{B}_j^{kT}\mathbf{s}_j^{(k)}$, and exchanging $\mathbf{y}_j^{(k)}$ with the

processors assigned to neighboring subdomains in order to assemble the global result. Interprocessor communication is required only during the solution of the interface problem (14) and takes place exclusively among neighboring processors during the assembly of the subdomain results.

At this point, we stress that the parallel solution method developed herein requires inherently less interprocessor communication than other domain decomposition based parallel algorithms. As mentioned earlier, interprocessor communication within the proposed method occurs only during the solution of the interface problem (14). The reader should trace back this problem as well as the presence of the Lagrange multipliers to the integral quantity:

$$(v_i - v_j, \lambda)_{\Gamma_{I_{ij}}} = \int_{\Gamma_{I_{ij}}} \lambda(v_i - v_j)\, d\Gamma \tag{25}$$

where $\Gamma_{I_{ij}}$ is the interface between subdomains Ω_i and Ω_j. If $\Gamma_{I_{ij}}$ has a zero measure, then $(v_i - v_j, \lambda)_{\Gamma_{I_{ij}}} = 0$ and no exchange of information is needed between Ω_i and Ω_j. Therefore the subdomains which interconnect along one edge in three-dimensional problems and those which interconnect along one vertex in both two and three-dimensional problems do not require any interprocessor communication. This is unlike the parallel method of substructures, whether the interface problem is solved with a direct scheme [1] or with an iterative one [2]. For a three-dimensional regular mesh that is partitioned into subcubes, the proposed method of finite element tearing and interconnecting requires that each subdomain communicate with at most six neighboring subdomains (since a cube has only six faces), while the parallel method of substructures necessitates that each subdomain communicate with up to 26 neighbors (fig. 5). This communication characteristic makes the proposed parallel solution method very attractive for a multiprocessor with a distributed memory such as a hypercube. Indeed, the advantages of the method for this family of parallel processors are two folds: (a) the number of message-passing is dramatically reduced, which reduces the overhead due to communication start-up, and (b) the complexity of the communication requirements is improved so that an optimal mapping of the processors onto the subdomains can be reached (Bokhari [14], Farhat [15]); therefore the elapsed time for a given message is improved. Both enhancements (a) and (b) reduce the communication overhead of the parallel solution algorithm in a synergistic manner. This algorithmic feature of the proposed method is still desirable for shared memory multiprocessors because it eases the assembly process during

the interface solution and makes the latter more manageable. It is not however as critical for the performance as it is for local memory multiprocessors.

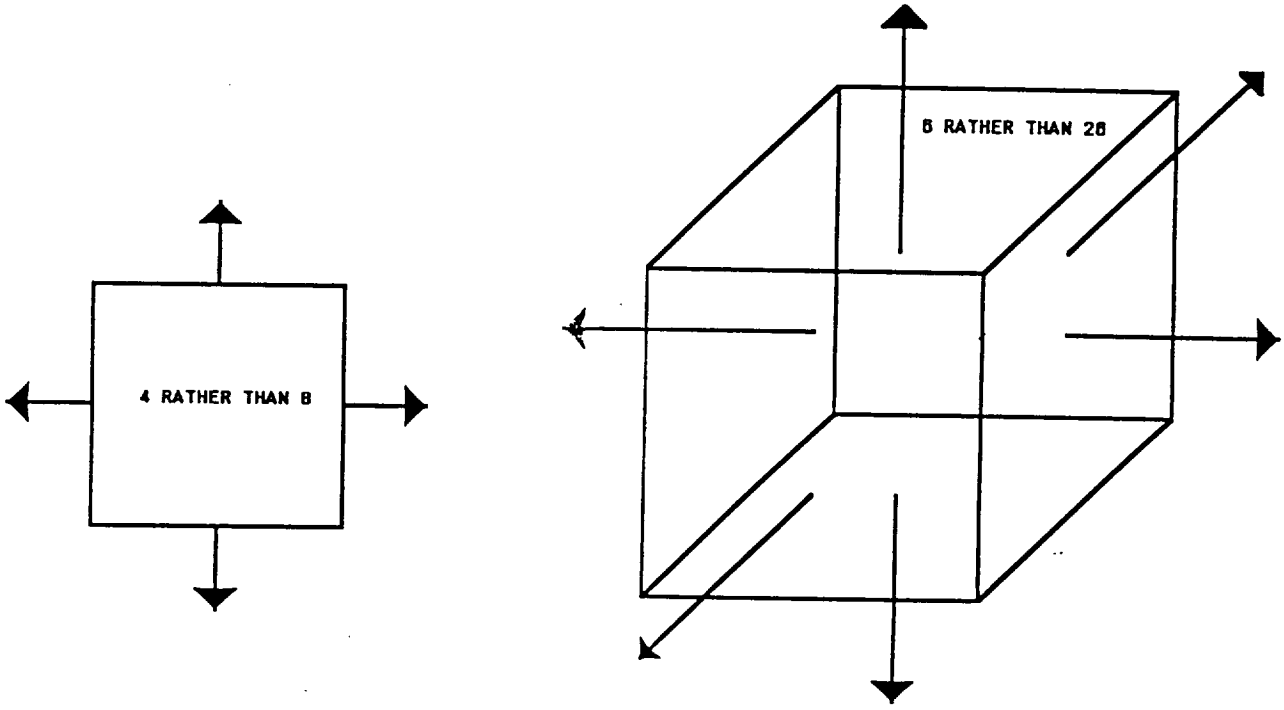


FIG. 5 *Reduced interprocessor communication patterns for two and three-dimensional regular mesh partitions*

7. Tearing vs. substructuring

Another difference between the subdomain based parallel solution method developed in this paper and the parallel method of substructures lies in the formulation of the interface problem. For the method of substructures, the interface problem corresponds to a *stiffness* formulation. For the two-subdomain decomposition it can be written as:

$$(\mathbf{K}_{II} - \mathbf{K}_{1I}^T \mathbf{K}_{11}^{-1} \mathbf{K}_{1I} - \mathbf{K}_{2I}^T \mathbf{K}_{22}^{-1} \mathbf{K}_{2I}) \mathbf{u}_I = \mathbf{f}_{II} - \mathbf{K}_{1I}^T \mathbf{K}_{11}^{-1} \mathbf{f}_{11} - \mathbf{K}_{2I}^T \mathbf{K}_{22}^{-1} \mathbf{f}_{22} \quad (26)$$

where $\mathbf{K}_{II}$, $\mathbf{K}_{11}$ and $\mathbf{K}_{22}$ are the stiffness matrices associated respectively with the interface nodes and the interior nodes of subdomains Ω_1 and Ω_2, and $\mathbf{K}_{1I}$ and

$\mathbf{K}_{2I}$ are the coupling stiffnesses between respectively Ω_1 and Γ_I and Ω_2 and Γ_I (see, for example [1] for further details). A standard conjugate gradient algorithm may be used for solving (26). On the other hand, the resulting interface problem for the method of finite element tearing and interconnecting corresponds to a *flexibility* formulation. For the two-subdomain decomposition, it can be written as in (14) and necessitates the use of a conjugate projected gradient algorithm for finding the solution $\boldsymbol{\lambda}$.

If $\mathbf{K}_1$ and $\mathbf{K}_2$ are partitioned into internal and boundary (interface) components and then are injected into the first of equations (7), it can be easily shown that:

$$\begin{aligned} \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{B}_1^T &= (\mathbf{K}_{II}^{(1)} - \mathbf{K}_{1I}^T\mathbf{K}_1^{-1}\mathbf{K}_{1I})^{-1} \\ \mathbf{B}_2\mathbf{K}_2^{-1}\mathbf{B}_2^T &= (\mathbf{K}_{II}^{(2)} - \mathbf{K}_{2I}^T\mathbf{K}_2^{-1}\mathbf{K}_{2I})^{-1} \end{aligned} \tag{27}$$

where $\mathbf{K}_{II}^{(1)}$ and $\mathbf{K}_{II}^{(2)}$ denote respectively the contributions of the first and second subdomains to $\mathbf{K}_{II}$. Equations (27) above establish the relationship between both approaches to domain decomposition.

The computational implications of the differences between the two solution methods are as follows:

- within each iteration, the solution process of problem (14) requires an additional computational step which corresponds to the projection of the search direction onto the null space of $\mathbf{R}_2^I$.

- within each iteration, the solution process of problem (14) requires the evaluation of the matrix-vector product $\mathbf{B}_j\mathbf{K}_j^{-1}\mathbf{B}_j^T\mathbf{s}^{(k)}$, while the solution process of problem (26) requires the evaluation of the matrix-vector product $\mathbf{K}_{jI}^T\mathbf{K}_{jj}^{-1}\mathbf{K}_{jI}\mathbf{s}^{(k)}$. Given that $\mathbf{B}_j$ is a boolean matrix and that its application to a matrix or a vector defines a floating-point-free extraction process, each conjugate gradient iteration applied to (14) is less computationally intensive than its counterpart that is applied to (26).

- since a conjugate gradient algorithm captures initially the high frequency mesh mode of a problem, it can be expected to perform better on a flexibility matrix than on a stiffness matrix because the high frequencies of the former are indeed the low frequencies of the stiffness matrix which are closer to the solution of the static problem.

In the light of the above remarks, it is reasonable to expect that for a given mesh partition:

- each conjugate projected gradient iteration that is applied to the solution of the interface problem (14) which results from the method of finite element tearing and interconnecting will not be slower — and may be even faster for large-scale problems and a small number of interface nodes, than each conjugate gradient iteration applied to the solution of the interface problem (26) which results from the method of substructures.

- the iterative solution of the interface problem associated with the tearing method will exhibit a faster rate of convergence than the iterative solution of the interface problem resulting from the conventional method of substructures.

Finally, it should be noted that domain decomposition methods in general exhibit a larger degree of parallelism than parallel direct solvers. The efficiency of the latter is governed by the bandwidth of the given finite element system of equations. If the bandwidth is not large enough, interprocessor communication and/or process synchronization can dominate the work done in parallel by each processor. This is true not only for multiprocessors with a message-passing system, but also for super-vector-multiprocessors with a shared memory such as the CRAY systems, where synchronization primitives are rather expensive. Therefore, the computational method described in this paper should be seriously considered for large-scale problems with a relatively small or medium bandwidth. These problems are typically encountered in the finite element analysis of large space structures which are often elongated and include only a few elements along one or two directions (Farhat [16]). The method is also recommended for problems where the storage requirements of direct solvers cannot be met.

8. Optimal mesh decomposition

The computational method described in this paper requires that the given finite element mesh be partitioned into as many submeshes as there are available processors. In this section, we establish some guidelines for the design of an optimal mesh partition by analyzing the effect of its structure on the performance of the global solution algorithm.

From the numerical point of view, the proposed solution method is hybrid in the sense it combines a direct and an iterative schemes. The direct solver is applied to each subdomain problem, the iterative one to the interface between these

subdomains. If the mesh partition is such that the bandwidth of each subdomain problem is of the same order as the bandwidth of the global unpartitioned system of equations, the overall algorithm performs more operations than a direct method applied to the global unpartitioned system, independently of how fast the interface problem converges. The slicing of a parallelepiped along its largest dimension yields such a partition (fig. 6). If on the other hand the same parallelepiped is partitioned such that the bandwidth of each subdomain problem is much smaller than the bandwidth of the original finite element system (fig. 7), and if the convergence of the interface problem is fast enough, the method of finite element tearing and interconnecting may produce the solution with fewer computations than a global direct solver.

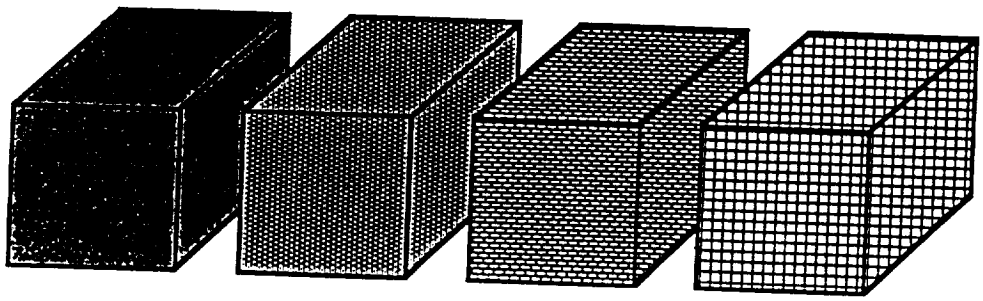


FIG. 6 *Stripwise partitioning of a parallelepiped*

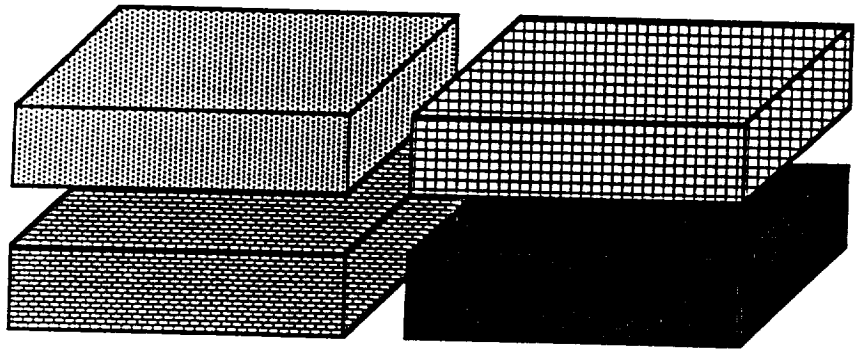


FIG. 7 *Boxwise partitioning of a parallelepiped*

Besides conditioning, there are two other factors which affect the convergence of the interface problem (14) and which are directly related to the mesh partition:

(a) the number of interface nodes, and (b) the interconnectivity of the subdomains along their interface. It can be easily checked that within one iteration of the conjugate projected gradient algorithm, a new information that is issued from a subdomain Ω_j reaches only those subdomains that interconnect with Ω_j along an edge or a plane. Therefore, the interface problem converges faster for a mesh partition that is characterized by a larger effective interconnectivity bandwidth.

The above observations suggest that an automatic finite element mesh decomposer that is suitable for the computational method described herein should meet or strike a balanced compromise between the seven following requirements:

1. it should yield a set of subdomains where the bandwidth of each local problem is only a fraction of the bandwidth of the global system of equations;
2. it should keep the amount of interface nodes as small as possible in order to reduce the size of the interface problem;
3. it should yield a set of subdomains with a relatively high interconnectivity bandwidth so that within each iteration a new correction reaches as many subdomains as possible;
4. it should avoid producing subdomains with "bad" aspect ratio (for example, elongated and flat subdomains) in order to keep the local problems as well-conditioned as possible;
5. it should deliver as few as possible floating subdomains in order to keep the cost associated with the projected gradients as low as possible;
6. it should yield a set of balanced subdomains in order to ensure that the overall computational load will be as evenly distributed as possible among the processors;
7. it should be able to handle irregular geometry and arbitrary discretization in order to be general purpose.

For some mesh topologies, it becomes very difficult to meet simultaneously requirements (1), (2) and (4). In that case, priority should be given to the first two requirements. However, we have found that for many problems, the above requirements can be met, using for example a slightly modified version of the general purpose finite element decomposer presented by Farhat in [17]. Several decomposition examples are described in Section 9. The most challenging problem that is yet to be resolved is the rational relationship between the mesh decomposition and the interface conditioning.

9. Applications and performance assessment

We first illustrate the proposed parallel computational method with the static analysis on a 32 processor iPSC/2 hypercube of a three-dimensional mechanical joint subjected to internal pressure loading. We report performance results which show that the parallel method of tearing exhibits a better speed-up than the parallel method of conventional substructuring because it consumes three times less interprocessor communication. Next, we apply our algorithm to the large-scale finite element analysis on a 4 processor CRAY-2 of a three-dimensional cantilever composite beam made of more than one hundred stiff carbon fibers bound by a nearly incompressible elastomer matrix. We report and discuss in details the measured performance results for various mesh partitioning strategies. For that problem, the proposed solution method outperforms the direct Choleski factorization by a factor greater than three, even for configurations that yield very ill-conditioned systems. In the following, NP, NE, NDF, T_{msg}, T_p and SP denote respectively the number of processors, the number of elements, the number of degrees of freedom, the time elapsed in message-passing, the total parallel time and the overall parallel speed-up.

The finite element discretization of the mechanical joint using 8 node brick elements is shown in figure 8. Two meshes are considered. The first one contains 5002 elements, 14932 degrees of freedom and is intended for a 16 processor cluster of the iPSC/2. The second mesh has 9912 elements, 29654 degrees of freedom and is constructed for a 32 processor configuration of the same hypercube. The mesh decompositions into 16 and 32 subdomains are carefully designed to be topologically equivalent as much as possible to a checkerboard partitioning.

Consequently, many of the resulting subdomains are floating.

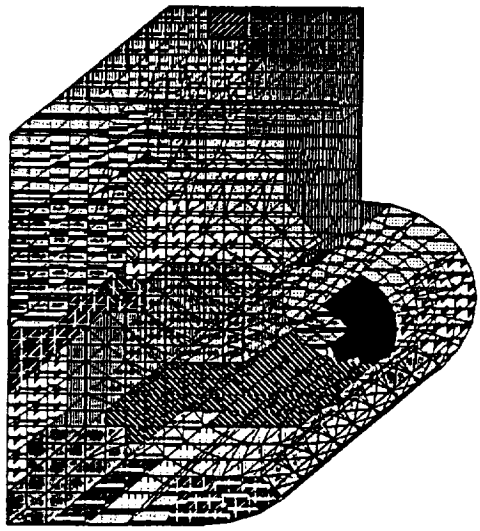


FIG. 8 *Finite element discretization of a mechanical joint*

The interprocessor communication time per iteration, the total parallel execution time, and the overall parallel speed-up associated with the parallel method of tearing and the parallel method of substructures are reported in table 2 for both meshes. For all cases, a tolerance of 10^{-3} on the global relative residuals is selected as a convergence criterion.

TABLE 2

Performance results on iPSC/2

Mechanical joint - brick elements - 16 and 32 subdomains

NP	NE	NDF	$T_{msg}/itr.$ subs.	$T_{msg}/itr.$ tearing	T_p subs.	T_p tearing	SP subs.	SP tearing
16	5002	14932	16.3 m.s.	5.2 m.s.	602 s.	546 s.	14.4	15.4
32	9912	29654	17.9 m.s.	5.4 m.s.	1103 s.	917 s.	24.0	28.8

For both cases, the parallel tearing and parallel substructuring algorithms achieve excellent speed-up. This is generally true for all balanced algorithms that require message-passing only between neighboring processors. However, for this problem, the tearing algorithm is faster and exhibits a 20 % higher speed-up than the conventional substructuring algorithm for which the time elapsed in interprocessor communication is 3.31 times higher. Again, because it avoids interprocessor communication along the edges and corners of the subdomains, the tearing algorithm requires fewer message-passing startups which, in the case of short messages, are known to account for the largest portion of the time elapsed in interprocessor communication on the iPSC/2 (see, for example, the benchmarks of Boman and Rose [18]). A performance comparison with a parallel direct solver is not provided because of the lack of memory space to store in-core the triangular factors of K.

Now that the parallel properties of the presented algorithm have been illustrated, we focus next on example problems that illustrate its intrinsic properties and performance. We consider the large-scale finite element static analysis of the pure bending of a set of beams made of similar jointed composite "pencils" (fig. 9). Each composite pencil contains one carbon fiber with its elastomer matrix and is discretized in 51 vertical layers containing each 25 mesh points. The cross section of the finite element mesh corresponding to a 16 pencil beam is shown in figure 10.

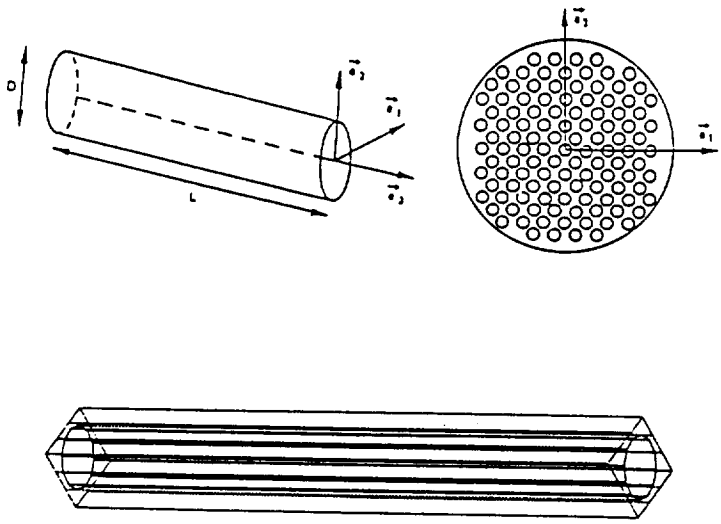


FIG. 9 *A composite beam and a composite pencil*

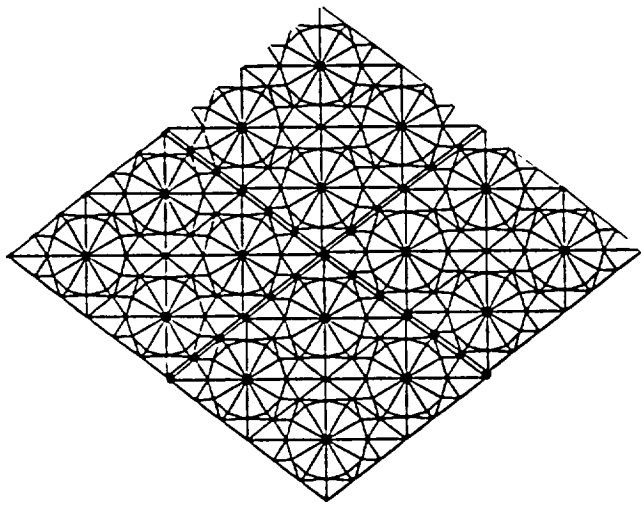


FIG. 10 *Cross section of the finite element mesh for a 16 pencil composite beam*

The numerical results obtained on a 4 processor CRAY-2 for a 16 pencil beam with 48000 degrees of freedom are reported in figures (11-12). These correspond to two extreme mesh decompostions, namely: a horizontal cross-slicing into 4 subdomains each containing 4 cantilever parallel pencils ($D1$), and (b) a vertical slicing into 4 subdomains of which three are floating ($D2$). Poisson's ratio for the

elastomer is 0.49.

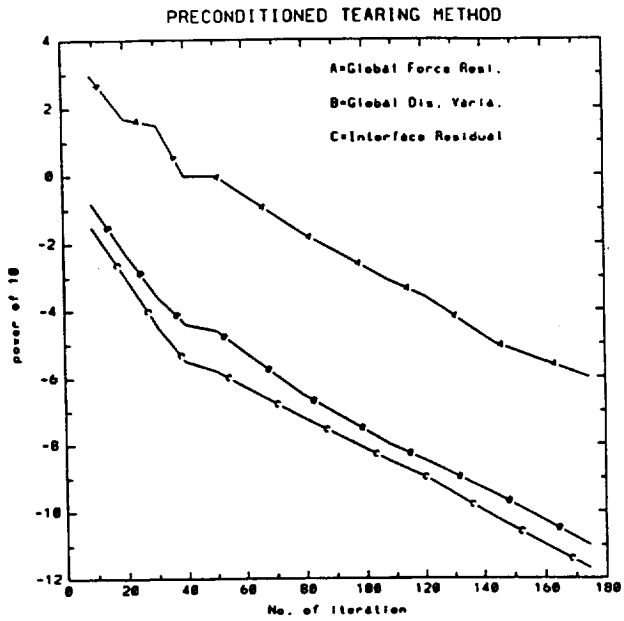


FIG. 11 *Numerical results for decomposition D1*

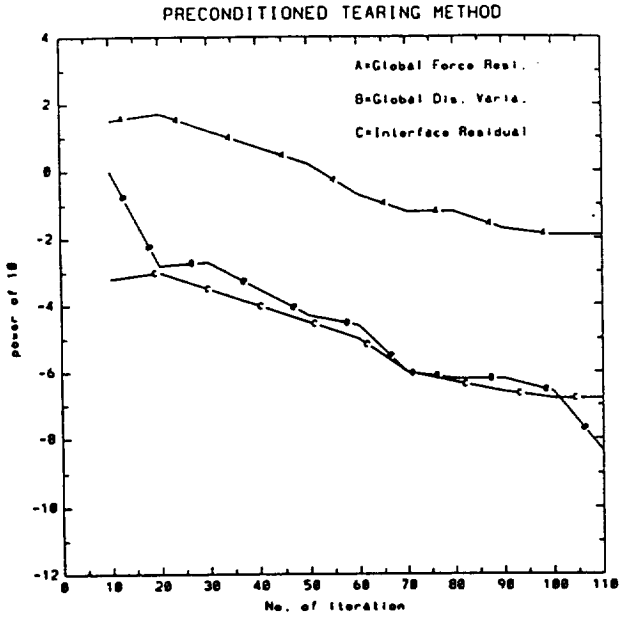


FIG. 12 *Numerical results for decomposition D2*

For each decomposition case, three curves are reported which correspond

to monitoring convergence with three different measures: (A) the global force relative residual, (B) the displacement relative variation, and (C) the interface relative residual. Clearly, decomposition $D1$ induces a faster convergence rate than decomposition $D2$. We have predicted this result since within each iteration of the iterative solution of the interface problem, information reaches all of the subdomains in decomposition $D1$, while it reaches only half of these in decomposition $D2$. Another important result relates to the relative positioning of the three curves, independently from the decomposition pattern. Note first that convergence with the global force relative residual is harder to achieve than convergence with the displacement relative variation. This is because the problem suffers from a severe ill-conditioning due to the incompressibility of the elastomer (Poisson ratio = 0.49) and the elongated shape of the cantilever composite beam. Note also that convergence with the interface relative residual is closer to convergence with the displacement relative variation than it is to convergence with the global force relative residual. This is because the interface problem is formulated in the functional space of the stresses, so that its residuals correspond to a displacement increment.

Finally, the tearing method is compared for performance with a direct Cholesky factorization. The same bending problem is selected for that purpose. Several different mesh configurations which correspond to different numbers of pencils are considered. Performance results on a CRAY-2 single processor are reported in Table 3. A tolerance of 10^{-6} on the global relative residuals is selected as a convergence criterion.

TABLE 3

Performance results on CRAY-2

Composite beam - brick elements - direct vs. 4-subdomain tearing

Number of pencils	4	9	16
NDF	13000	28000	48000
4-subdomain	**tearing**	**method**	
NDF interface	2450	7350	14700
# iterations	130	210	300
CPU time	20 s.	73 s.	193 s.
Memory size	1.6 m.w.	4.5 m.w.	9.5. m.w.
Global	**Cholesky**	**factorization**	
CPU time	15 s.	130 s.	650s.
Memory size	3.6 m.w.	16 m.w.	50 m.w.

The above results demonstrate that for sufficiently large problems, the tearing method can outperform direct solvers. For the particular problem above, it runs up to 3.3 times faster than Cholesky factorization and requires 5.2 times less memory space.

10. Closure and overview of subsequent research

A novel domain decomposition approach for the parallel finite element solution of equilibrium equations is presented. The spatial domain is partitioned into a set of totally disconnected subdomains, each assigned to an individual processor. Lagrange multipliers are introduced to enforce compatibility at the interface nodes. In the static case, each floating subdomain induces a local singularity that is resolved in two phases. First, the rigid body modes are eliminated in parallel from each local problem and a direct scheme is applied concurrently to all subdomains in order to recover each partial local solution. Next, the contributions of these modes are related to the Lagrange multipliers through an orthogonality condition. A parallel conjugate *projected* gradient algorithm is developed for the

solution of the coupled system of local rigid modes components and Lagrange multipliers, which completes the solution of the problem. When implemented on local memory multiprocessors, this proposed method of tearing and interconnecting requires less interprocessor communications than the classical method of substructuring. It is also suitable for parallel/vector computers with shared memory. Large-scale example applications are reported on the iPSC/1 and CRAY-2. Measured performance results illustrate the advantages of the proposed method and demonstrate its potential to outperform the classical method of substructures and parallel direct solvers.

It is our experience that domain decomposition methods are very sensitive to the mesh partition. In this paper, we have outlined some guidelines for the practical decomposition of a given finite element mesh. Subsequent research will focus on determining the relationship between a pattern of decomposition and the resulting conditioning of each of the local problems and the interface one. While several preconditioners for conventional domain decomposition methods (Schur methods) are available in the litterature, further research is needed to develop a preconditioner for hybrid domain decomposition algorithms such as the tearing method developped herein.

Appendix A. Solving a consistent singular system $\mathbf{K}_j\mathbf{u}_j = \mathbf{f}_j$

For completeness, we include in this appendix a derivation of the solution of a consistent singular system of equations. In this work, such a system arises in every floating subdomain Ω_j and takes the form:

$$\mathbf{K}_j\mathbf{u}_j \;=\; \mathbf{f}_j \tag{28}$$

where $\mathbf{K}_j$ is the $(n_j^s + n_j^I) \times (n_j^s + n_j^I)$ stiffness matrix associated with Ω_j, and $\mathbf{u}_j$ and $\mathbf{f}_j$ are the corresponding displacement and forcing vectors. If Ω_j has n_j^r rigid body modes, $\mathbf{K}_j$ is rank n_j^r deficient. Provided that $\mathbf{f}_j$ is orthogonal to the null space of $\mathbf{K}_j$, the singular system (28) is consistent and admits a general solution of the form:

$$\mathbf{u}_j \;=\; \mathbf{K}_j^+\mathbf{f}_j + \mathbf{R}_j\boldsymbol{\alpha} \tag{29}$$

where $\mathbf{K}_j^+$ is a pseudo-inverse of $\mathbf{K}_j$ — that is, $\mathbf{K}_j^+$ verifies $\mathbf{K}_j\mathbf{K}_j^+\mathbf{K}_j \;=\; \mathbf{K}_j$, $\mathbf{R}_j$ is a basis of the null space of $\mathbf{K}_j$ — that is, $\mathbf{R}_j$ stores the n_j^r rigid body modes of Ω_j, and $\boldsymbol{\alpha}$ is a vector of length n_j^r containing arbitrary real coefficients.

A. 1 Computing the rigid body modes

Let the superscripts p and r denote respectively a principal and a redundant quantity. The singular stiffness matrix $\mathbf{K}_j$ is partitioned as:

$$\mathbf{K}_j = \begin{bmatrix} \mathbf{K}_j^{pp} & \mathbf{K}_j^{pr} \\ \mathbf{K}_j^{prT} & \mathbf{K}_j^{rr} \end{bmatrix} \tag{30}$$

where $\mathbf{K}_j^{pp}$ has full rank equal to $n_j^s + n_j^I - n_j^r$. If $\mathbf{R}_j$ is defined as:

$$\mathbf{R}_j = \begin{bmatrix} -\mathbf{K}_j^{pp-1}\mathbf{K}_j^{pr} \\ \mathbf{I}_{n_j^r} \end{bmatrix} \tag{31}$$

where $\mathbf{I}_{n_j^r}$ is the $n_j^r \times n_j^r$ identity matrix, then $\mathbf{R}_j$ satisfies:

$$\mathbf{K}_j\mathbf{R}_j = 0$$

Moreover, $\mathbf{I}_{n_j^r}$ has full column rank and so does $\mathbf{R}_j$. Therefore, the n_j^r columns of $\mathbf{R}_j$ as defined in (29) form a basis of the null space of $\mathbf{K}_j$.

A. 2 Computing $\mathbf{K}_j^+\mathbf{f}_j$

The partitioning of the singular matrix $\mathbf{K}_j$ defined in (30) implies that:

$$\mathbf{K}_j^{rr} = \mathbf{K}_j^{prT}\mathbf{K}_j^{pp-1}\mathbf{K}_j^{pr} \tag{32}$$

Using the above identity, it can be easily checked that the matrix $\mathbf{K}_j^+$ defined as:

$$\mathbf{K}_j^+ = \begin{bmatrix} \mathbf{K}_j^{pp-1} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{bmatrix}$$

is a pseudo-inverse of $\mathbf{K}_j$. Therefore, a solution of the form $\mathbf{K}_j^+\mathbf{f}_j$ can be also written as:

$$\mathbf{u}_j = \mathbf{K}_j^+\mathbf{f}_j = \begin{bmatrix} \mathbf{K}_j^{pp-1}\mathbf{f}_j^p \\ \mathbf{O} \end{bmatrix}$$

In practice, $\mathbf{K}_j$ cannot be explicitly re-arranged as in (30). Rather, the following should be implemented when $\mathbf{K}_j$ is stored in skyline form. A zero

pivot that is encountered during the factorization process of $\mathbf{K}_j$ corresponds to a redundant equation which needs to be labeled and removed from the system. The zero pivot is set to one, the reduced column above it is copied into an extra right hand side — this corresponds to a forward reduction with $\mathbf{K}_j^{pr}\mathbf{u}_j^r$ as right hand side, and the coefficients in the skyline corresponding to that pivotal equation are set to zero. At the end of the factorization process, the non-labeled equations define the full rank matrix $\mathbf{K}_j^{pp}$. The backward substitution is modified to operate also on the n_j^r extra right hand sides in order to recover $\mathbf{u}_j^p = -\mathbf{K}_j^{pp\,-1}\mathbf{K}_j^{pr}\mathbf{u}_j^r$.

The above procedure for solving a consistent singular system of equations has almost the same computational complexity as the solution of a non-singular one.

Appendix B. Starting Lagrange multiplier vector

In this appendix we present a fast scheme for generating a starting vector $\boldsymbol{\lambda}^{(0)}$ for the conjugate projected gradient algorithm (19-20). We consider the general case of an arbitrary mesh partition.

For each floating subdomain Ω_j, the corresponding component of the starting vector has to satisfy the equality constraint:

$$\mathbf{R}_j^{I\,T}\boldsymbol{\lambda}_j^{(0)} \;=\; \mathbf{R}_j^T\mathbf{f}_j \tag{33}$$

where $\mathbf{R}_j$ is an $(n_j^s + n_j^I) \times n_j^r$ full column rank matrix which stores the rigid body modes of the floating subdomain Ω_j, $\mathbf{R}_j^I$ is the restriction of $\mathbf{R}_j$ to the intersection of Ω_j and the interface Γ_I, and $\mathbf{f}_j$ is the vector of prescribed forces in Ω_j. If $\boldsymbol{\lambda}_j^{(0)}$ is written as:

$$\boldsymbol{\lambda}_j^{(0)} \;=\; \mathbf{R}_j^I\boldsymbol{\mu}_j^{(0)} \tag{34}$$

then (33) becomes:

$$(\mathbf{R}_j^{I\,T}\mathbf{R}_j^I)\boldsymbol{\mu}_j^{(0)} \;=\; \mathbf{R}_j^T\mathbf{f}_j \tag{35}$$

which admits as solution:

$$\boldsymbol{\mu}_j^{(0)} = (\mathbf{R}_j^{I^T}\mathbf{R}_j^I)^{-1}\mathbf{R}_j^T\mathbf{f}_j \tag{36}$$

Therefore, a starting vector $\boldsymbol{\lambda}_j^{(0)}$ which satisfies the constraint equation (33) is given by:

$$\boldsymbol{\lambda}_j^{(0)} = \mathbf{R}_j^I(\mathbf{R}_j^{I^T}\mathbf{R}_j^I)^{-1}\mathbf{R}_j^T\mathbf{f}_j \tag{37}$$

The matrix product $(\mathbf{R}_j^{I^T}\mathbf{R}_j^I)$ is only $n_j^r \times n_j^r$, where n_j^r is the number of rigid body modes of the floating subdomain Ω_j. Therefore, $(\mathbf{R}_j^{I^T}\mathbf{R}_j^I)$ is at most 3×3 in two-dimensional problems and at most 6×6 in three-dimensional problems, and the evaluation of $\boldsymbol{\lambda}_j^{(0)}$ according to (37) requires little computational effort.

Acknowledgments

The first author would like to thank Professor M. Geradin of the University of Liege, Belgium, for his valuable comments. He also wishes to acknowledge partial support by NASA Langley Research Center under Grant NAG1-756, the National Science Foundation under Grant ASC-8717773, and the Air Force Office of Scientific Research under Grant AFOSR-89-0422.

References

[1] C. Farhat and E. Wilson, "A New Finite Element Concurrent Computer Program Architecture", *Int. J. Num. Meth. Eng.*, Vol. 24, No. 9, (1987), pp. 1771-1792.

[2] B. Nour-Omid, A. Raefsky and G. Lyzenga, "Solving Finite Element Equations on Concurrent Computers", *Parallel Computations and Their Impact on Mechanics*, ed. by A. K. Noor, ASME, New York, (1987), pp. 209-228.

[3] M. Ortiz and B. Nour-Omid, "Unconditionally Stable Concurrent Procedures for Transient Finite Element Analysis", *Comp. Meth. App. Mech. Eng.*, Vol. 58, (1986), pp. 151-174.

[4] C. Farhat "A Multigrid-Like Semi-Iterative Algorithm for the Massively Parallel Solution of Large Scale Finite Elements Systems", *Multigrid Methods: Proc.*

Fourth Copper Mountain Conference on Multigrid Methods, SIAM, Copper Mountain, Colorado, (1989), pp. 171-180.

[5] F. X. Roux, "Test on Parallel Machines of a Domain Decomposition Method for a Composite Three Dimensional Structural Analysis Problem", *Proc. International Conference on Supercomputing*, Saint Malo, France, (1988), pp. 273-283.

[6] F. X. Roux, "A Parallel Solver for the Linear Elasticity Equations on a Composite Beam", *Proc. Second International Conference on Domain Decomposition Methods*, ed. by T. F. Chan, R. Glowinski, J. Periaux and O. Widlund, SIAM, Los Angeles, California, (1989), pp. .

[7] G. Kron, "A Set of Principles to Interconnect the Solutions of Physical Systems", *J. Applied Physics*, Vol. 24, No. 8, (1953), pp. 965-980.

[8] T. H. H. Pian, "Finite Element Formulation by Variational Principles with Relaxed Continuity Requirements", in *The Mathematical Foundation of the Finite Element Method with Applications to Partial Differential Equations*, Part II, ed. by A. K. Aziz, Academic Press, London, (1972), pp. 671-687.

[9] Q. V. Dihn, R. Glowinski and J. Periaux, "Solving Elliptic Problems by Domain Decomposition Methods with Applications", in *Elliptic Problem Solvers II*, ed. by A. Schoenstadt, Academic Press, (1984).

[10] M. R. Dorr, "Domain Decomposition via Lagrange Multipliers", *UCRL-98532*, Lawrence Livermore National Laboratory, (1988).

[11] R. Fletcher, "Practical Methods of Optimization", Vol. 2, Constrained Optimization, J. Wiley, New York, (1981), pp. 86-87.

[12] P. E. Gill and W. Murray, "Numerical Methods for Constrained Optimization", ed. by P. E. Gill and W. Murray, Academic Press, London, (1974), pp. 132-135.

[13] F. X. Roux, "Acceleration of the Outer Conjugate Gradient by Reorthogonalization for a Domain Decomposition Method for Structural Analysis Problems", *Proc. Third International Conference on Supercomputing*, Crete, Greece, (1989), pp. 471-477.

[14] S. H. Bokhari, "On the Mapping Problem", *IEEE Transactions on Computers*, Vol. C-30, No. 3, (1981), pp. 207-214.

[15] C. Farhat, "On the Mapping of Massively Parallel Processors Onto Finite Element Graphs", *Computers & Structures*, Vol. 32, No. 2, (1989), pp. 347-354.

[16] C. Farhat, "Which Parallel Finite Element Algorithm for Which Architecture and Which Problem", *Proc. ASME Winter Annual Meeting*, San Francisco, California, December 14 (1989)

[17] C. Farhat, "A Simple and Efficient Automatic FEM Domain Decomposer", *Computers & Structures*, Vol. 28, No. 5, (1988), pp. 579-602.

[18] L. Bomans and D. Roose, "Benchmarking the iPSC/2", *Report TW 114*, Katholieke Universiteit Leuven, Department of Computer Science, Belgium, (1988).

USING A REDUCED NUMBER OF LAGRANGE MULTIPLIERS FOR ASSEMBLING PARALLEL INCOMPLETE FIELD FINITE ELEMENT APPROXIMATIONS

Charbel Farhat
Department of Aerospace Engineering Sciences
and Center for Space Structures and Controls
University of Colorado at Boulder
Boulder, CO 80309-0429, U. S. A.

and

M. Geradin
Laboratoire de Techniques Aeronautiques et Spatiales
University of Liege
Rue Ernest Solvay, 21, B-4000 Liege, Belgium
and Conseiller Scientifique, O. N. E. R. A.
29 Av. de la Division Leclerc
BP72 92322 Chatillon Cedex, France

Abstract. A domain decomposition algorithm based on a hybrid variational principle was proposed in reference [1] for the parallel finite element solution of self-adjoint elliptic partial differential equations. First, the spatial domain was partitioned into a set of totally disconnected subdomains and an incomplete finite element solution was computed in each of these subdomains. Next, a number of Lagrange multipliers equal to the number of degrees of freedom located at the binding interface were introduced to enforce compatibility constraints between the independent local finite element approximations. For structural and mechanical problems, the resulting algorithm was shown to outperform the conventional method of substructures, especially on parallel processors. Here, the use of a much lower number of Lagrange multipliers for interconnecting the incomplete field finite element solutions is investigated. When accuracy is preserved, this approach reduces drastically the computational complexity of the Schur-complement-like coupling system that is associated with the interface region and enhances significantly the overall performance of the methodology. Finite element procedures for both global and piecewise polynomial approximations of the Lagrange multipliers are derived. Finally, some numerical results obtained for structural example problems that validate the main idea and highlight its advantages are presented.

1. Introduction.

Recently, Farhat and Roux [1] have presented a parallel finite element computational method for the solution of static equilibrium problems that is a departure from the parallel method of substructures (see, for example, Nour-Omid, A. Raefsky and G. Lyzenga [2], Farhat and Wilson [3]). The unique feature about the proposed procedure is that it requires fewer interprocessor communication than traditional domain decomposition algorithms, while it still offers the same amount of parallelism. The computational strategy was denoted by "finite element tearing and interconnecting" because of its resemblance with the very early work of Kron [4] on tearing methods for electric circuit models. Basically, the finite element mesh is "torn" into a set of totally disconnected submeshes and a computational strategy is derived from a hybrid variational principle where the inter-subdomain continuity constraint is removed via the introduction of a Lagrange multiplier function.

In reference [1], the authors have interconnected the subdomain incomplete finite element solutions with a number of discrete Lagrange multipliers that is equal to the number of degrees of freedom that are lying on the binding interface. That allowed them to recover exactly the same finite element solution as with non-hybrid variational principles. Here, we consider the use of a substantially lower number of discrete Lagrange multipliers, which would further enhance the serial and parallel performance of the proposed computational algorithm when an adequate accuracy is preserved. The fundamental idea is not essentially different from the one presented in the mathematical work of Dorr [5]. In order to motivate this approach, we first re-derive in Section 2 the basic method of tearing and interconnecting and summarize its major computational advantages. In Sections 3 and 4, we develop polynomial and piecewise low order polynomial expressions for the finite element discretization of the interface Lagrange multiplier function and describe their computer implementation. We consider both cases of continuum and lattice structures. In Section 5, we present an iterative refinement procedure for improving the accuracy of the resulting algorithm and in Section 6 we report on some numerical results obtained for two-subdomain problems and problems where the meshes are decomposed with one-way separators only. These preliminary results indicate that a very high accuracy is achieved with a very low number of discrete Lagrange multipliers. We also highlight the computational advantages of the proposed parallel algorithm with the large-scale static analysis of the Solid Rocket Booster (SRB) on the CRAY Y-MP; for that problem, the parallel skyline and banded solvers are outperformed.

2. A method of finite element tearing and interconnecting

Here we summarize a domain decomposition based algorithm associated with a hybrid formulation for the parallel finite element solution of the linear elastostatic problem (Farhat and Roux [1]). The method is equally applicable to the finite element solution of any self-adjoint elliptic partial differential equation. For the sake of clarity, we consider only the case of two subdomains. The generalization for an arbitrary number of subdomains is fully developed in [1].

The variational form of the three-dimensional elastostatic boundary-value problem goes as follows. Given g and h, find the displacement function u which is a stationary point of the energy functional:

$$J(v) = \frac{1}{2}a(v,v) - (v,g) - (v,h)_\Gamma$$

where

$$\begin{aligned} a(v,w) &= \int_\Omega v_{(i,j)} c_{ijkl} w_{(k,l)} \, \delta\Omega \\ (v,g) &= \int_\Omega v_i g_i \, \delta\Omega \\ (v,h)_\Gamma &= \int_{\Gamma_h} v_i h_i \, \delta\Gamma \end{aligned} \tag{1}$$

In the above, the indices i, j, k take the value 1 to 3, $v_{(i,j)} = (v_{i,j} + v_{j,i})/2$ and $v_{i,j}$ denotes the partial derivative of the $i-th$ component of v with respect to the $j-th$ spatial variable, c_{ijkl} are the elastic coefficients, Ω denotes the volume of the elastostatic body, Γ its piecewise smooth boundary, and Γ_h the piece of Γ where the tractions h_i are prescribed.

If Ω is torn into two regions Ω_1 and Ω_2 (Fig. 1), solving the above elastostatic problem is equivalent to finding the two displacements functions u_1 and u_2 which are stationary points of the energy functionals:

$$J_1(v_1) = \frac{1}{2}a(v_1, v_1)_{\Omega_1} - (v_1, g)_{\Omega_1} - (v_1, h)_{\Gamma_1}$$
$$J_2(v_2) = \frac{1}{2}a(v_2, v_2)_{\Omega_2} - (v_2, g)_{\Omega_2} - (v_2, h)_{\Gamma_2}$$

where

$$\begin{aligned}
a(v_1, w_1)_{\Omega_1} &= \int_{\Omega_1} v_{1(i,j)} c_{ijkl} w_{1(k,l)} \, \delta\Omega \\
a(v_2, w_2)_{\Omega_2} &= \int_{\Omega_2} v_{2(i,j)} c_{ijkl} w_{2(k,l)} \, \delta\Omega \\
(v_1, g)_{\Omega_1} &= \int_{\Omega_1} v_{1i} f_i \, \delta\Omega \\
(v_2, g)_{\Omega_2} &= \int_{\Omega_2} v_{2i} f_i \, \delta\Omega \\
(v_1, h)_{\Gamma_1} &= \int_{\Gamma_{h1}} v_{1i} h_i \, \delta\Gamma \\
(v_2, h)_{\Gamma_2} &= \int_{\Gamma_{h2}} v_{2i} h_i \, \delta\Gamma
\end{aligned} \tag{2}$$

and which satisfy on the interface boundary Γ_I the continuity constraint:

$$u_1 = u_2 \quad \text{on } \Gamma_I \tag{3}$$

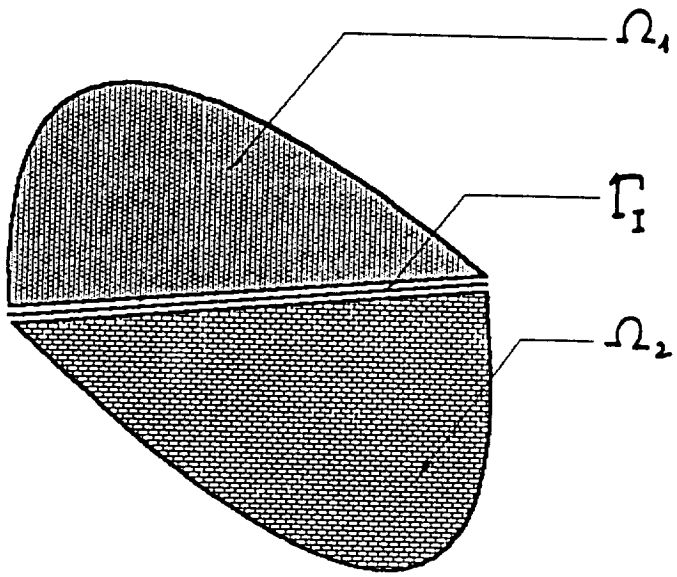


FIG. 1 *Tearing in two subdomains*

The two above variational problems (2) with the subsidiary continuity condition (3) can be casted into a single hybrid variational principle (see, for example, Pian [6], Zienkiewicz and Taylor [7] and references cited therein) which corresponds to finding the saddle point of the total potential energy:

$$J^*(v_1, v_2, \mu) \;=\; J_1(v_1) + J_2(v_2) - \int_{\Gamma_I} \lambda(v_1 - v_2)\, \delta\Gamma \tag{4}$$

If now the displacement fields u_1 and u_2 are expressed by suitable shape functions as:

$$u_1 \;=\; \mathbf{N}\mathbf{u}_1 \quad \text{and} \quad u_2 \;=\; \mathbf{N}\mathbf{u}_2 \tag{5}$$

and the continuity equation is enforced for the discrete problem — that is, if a discrete Lagrange multiplier λ_i is introduced at each $i-th$ degree of freedom of the discrete interface boundary Γ_I, a standard Galerkin procedure transforms the hybrid variational principle (4) in the following algebraic system:

$$\begin{aligned} \mathbf{K}_1\mathbf{u}_1 &= \mathbf{f}_1 + \mathbf{B}_1^T\boldsymbol{\lambda} \\ \mathbf{K}_2\mathbf{u}_2 &= \mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda} \\ \mathbf{B}_1\mathbf{u}_1 &= \mathbf{B}_2\mathbf{u}_2 \end{aligned} \tag{6}$$

where $\mathbf{K}_j$, $\mathbf{u}_j$, and $\mathbf{f}_j$, $j = 1, 2$, are respectively the stiffness matrix, the displacement vector, and the prescribed force vector associated with the finite element discretization of Ω_j. The vector of Lagrange multipliers $\boldsymbol{\lambda}$ represents the interaction forces between the two subdomains Ω_1 and Ω_2 along their common boundary Γ_I. It introduces in the above system of equations the quantities $\mathbf{K}_1^{-1}\mathbf{B}_1^T\boldsymbol{\lambda}$ and $\mathbf{K}_2^{-1}\mathbf{B}_2^T\boldsymbol{\lambda}$ which implicitly correct the incomplete finite element solutions $\mathbf{K}_1^{-1}\mathbf{f}_1$ and $\mathbf{K}_2^{-1}\mathbf{f}_2$.

Within each subdomain Ω_j, we denote the number of interior nodal unknowns by n_j^s and the number of interface nodal unknowns by n_j^I. The total number of interface nodal unknowns is denoted by n_I. Note that $n_I = n_1^I = n_2^I$ in the particular case of two subdomains. If the interior degrees of freedom are numbered first and the interface ones are numbered last, each of the two boolean connectivity matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ takes the form:

$$\mathbf{B}_j = [\mathbf{O}_j \quad \mathbf{I}_j] \quad j = 1, 2 \tag{7}$$

where $\mathbf{O}_j$ is an $n_j^I \times n_j^s$ null matrix and $\mathbf{I}_j$ is the $n_j^I \times n_j^I$ identity matrix. The vector of Lagrange multipliers $\boldsymbol{\lambda}$ is n_I long.

The stiffness matrices $\mathbf{K}_1$ and $\mathbf{K}_2$ are non singular if and only if each of the defined subdomains has enough prescribed boundary conditions to eliminate its rigid body modes. However a typical mesh decomposition often produces a certain number of floating subdomains. If in the above example Ω_2 is a floating subdomain, equations (6) can be re-arranged after some algebraic manipulations (see [1]) as:

$$\begin{aligned} \begin{bmatrix} \mathbf{F}_I & -\mathbf{R}_2^I \\ -\mathbf{R}_2^{I\,T} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\alpha} \end{bmatrix} &= \begin{bmatrix} \mathbf{B}_2\mathbf{K}_2^+\mathbf{f}_2 - \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{f}_1 \\ -\mathbf{R}_2^T\mathbf{f}_2 \end{bmatrix} \\ \mathbf{u}_1 &= \mathbf{K}_1^{-1}(\mathbf{f}_1 + \mathbf{B}_1^T\boldsymbol{\lambda}) \\ \mathbf{u}_2 &= \mathbf{K}_2^+(\mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda}) + \mathbf{R}_2\boldsymbol{\alpha} \end{aligned} \tag{8}$$

where $\mathbf{F}_I = \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{B}_1^T + \mathbf{B}_2\mathbf{K}_2^+\mathbf{B}_2^T$, $\mathbf{K}_2^+$ is a pseudo-inverse of $\mathbf{K}_2$, $\mathbf{R}_2$ is an $(n_2^s + n_2^I) \times n_2^r$ rectangular matrix whose columns represent the n_r rigid body modes of

Ω_2 and α specifies a linear combination of these. For three-dimensional problems $n_2^r \leq 6$, and for two-dimensional problems $n_2^r \leq 3$. Clearly, the Lagrangian matrix is indefinite. However, $\mathbf{F}_I$ is symmetric positive definite and $\mathbf{R}_2^I$ has full column rank. Therefore, the system of equations in $(\boldsymbol{\lambda}, \boldsymbol{\alpha})$ is symmetric and non-singular. It admits a unique solution $(\boldsymbol{\lambda}, \boldsymbol{\alpha})$ which uniquely determines $\mathbf{u}_1$ and $\mathbf{u}_2$.

It is important to note that since $n_2^r \leq 6$, the Lagrangian system (8) and $\mathbf{F}_I$ have almost the same size. For an arbitrary number of subdomains N_s of which N_f are floating, the additional number of equations introduced by the handling of local singularities is bounded by $6N_f$. For large-scale problems and relatively coarse mesh partitions, this number — which determines the size of $\boldsymbol{\alpha}$, is a very small fraction of the size of the global system. On the other hand, if a given tearing process does not result in any floating subdomain, $\boldsymbol{\alpha}$ vanishes and the corresponding Lagrangian and $\mathbf{F}_I$ systems become identical.

In reference [1], a set of guidelines for carrying out the practical decomposition of an arbitrary mesh, as well as a parallel computational scheme for solving equations (8) in the presence of an arbitrary number of subdomains were presented. The proposed computational scheme featured a parallel preconditioned conjugate *projected* gradient algorithm for the solution of the indefinite Lagrangian system. It was also shown that the proposed method of finite element tearing and interconnecting compares favorably with the conventional method of substructures and with direct solvers on both serial and parallel computers. It is particularly attractive for local memory multiprocessors such as hypercubes because it intrinsically requires much less interprocessor communication than the parallel method of substructures [2]. This is because the need for interprocessor communication in this formulation is exclusively induced by the weak form of the continuity constraint:

$$(v_i - v_j, \lambda)_{\Gamma_{I_{ij}}} = \int_{\Gamma_{I_{ij}}} \lambda(v_i - v_j)\, \delta\Gamma \tag{9}$$

and because if $\Gamma_{I_{ij}}$ has a zero measure, then $(v_i - v_j, \lambda)_{\Gamma_{I_{ij}}} = 0$ and no exchange of information is needed between subdomains Ω_i and Ω_j. Therefore the subdomains which interconnect along one edge in three-dimensional problems and those which interconnect along one vertex in both two and three-dimensional problems do not require any interprocessor communication. This is unlike the parallel method of substructures and other conventional domain decomposition algorithms.

The efficiency of the tearing method outlined above depends on how fast the Schur complement or interface system represented here by $\mathbf{F}_I$ can be solved. This is often the case for many of the subdomain based implicit/explicit parallel solution algorithms. The nature of $\mathbf{F}_I = \mathbf{B}_1\mathbf{K}_1^{-1}\mathbf{B}_1^T + \mathbf{B}_2\mathbf{K}_2^{+}\mathbf{B}_2^T$ makes the solution of the interface system inadequate by any technique which requires this submatrix explicitly. This implies that a direct method or an iterative method of the SOR type cannot be used. The only efficient method for solving this system is that of conjugate gradients, because once $\mathbf{K}_1$ and $\mathbf{K}_2$ have been factorized, matrix-vector products of the form $\mathbf{F}_I\mathbf{v}$ can be performed very efficiently using only forward and backward substitutions. Therefore convergence rate becomes the key factor for enhancing the overall efficiency of the procedure. In reference [1], the authors have considered careful mesh partitioning schemes and a suitable preconditioner for improving this convergence rate. Here we investigate an approach for speeding up the solution of the interface system which consists of reducing drastically its size. When this can be achieved (without hurting accuracy) to an extent where $\mathbf{F}_I$ can be explicitly formed, assembled and stored, a direct solution method is applied to the Schur complement equations so that the convergence rate is not any longer an issue. Otherwise, the same semi-iterative algorithm as presented in [1] is used for solving the new interface system that is characterized by a much smaller size than in our previous work.

In this paper, we concentrate on the two-subdomain problem which highlights the main idea and does not require a substantial amount of coding. The obtained results are so encouraging (Section 6) that we have started developing the necessary software for handling arbitrary mesh decompositions with multiple subdomains. This effort will be reported in a forthcoming paper.

Next, we discretize the Lagrange multiplier function that binds the subdomain incomplete solutions using a polynomial approximation and derive the finite element representation of the new interface system.

3. Approximating the Lagrange multipliers with polynomials

The weak form of the equations of static equilibrium associated with the hybrid variational principle formulated in equation (4) is obtained using the standard

virtual work principle. It is expressed as:

$$\begin{aligned}
\int_{\Omega_1} \delta \mathbf{u}_1{}^T \mathbf{L}^T \mathbf{D} \mathbf{L} \mathbf{u}_1 \delta\Omega_1 - \int_{\Gamma_I} \delta \mathbf{u}_1{}^T \lambda \delta\Gamma - \int_{\Omega_1} \delta \mathbf{u}_1{}^T \mathbf{g} \delta\Omega - \int_{\Gamma_I} \delta \mathbf{u}_1{}^T \mathbf{h} \delta\Gamma &= 0 \\
\int_{\Omega_2} \delta \mathbf{u}_2{}^T \mathbf{L}^T \mathbf{D} \mathbf{L} \mathbf{u}_2 \delta\Omega_2 + \int_{\Gamma_I} \delta \mathbf{u}_2{}^T \lambda \delta\Gamma - \int_{\Omega_2} \delta \mathbf{u}_2{}^T \mathbf{g} \delta\Omega - \int_{\Gamma_I} \delta \mathbf{u}_2{}^T \mathbf{h} \delta\Gamma &= 0 \\
\int_{\Gamma_I} \delta\lambda (\mathbf{u}_1^T - \mathbf{u}_2^T) \delta\Gamma &= 0
\end{aligned} \tag{10}$$

where the vectors $\mathbf{g}$ and $\mathbf{h}$ have been defined in equations (2), the vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ in equations (5), and $\mathbf{D}$ and $\mathbf{L}$ are the matrix representations of, respectively, a constitutive equation and a spatial derivative operator.

If the Lagrange multiplier function λ is degree-of-freedom collocated along the interface — that is, a discrete Lagrange multiplier scalar λ_i is attached at each degree of freedom lying on the interface boundary Γ_I, the above equations are transformed into the algebraic equations (6), where the vector of Lagrange multipliers $\boldsymbol{\lambda}$ is n_I long. As a result, the interface system of equations (8) is $n_I \times n_I$ large. In order to reduce the size of this system, we consider first a polynomial approximation for λ. For this purpose, we assume that the finite element problem of interest has d degrees of freedom per node and that the interface Γ_I between Ω_1 and Ω_2 is parametrized by a curvilinear abscissa s (Fig. 2).

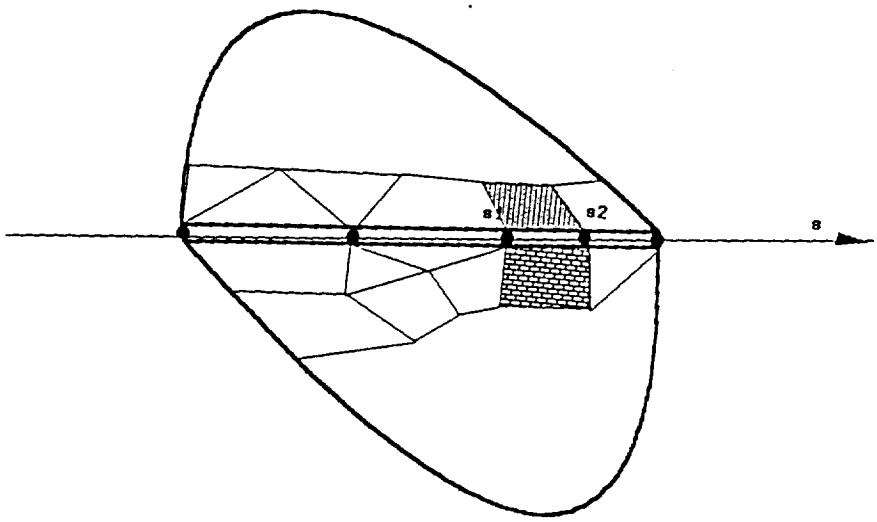


FIG. 2 *Parametrization of a two-subdomain interface*

We define d polynomials of degree p as:

$$\begin{aligned}
\lambda^1(s) &= \sum_{k=0}^{k=p} \lambda_k^1 s^k \\
\lambda^2(s) &= \sum_{k=0}^{k=p} \lambda_k^2 s^k \\
&\cdot \\
&\cdot \\
&\cdot \\
\lambda^d(s) &= \sum_{k=0}^{k=p} \lambda_k^d s^k
\end{aligned} \tag{11}$$

where p is much smaller than n_I and $\{\lambda_k^1, \lambda_k^2, ..., \lambda_k^d, k = 0, 1, ..., p\}$ are $(p+1)d$ unknown discrete Lagrange multipliers. Physically, these still represent the interface tractions that are necessary to maintain equilibrium between the two subdomains Ω_1 and Ω_2. The superscript $j, j = 1, 2. ..., d$ denotes the directional freedom (x, y, or z displacement/rotation) of the corresponding traction component. However, unlike in our previous work, these multipliers are not specified at any location of the discrete interface Γ_I. In particular, they are not necessarily attached to any particular node. Substituting (11) into (10) after re-arranging the third of equations (10) results in the algebraic system:

$$\begin{aligned}
\mathbf{K}_1\mathbf{u}_1 &= \mathbf{f}_1 + \mathbf{B}_1^{pT}\boldsymbol{\lambda}_p \\
\mathbf{K}_2\mathbf{u}_2 &= \mathbf{f}_2 - \mathbf{B}_2^{pT}\boldsymbol{\lambda}_p \\
\mathbf{B}_1^p\mathbf{u}_1 &= \mathbf{B}_2^p\mathbf{u}_2
\end{aligned} \tag{12}$$

where $\boldsymbol{\lambda}_p$ is now the $(p+1)d$ long vector:

$$\boldsymbol{\lambda}_p = [\lambda_0^1 \quad \lambda_0^2 \quad . \quad . \quad . \quad \lambda_0^d \quad . \quad . \quad . \quad \lambda_p^1 \quad \lambda_p^2 \quad . \quad . \quad . \quad \lambda_p^d]^T \tag{13}$$

and $\mathbf{B}_1^p$ and $\mathbf{B}_2^p$ are now non-boolean finite element matrices of sizes $(p+1)d \times (n_1^s + n_I)$ and $(p+1)d \times (n_2^s + n_I)$ that are assembled from their element level correspondents $\mathbf{B}_1^{p(e)}$ and $\mathbf{B}_2^{p(e)}$ in the usual manner:

$$\mathbf{B}_j^p = \sum_e \mathbf{B}_j^{p(e)} \quad j = 1, 2 \tag{14}$$

where e spans only the set of elements that are connected to the interface boundary Γ_I. For a finite element e with q nodes lying on Γ_I, the $qd \times (p+1)d$ element level matrices $\mathbf{B}_j^{p(e)}$, $j = 1, 2$ are given by:

$$\mathbf{B}_j^{p(e)} = \begin{bmatrix} {}^1\mathbf{B}_j^{p(e)} \\ {}^2\mathbf{B}_j^{p(e)} \\ \cdot \\ \cdot \\ \cdot \\ {}^q\mathbf{B}_j^{p(e)} \end{bmatrix} \tag{15}$$

where ${}^l\mathbf{B}_j^{p(e)}$, $l = 1, 2, ..., q$ is a $d \times (p+1)d$ matrix associated with the $l-th$ node of element e and has the following form:

$${}^l\mathbf{B}_j^{p(e)} = \begin{bmatrix} {}^l_0\mathcal{B}_j^{p(e)} & {}^l_1\mathcal{B}_j^{p(e)} & {}^l_2\mathcal{B}_j^{p(e)} & \cdot & \cdot & \cdot & {}^l_p\mathcal{B}_j^{p(e)} \end{bmatrix} \tag{16}$$

and ${}^l_k\mathcal{B}_j^{p(e)}$, $k = 0, ..., p$ is a $d \times d$ diagonal matrix associated with the $k-th$ monomial s^k and is expressed as:

$${}^l_k\mathcal{B}_j^{p(e)} = \left(\int_{\Gamma_I{}^{(e)}} \mathbf{N}_l s^k \delta\Gamma \right) \mathbf{I}_d \tag{17}$$

where $\mathbf{N}_l$ is the shape function associated with the $l-th$ node of element e and $\mathbf{I}_d$ is the $d \times d$ identity matrix. As an example, for elements that have two and only two nodes lying on Γ_I ($q = 2$) and for the case of linear shape functions $\mathbf{N}_l$, the submatrices ${}^l_k\mathcal{B}_j^{p(e)}$, $l = 1, 2$, $k \leq p$, are given by:

$$\begin{aligned} {}^1_k\mathcal{B}_j^{p(e)} &= \frac{1}{s_2 - s_1} \left(\frac{s_1^{k+2} - s_2^{k+2}}{k+2} + \frac{s_2^{k+2} - s_1^{k+1} s_2}{k+1} \right) \mathbf{I}_d \\ {}^2_k\mathcal{B}_j^{p(e)} &= \frac{1}{s_2 - s_1} \left(\frac{s_2^{k+2} - s_1^{k+2}}{k+2} + \frac{s_1^{k+2} - s_2^{k+1} s_1}{k+1} \right) \mathbf{I}_d \end{aligned} \tag{18}$$

where $s_1 < s_2$, and s_1 and s_2 are the curvilinear abscissae of the two nodes of element e that lie on Γ_I (Fig. 2).

At this point, it is worthwhile to point out that increasing the degree of the polynomial approximation of $\lambda^k, k = 1, 2, ..., d$ involves only adding a few columns to the existing element level matrices ${}^l\mathbf{B}_j^{p(e)}$, as it is suggested by equation (16).

For two-subdomain tearings, the constraint matrices $\mathbf{B}_j^p$ have the following pattern:

$$\mathbf{B}_j^p = [\mathbf{O}_j \quad \bar{\mathbf{B}}_j^p] \quad j = 1, 2 \tag{19}$$

where $\mathbf{O}_j$ is an $(p+1)d \times n_j^s$ null matrix and $\bar{\mathbf{B}}_j^p$ is the $(p+1)d \times n_I$ sparse matrix:

$$\bar{\mathbf{B}}_j^p = \begin{bmatrix} \mathcal{D}_{11} & \dots & \mathcal{D}_{rr} \\ \dots & \dots & \dots \\ \mathcal{D}_{p+1,1} & \dots & \mathcal{D}_{p+1,r} \end{bmatrix} \tag{20}$$

where $r = n_I/d$ and $\mathcal{D}_{ij}$ is a $d \times d$ diagonal matrix.

Equations (12) above can be re-arranged as:

$$\begin{aligned} \begin{bmatrix} \mathbf{F}_I^p & -\mathbf{R}_2^I \\ -\mathbf{R}_2^{I^T} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}_p \\ \boldsymbol{\alpha} \end{bmatrix} &= \begin{bmatrix} \mathbf{B}_2^p \mathbf{K}_2^+ \mathbf{f}_2 - \mathbf{B}_1^p \mathbf{K}_1^{-1} \mathbf{f}_1 \\ -\mathbf{R}_2^T \mathbf{f}_2 \end{bmatrix} \\ \mathbf{u}_1 &= \mathbf{K}_1^{-1}(\mathbf{f}_1 + \mathbf{B}_1^T \boldsymbol{\lambda}_p) \\ \mathbf{u}_2 &= \mathbf{K}_2^+(\mathbf{f}_2 - \mathbf{B}_2^T \boldsymbol{\lambda}_p) + \mathbf{R}_2 \boldsymbol{\alpha} \end{aligned} \tag{21}$$

where all variables have the same physical meaning as previously. However, the size resulting interface system

$$\mathbf{F}_I^p = \mathbf{B}_1^p \mathbf{K}_1^{-1} \mathbf{B}_1^{pT} + \mathbf{B}_2^p \mathbf{K}_2^+ \mathbf{B}_2^{pT} \tag{22}$$

is now only $(p+1)d \times (p+1)d$. Since $\boldsymbol{\lambda}_p$ does not enforce the continuity constraint equation (3) at each of the nodes of the discrete interface Γ_I, the finite element field approximations $\mathbf{u}_1$ and $\mathbf{u}_2$ given by the solution of system (21) are in general discontinuous along Γ_I. In order to uniquely define the finite element solution along the interface boundary, we average the two computed solutions to obtain:

$$\mathbf{u}^* = \mathbf{u}|_{\Gamma_I} = \frac{1}{2}(\mathbf{u}_1 + \mathbf{u}_2)|_{\Gamma_I} \tag{23}$$

We postulate that the above averaged interface solution $\mathbf{u}^*$ is more accurate than each of the restrictions of the subdomain solutions $\mathbf{u}_1|_{\Gamma_I}$ and $\mathbf{u}_2|_{\Gamma_I}$. Therefore, we back-propagate to the interior of the subdomains Ω_1 and Ω_2 the enhancing effect of the averaging procedure (23) by imposing $u = u^*$ on Γ_I and solving two independent displacement-driven subdomain problems. For this purpose, we first partition the stiffness matrix of each subdomain as:

$$\mathbf{K}_j = \begin{bmatrix} \mathbf{K}_{j_{ss}} & \mathbf{K}_{j_{sI}} \\ \mathbf{K}_{j_{sI}}^T & \mathbf{K}_{j_{II}} \end{bmatrix} \qquad j = 1, 2 \tag{24}$$

where the subscripts ss, II and sI refer respectively to interior, interface and interior/interface coupling quantities. For any set of given boundary conditions and any mesh decomposition pattern, the resulting $\mathbf{K}_{j_{ss}}$ stiffness matrix is non-singular. Next, the improved finite element subdomain solutions are computed as:

$$\mathbf{u}_j = \mathbf{K}_{j_{ss}}^{-1}(\mathbf{f}_{j_s} - \mathbf{K}_{j_{sI}}\mathbf{u}^*) \qquad j = 1, 2 \tag{25}$$

It should be noted that the above improvement of the subdomain solutions $\mathbf{u}_1$ and $\mathbf{u}_2$ is perfectly parallelizable and requires only one sparse matrix-vector multiply and one pair of sparse forward/backward substitutions per subdomain. The triangular factors of $\mathbf{K}_{j_{ss}}$ are embedded in those of $\mathbf{K}_j$ which have been previously computed.

Usually, the stresses that develop in a structure are more important to the analyst than the displacements it undergoes. However, the above improvement procedure is such that if $\mathbf{u}^*$ is a highly accurate approximation of the interface solution, $\mathbf{u}_j$, $j = 1, 2$ become highly accurate approximations of the subdomain solutions and therefore it is not necessary to monitor the stress fields.

The solution approach presented here requires a parametrization of the interface boundary Γ_I. For a given finite element model and a given mesh decomposition, the interface boundary Γ_I is always well defined for continuum problems. Therefore, its parametrization is straightforward, especially for two-subdomain problems. However, lattice structures require a special treatment. For the latter problems, if Γ_I is constrained to follow the path defined by the structural

members that connect the nodes that are shared by two lattice subdomains, Γ_I will not be identical on both sides of the interface (Fig. 3a-3c). Therefore for lattice structures we select Γ_I as the "geometrical path" that (a) is the simplest to parametrize, and (b) has the same trace on the lattice subdomains it interconnects. In particular, only the finite element nodes of this interface need to intersect with the structure. Figure 3d depicts Γ_I for the structure shown in Figure 3a.

In general, the number of Lagrange multipliers, N_λ, that is needed to achieve a certain accuracy is problem dependent. If this number is rather small — say less than a hundred, then it is feasible to form explicitly $\mathbf{F}_I^p$ and solve the system of equations (21) using a direct method. Otherwise, the semi-iterative solution algorithm developed in reference [1] is recommended. However, beyond a certain value of N_λ, the polynomial approach developed in this Section becomes numerically problematic. Indeed, approximating the Lagrange multiplier functions with higher order polynomials of degree $p = N_\lambda/d - 1$ typically results in very ill-conditioned matrices $\mathbf{B}_j^p \mathbf{K}_j^{-1} \mathbf{B}_j^{pT}$, which may cause the performance and/or accuracy of the proposed computational method to deteriorate. Next in Section 4 we develop piecewise low order polynomial approximations for the finite element discretization of the Lagrange multiplier functions (11), that are suitable for the case of a rather large value of N_λ. We remind the reader that d denotes the number of degrees of freedom per node; for simplicity, it is assumed to be constant over the nodes. Therefore, since N_λ denotes the total number of discrete Lagrange multipliers, N_λ/d represents the number of locations where surface tractions, or discrete Lagrange multipliers, are to be introduced.

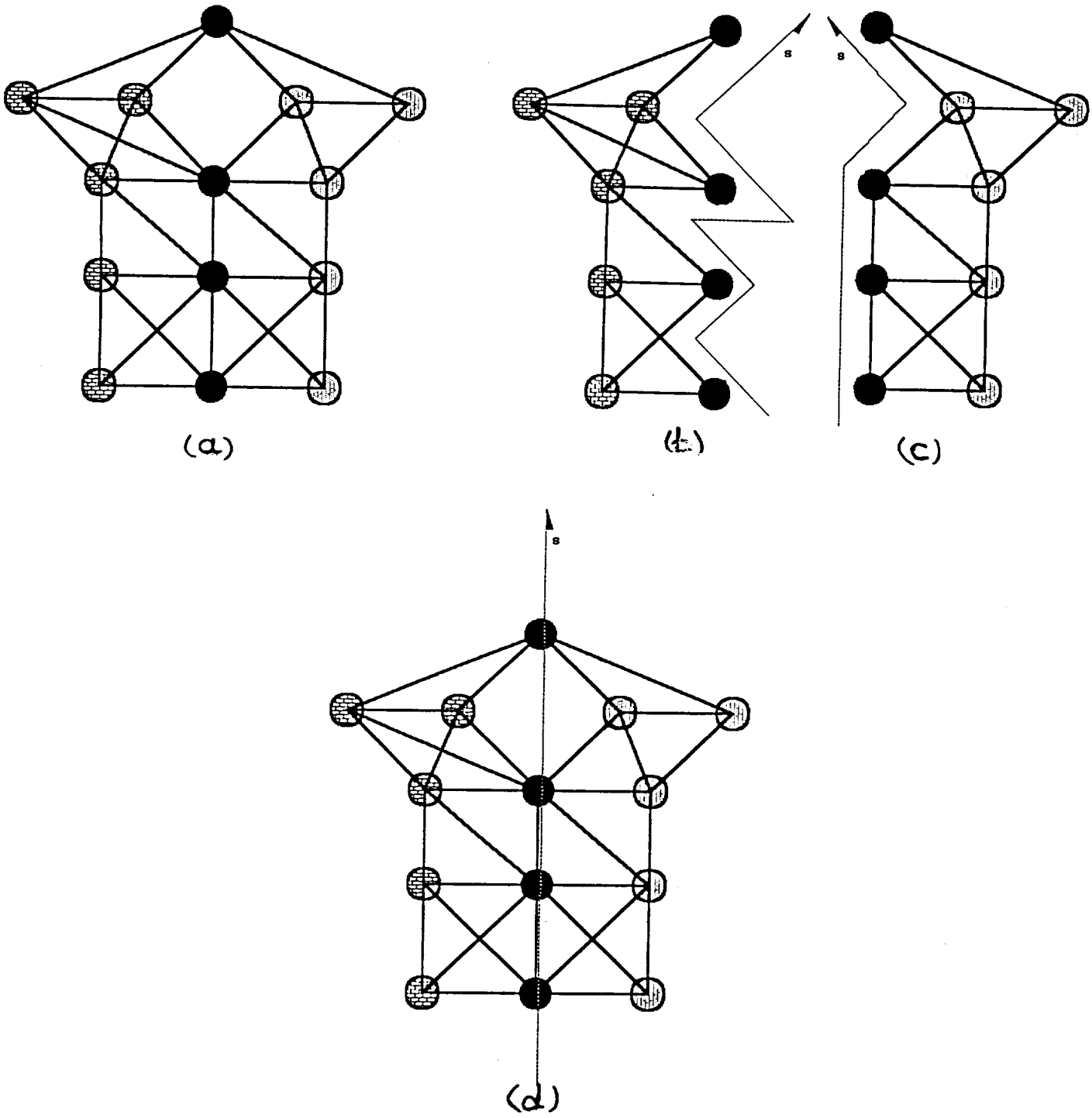


FIG. 3 *Interface boundary definition for lattice structure*
(a) truss structure - (b) continuum-like left interface
(c) continuum-like right interface - (d) adopted interface boundary

4. Piecewise low order polynomial approximations

The objective of this section is to develop an alternative procedure for the finite element discretization of the interface tractions that results in a better conditioned interface problem than previously when the total number of discrete Lagrange multipliers that are introduced, N_λ, is rather large.

Let Γ_I^k, $k = 0, ..., N_\lambda/d - 2$ denote a partition $\mathcal{P}$ of the interface boundary Γ_I defined as:

$$\Gamma_I^k = [s_k, s_{k+1}] \qquad k = 0, ..., N_\lambda/d - 2 \tag{26}$$

where s_k, $k = 0, ..., N_\lambda/d - 1$ are the curvilinear abscissae of N_λ/d specified points on Γ_I where the discrete surface tractions λ_k^j are introduced. Within each subinterval Γ_I^k, we define d cubic polynomial expressions for the Lagrange multiplier approximations as:

$$\begin{aligned}
\tilde{\lambda}_k^1(s) &= c_{1\,k}^1 + c_{2\,k}^1(s - s_k) + c_{3\,k}^1(s - s_k)^2 + c_{4\,k}^1(s - s_k)^3 \\
\tilde{\lambda}_k^2(s) &= c_{1\,k}^2 + c_{2\,k}^2(s - s_k) + c_{3\,k}^2(s - s_k)^2 + c_{4\,k}^2(s - s_k)^3 \\
&\cdot \\
&\cdot \\
&\cdot \\
\tilde{\lambda}_k^d(s) &= c_{1\,k}^d + c_{2\,k}^d(s - s_k) + c_{3\,k}^d(s - s_k)^2 + c_{4\,k}^d(s - s_k)^3
\end{aligned} \tag{27}$$

where $c_{i\,k}^j$, $i = 1, ..., 4$, $j = 1, ..., d$ are determined by imposing:

$$\begin{aligned}
\tilde{\lambda}_k^j(s_k) &= \lambda_k^j \quad ; \quad \tilde{\lambda}_k^j(s_{k+1}) = \lambda_{k+1}^j \\
\frac{d\tilde{\lambda}_k^j}{ds}(s_k) &= \frac{d\lambda^j}{ds}(s_k) \; ; \; \frac{d\tilde{\lambda}_k^j}{ds}(s_{k+1}) = \frac{d\lambda^j}{ds}(s_{k+1}) \\
k &= 0, ..., N_\lambda/d - 2 \\
j &= 1, ..., d
\end{aligned} \tag{28}$$

The first set of equations (28) imply that $\tilde{\lambda}_k^j(s_{k+1}) = \tilde{\lambda}_{k+1}^j(s_{k+1})$, so that all λ^j are guaranteed to be continuously approximated on Γ_I. The second set of equations (28) involve the derivatives of the Lagrange multiplier functions which are neither available nor part of the weak form of the static equations of equilibrium (10). Following Conte and de Boor [8], we approximate these derivatives by:

$$\frac{d\tilde{\lambda}_k^j}{ds}(s_k) = \frac{\frac{\Delta s_{k-1}}{\Delta s_k}(\lambda_{k+1} - \lambda_k) + \frac{\Delta s_k}{\Delta s_{k-1}}(\lambda_k - \lambda_{k-1})}{\Delta_2 s_k} \tag{29}$$

where Δs_k and $\Delta_2 s_k$ are defined as:

$$\begin{aligned} \Delta s_k &= s_{k+1} - s_k \\ \Delta_2 s_k &= s_{k+1} - s_{k-1} \end{aligned} \tag{30}$$

Note that (29) requires the two additional points s_{-1} and $s_{N_\lambda/d}$ which we choose as:

$$\begin{aligned} s_{-1} &= s_2 \\ s_{N_\lambda/d} &= s_{N_\lambda/d-3} \end{aligned} \tag{31}$$

Substituting (27) and (29) into (28) determines the constants $c_{i\,k}^j$ as functions of the discrete Lagrange multipliers:

$$\begin{aligned} c_{1k}^j &= \lambda_k^j \\ c_{2k}^j &= \xi_{2k}\lambda_{k+1}^j + \eta_{2k}\lambda_k^j + \zeta_{2k}\lambda_{k-1}^j \\ c_{3k}^j &= \xi_{3k}\lambda_{k+2}^j + \eta_{3k}\lambda_{k+1}^j + \zeta_{3k}\lambda_k^j + \nu_{3k}\lambda_{k-1}^j \\ c_{4k}^j &= \xi_{4k}\lambda_{k+2}^j + \eta_{4k}\lambda_{k+1}^j + \zeta_{4k}\lambda_k^j + \nu_{4k}\lambda_{k-1}^j \end{aligned} \tag{32}$$

where ξ_{2k}, ξ_{3k}, ξ_{4k}, η_{2k}, η_{3k}, η_{4k}, ζ_{2k}, ζ_{3k}, ζ_{4k}, ν_{3k}, and ν_{4k} are constants that depend only the curvilinear abscissae s_{k-1}, s_k, s_{k+1} and s_{k+2} (see *Appendix A*).

As previously, equations (32) are substituted into equations (27) and (27) into (10) to obtain:

$$\begin{aligned} \mathbf{K}_1\mathbf{u}_1 &= \mathbf{f}_1 + \mathbf{B}_1^{\mathcal{P}\,T}\boldsymbol{\lambda}_{\mathcal{P}} \\ \mathbf{K}_2\mathbf{u}_2 &= \mathbf{f}_2 - \mathbf{B}_2^{\mathcal{P}\,T}\boldsymbol{\lambda}_{\mathcal{P}} \\ \mathbf{B}_1^{\mathcal{P}}\mathbf{u}_1 &= \mathbf{B}_2^{\mathcal{P}}\mathbf{u}_2 \end{aligned} \tag{33}$$

where $\mathbf{B}_1^{\mathcal{P}}$ and $\mathbf{B}_2^{\mathcal{P}}$ are now non-boolean finite element matrices of sizes $N_\lambda \times (n_1^s + n_I)$ and $N_\lambda \times (n_2^s + n_I)$. The subscript/superscript $\mathcal{P}$ emphasizes the dependence of these quantities on the partition $\mathcal{P}$ of the interface boundary Γ_I (26).

Both matrices are assembled from their element level correspondents ${}_{k^e}\mathbf{B}_1^{\mathcal{P}(e)}$ and ${}_{k^e}\mathbf{B}_2^{\mathcal{P}(e)}$ in the usual manner:

$$\mathbf{B}_j^{\mathcal{P}} = \sum_e {}_{k^e}\mathbf{B}_j^{\mathcal{P}(e)} \quad j = 1, 2 \tag{34}$$

where e spans only the set of elements that are connected to Γ_I. The left subscript k^e emphasizes the dependence of ${}_{k^e}\mathbf{B}_1^{\mathcal{P}(e)}$ on the subinterval $\Gamma_I^k = [s_k,\ s_{k+1}]$ where one edge of element e falls. For a finite element e with q nodes lying on Γ_I, the $qd \times N_\lambda$ element level matrices ${}_{k^e}\mathbf{B}_j^{\mathcal{P}(e)}$, $j = 1, 2$ are given by:

$${}_{k^e}\mathbf{B}_j^{\mathcal{P}(e)} = \begin{bmatrix} {}^1_{k^e}\mathbf{B}_j^{\mathcal{P}(e)} \\ {}^2_{k^e}\mathbf{B}_j^{\mathcal{P}(e)} \\ \cdot \\ \cdot \\ \cdot \\ {}^q_{k^e}\mathbf{B}_j^{\mathcal{P}(e)} \end{bmatrix} \tag{35}$$

where ${}^l_{k^e}\mathbf{B}_j^{\mathcal{P}(e)}$, $l = 1, 2, ..., q$ is a $d \times N_\lambda$ matrix associated with the $l-th$ node of element e and has the following form:

$${}^l_{k^e}\mathbf{B}_j^{\mathcal{P}(e)} = \left[\mathcal{O}_L^{k^e} \quad {}^l_{k^e-1}\mathcal{B}_j^{\mathcal{P}(e)} \quad {}^l_{k^e}\mathcal{B}_j^{\mathcal{P}(e)} \quad {}^l_{k^e+1}\mathcal{B}_j^{\mathcal{P}(e)} \quad {}^l_{k^e+2}\mathcal{B}_j^{\mathcal{P}(e)} \quad \mathcal{O}_R^{k^e}\right] \tag{36}$$

where $\mathcal{O}_L^{k^e}$ and $\mathcal{O}_R^{k^e}$ are respectively left and right $d \times (k^e - 1)d$ and $d \times (N_\lambda - (k^e + 3)d)$ zero matrices, and ${}^l_{k^e}\mathcal{B}_j^{\mathcal{P}(e)}$ is expressed as:

$${}^l_{k^e}\mathcal{B}_j^{\mathcal{P}(e)} = \beta_{k^e}\mathbf{I}_d \tag{37}$$

where $\mathbf{I}_d$ is the $d \times d$ identity matrix and β_{k^e-1}, β_{k^e}, β_{k^e+1} and β_{k^e+2} are function only of the partition $\mathcal{P}$ of Γ_I and are given by the following integration:

$$\int_{\Gamma_I{}^{(e)}} \tilde{\lambda}^j_{k^e}\mathbf{N}_l\delta\Gamma = \beta_{k^e-1}\lambda^j_{k^e-1} + \beta_{k^e}\lambda^j_{k^e} + \beta_{k^e+1}\lambda^j_{k^e+1} + \beta_{k^e+2}\lambda^j_{k^e+2} \tag{38}$$

It should be noted that while the symbolic derivation of equations (36-38) appears to be somehow complicated, their computer implementation is straightforward and their processing is inexpensive.

For two-subdomain tearings, the constraint matrices $\mathbf{B}_j^{\mathcal{P}}$ take the following form:

$$\mathbf{B}_j^{\mathcal{P}} = \left[\mathbf{O}_j \quad \bar{\mathbf{B}}_j^{\mathcal{P}} \right] \quad j = 1, 2 \tag{39}$$

where $\mathbf{O}_j$ is an $N_\lambda \times n_j^s$ null matrix and $\bar{\mathbf{B}}_j^p$ is the $N_\lambda \times n_I$ four diagonal sparse matrix:

$$\bar{\mathbf{B}}_j^p = \begin{bmatrix} \searrow & \searrow & \searrow & \searrow & \mathcal{O} & \dots & \dots & \mathcal{O} \\ \mathcal{O} & \mathcal{D}_{k-1,k} & \mathcal{D}_{kk} & \mathcal{D}_{k,k+1} & \mathcal{D}_{k,k+2} & \mathcal{O} & \dots & \mathcal{O} \\ \dots & \dots & \searrow & \searrow & \searrow & \searrow & \dots & \mathcal{O} \\ \dots & \dots & \dots & \dots & \dots & \mathcal{O} & \mathcal{D}_{r-1,r} & \mathcal{D}_{rr} \end{bmatrix} \tag{40}$$

where $r = N_\lambda / d$ and $\mathcal{D}_{ij}$ is a $d \times d$ diagonal matrix.

After $\mathbf{B}_1^{\mathcal{P}}$ and $\mathbf{B}_2^{\mathcal{P}}$ are set up, the system of equations (33) is solved as described in Section 3. In particular, the averaging and correcting procedures outlined in Section 3 are also used.

Approximating λ with polynomials (Section 3) does not require the location of the corresponding physical surface tractions to be specified. On the other hand, using piecewise low order polynomials for that purpose (Section 4) requires the definition of a partitioning $\mathcal{P}$ of Γ_I, which corresponds to specifying the location of the physical surface tractions along Γ_I. Therefore from a practical viewpoint, the first approach seems more attractive. However, specifying where a surface traction is to be introduced can be turned into an advantage if one looks at it as an additional freedom. For example, if the stress field along Γ_I can be predicted qualitatively prior to the analysis, the partition $\mathcal{P}$ will be refined in the areas of oscillation or high concentration, and coarse otherwise. That would improve the efficiency of the approximation.

5. An iterative refinement procedure for accuracy improvement

Here we outline an iterative refinement procedure for improving the accuracy of the results when it is required. We discuss both cases of polynomial and piecewise low order polynomial approximations, and assume that a reasonable initial value $N_\lambda^{(0)}$ is given. We select as convergence criterion:

$$\begin{aligned} ||\mathbf{u}^{1(m+1)}||_\infty - ||\mathbf{u}^{1(m)}||_\infty &< \epsilon ||\mathbf{u}^{1(m)}||_\infty \\ ||\mathbf{u}^{2(m+1)}||_\infty - ||\mathbf{u}^{2(m)}||_\infty &< \epsilon ||\mathbf{u}^{2(m)}||_\infty \\ &\cdots \\ ||\mathbf{u}^{d(m+1)}||_\infty - ||\mathbf{u}^{d(m)}||_\infty &< \epsilon ||\mathbf{u}^{d(m)}||_\infty \end{aligned} \tag{41}$$

where the superscripts d and m refer respectively to the $d-th$ component of the solution at each node and to the $m-th$ iteration, and ϵ is a specified tolerance. As indicated by equations (41), we independently monitor the convergence of each of the d components of the displacement field. This is in order to avoid that potential important relative errors in a component of the solution whose magnitude is relatively small — for example, the x displacement of a cantilever beam with a load parallel to the y direction, are masked by an otherwise perfect convergence for a component of the solution whose magnitude is relatively large — for example, the y displacement.

5.1. Polynomial approximation

Let $N_\lambda^{(m)}$ and $p^{(m)} = N_\lambda^{(m)}/d - 1$ denote respectively the number of discrete Lagrange multipliers and the degree of the polynomial approximation of λ at iteration m. Suppose that for $N_\lambda^{(m)}$ the above convergence criterion (41) is not met. A simple iterative refinement procedure consists in introducing at iteration $m+1$ an additional discrete Lagrange multiplier by considering a polynomial approximation $\tilde{\lambda}$ of order $p^{(m+1)} = p^{(m)} + 1$. This entails the generation of the constraint matrices $\mathbf{B}_j^{p^{(m+1)}}$, $j = 1, 2$ and therefore of the element level matrices ${}^l\mathbf{B}_j^{p^{(m+1)(e)}}$, $l = 1, ..., q$. A careful examination of equations (15-17) reveals that ${}^l\mathbf{B}_j^{p^{(m+1)(e)}}$ can be computed very fast by updating ${}^l\mathbf{B}_j^{p^{(m)(e)}}$ as following:

$${}^l\mathbf{B}_j^{p^{(m+1)(e)}} = \left[{}^l\mathbf{B}_j^{p^{(m)(e)}} \quad {}^l_{p^{(m)}+1}\mathcal{B}_j^{p^{(m)}+1} \right] \tag{42}$$

where

$$ {}^{l}_{p^{(m)}+1}\mathcal{B}_j^{p^{(m)}+1} = \left(\int_{\Gamma_I{}^{(e)}} \mathbf{N}_l s^{p^{(m)}+1} \delta\Gamma\right) \mathbf{I}_d \tag{43} $$

Therefore, if at each iteration $m+1$ an additional Lagrange multiplier is introduced, the re-generation of the constraint matrices requires only the evaluation for each interface element of the integral $\int_{\Gamma_I{}^{(e)}} \mathbf{N}_l s^{p^{(m)}+1} \delta\Gamma$, and the re-generation of the interface system $\mathbf{F}_I$ requires only the pre/post multiplication of the subdomain flexibilities with these matrices. Given that $N_\lambda^{(m)}/d << n_I$, these multiplications are not expensive. In particular, they are much more economical than those corresponding to a typical conjugate gradient iteration for the case $N_\lambda = n_I$. The introduction at iteration $m+1$ of more than one discrete Lagrange multiplier is handled exactly in the same manner.

5.2. Piecewise low order polynomial approximation

Let $\Gamma_I^{k(m)}$, $k = 0, ..., N_\lambda^{(m)}/d - 2$ denote the partition $\mathcal{P}^{(m)}$ of the interface boundary Γ_I at iteration m:

$$ \Gamma_I^{k(m)} = [s_k^{(m)},\ s_{k+1}^{(m)}] \qquad k = 0, ..., N_\lambda^{(m)}/d - 2 \tag{44} $$

If at iteration $m+1$ an additional discrete Lagrange multiplier is introduced, say in the subinterval $\Gamma_I^{k^*(m)}$, the resulting partition $\mathcal{P}^{(m+1)}$ becomes:

$$ \Gamma_I^{k(m+1)} = [s_k^{(m+1)},\ s_{k+1}^{(m+1)}] \qquad k = 0, ..., N_\lambda^{(m)}/d - 1 \tag{45} $$

where

$$ \begin{aligned} s_k^{(m+1)} &= s_k^{(m)} \qquad k \le k^* \\ s_k^{(m+1)} &= s_{k-1}^{(m)} \qquad k > k^* + 1 \end{aligned} \tag{46} $$

It can be easily shown that the regeneration at iteration $m+1$ of the constraint matrices $\mathbf{B}_1^{\mathcal{P}}$ and $\mathbf{B}_2^{\mathcal{P}}$ involves basically recomputing the coefficients c_{ik}^j, $i = 1, ..., 4$, $j = 1, ..., d$ of the polynomial expressions (32), only for those interface elements which intersect Γ_I inside $\Gamma_I^{k^*-1(m+1)}$, or $\Gamma_I^{k^*(m+1)}$, or $\Gamma_I^{k^*+1(m+1)}$, or

$\Gamma_I^{k^*+2^{(m+1)}}$. However, a refinement procedure for this case should also specify the location where an additional discrete Lagrange multiplier is to be introduced during the following iteration, — that is to define $s_{k^*}^{(m+1)}$. In this work, we choose fo that purpose the point of Γ_I where $||\mathbf{u}^{(m+1)}||_\infty - ||\mathbf{u}^{(m)}||_\infty / ||\mathbf{u}^{(m)}||_\infty$ and/or the violation of static equilibrium prior to the averaging and improvement procedures (23) and (25) are the largest. The introduction at iteration $m+1$ of more than one discrete Lagrange multiplier is handled exactly in the same manner.

6. Validation and Performance Evaluation

Ideally, the accuracy of the proposed hybrid method for a given value of N_λ should be assessed by comparing its generated results to the exact (analytical) solution of the continuum or lattice problem of interest. However, the latter solution is seldom available. Therefore, we select as reference the conventional finite element solution of the problem — that is, the solution that is obtained without the introduction of a hybrid variational principle, and refer to it as the exact solution.

In this section, we validate first the essence of this paper with simple two-subdomain structural problems. For each example, we apply the iterative refinement procedures outlined in Section 5 to generate numerical results corresponding to various numbers of Lagrange multipliers, N_λ. We report on only the computed solutions associated with the interface boundary Γ_I. This is because whenever these converge to the reference solution, the improvement procedure (25) guarantees that the computed subdomain displacement and stress fields also converge to their reference solution. All examples indicate that a number of traction forces that is only a small fraction of the size of the discrete interface boundary Γ_I are required to "glue" the incomplete subdomain solutions. Next, we assess the performance of the developed computational hybrid algorithm with the large-scale finite element static analysis of the Solid Rocket Booster (SRB) on a 4 processor CRAY Y-MP; we demonstrate that for that problem, our algorithm outperforms the fastest of the available parallel skyline solvers.

6.1 Validation

First, we consider the static analysis of an unsymmetric beam that is clamped at both ends and subjected to both a horizontal and vertical point loadings. The beam is discretized using 4-node plane stress elements ($q = 4$) with two degrees of freedom per node ($d = 2$). The finite element mesh is decomposed into 2 subdomains, each with 108 interior degrees of freedom. For this problem, the size of interface problem is $n_I = 18$.

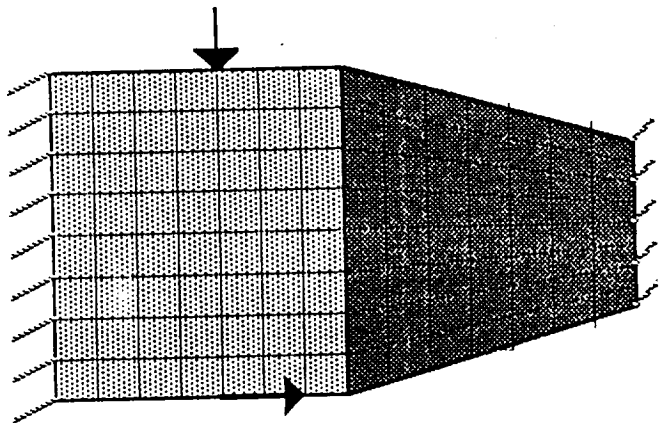


FIG. 4 *Two-subdomain decomposition of an unsymmetric clamped-clamped beam*

The interface tractions are approximated successively with polynomials of order zero, one, two, and three, — that is, $N_\lambda = 2, 4, 6$, and 8. The generated hybrid solutions are reported in Figures (5-6) for both the horizontal and vertical displacement fields along the interface boundary Γ_I. For $N_\lambda = 6$, both displacement fields are shown to be in excellent agreement with the exact solution.

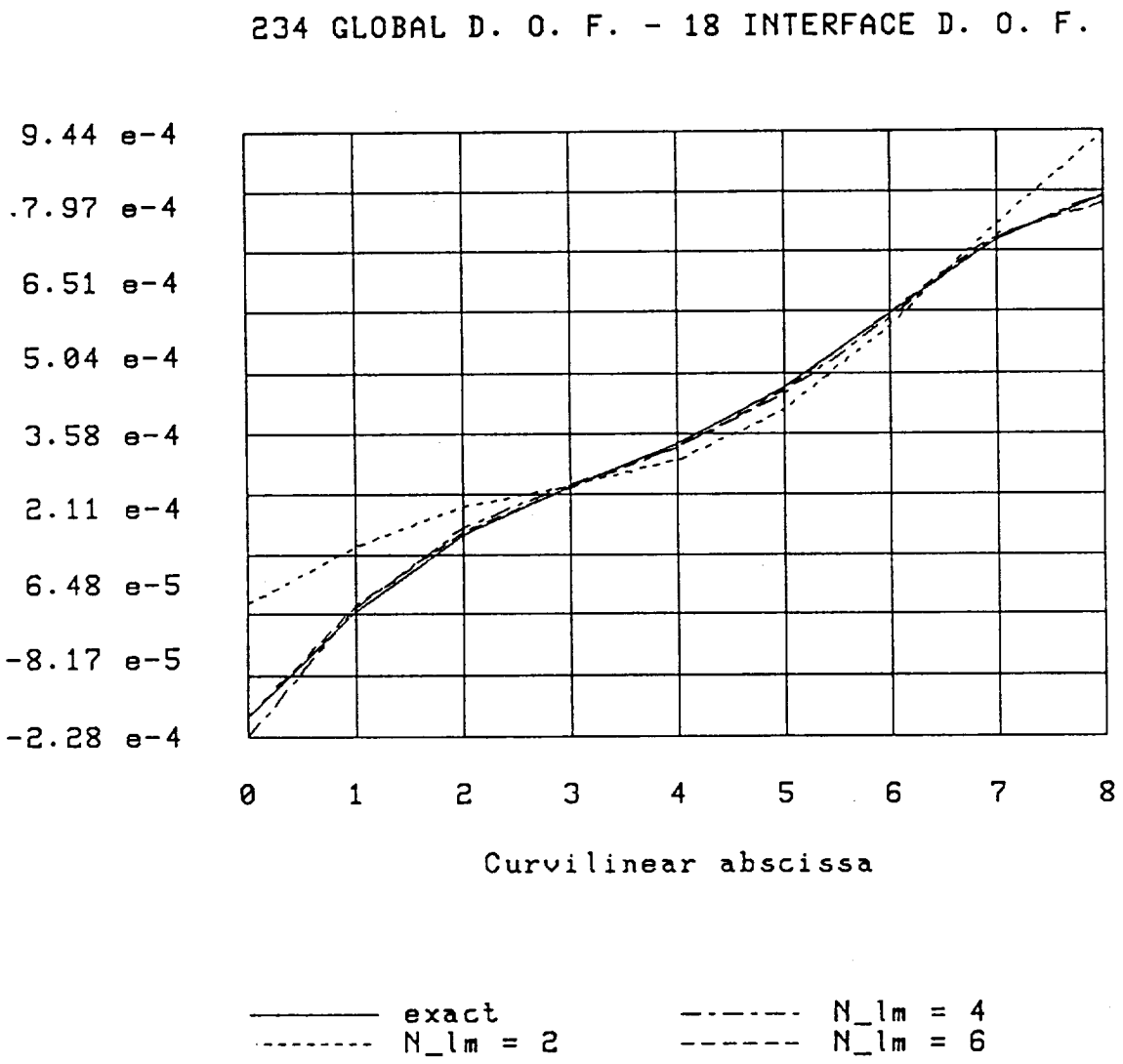


FIG. 5 *Unsymmetric beam problem: predicted horizontal displacement field*

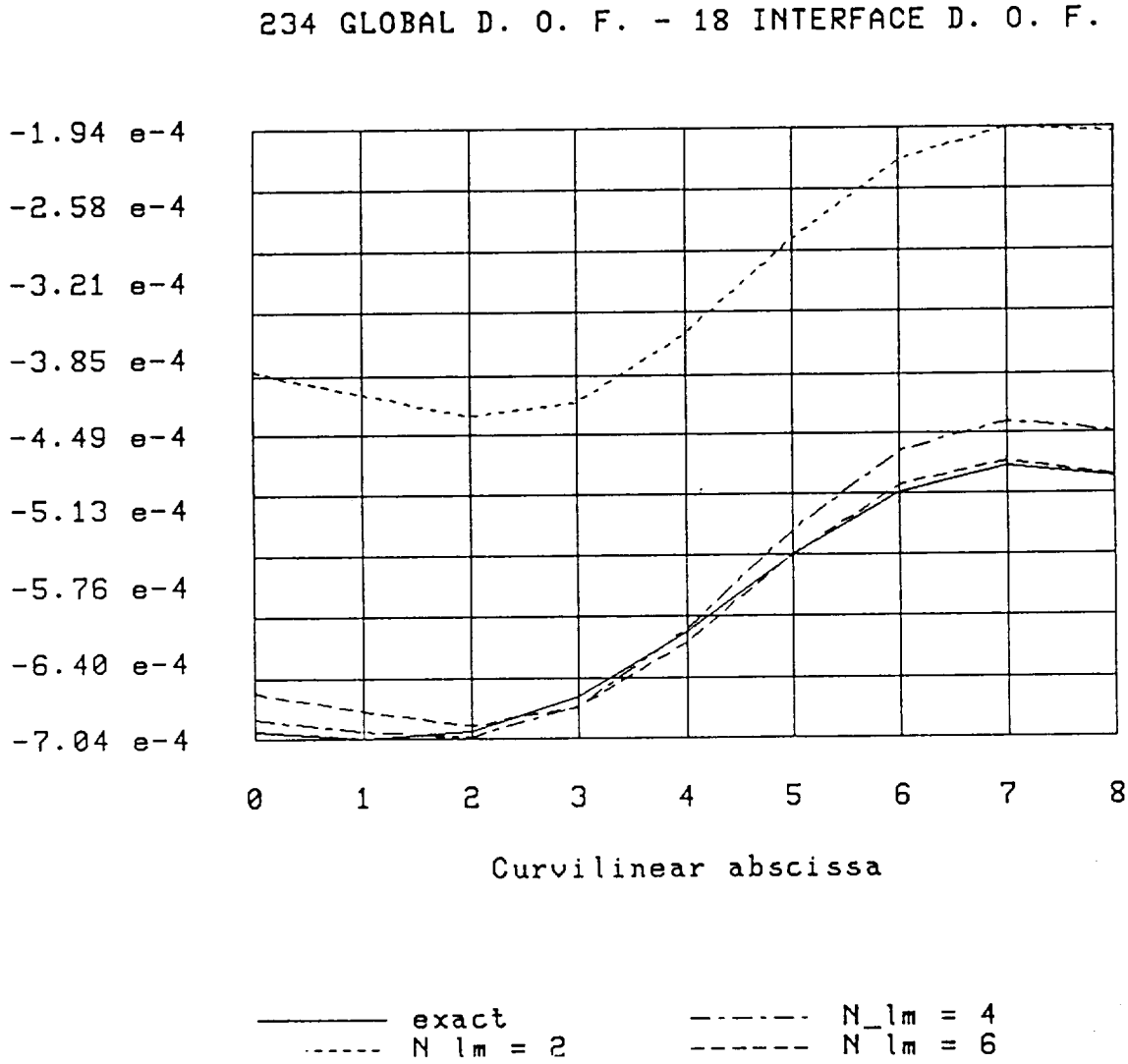


FIG. 6 *Unsymmetric beam problem: predicted vertical displacement field*

Next, we analyze an unsymmetric planar truss structure ($q = 2$, $d = 2$) with 312 degrees of freedom. The unsymmetry is induced by the members material properties which are different on both sides of the axis of geometrical symmetry. The truss structure is also loaded in both directions as shown in Figure 7. The lattice mesh is decomposed in two subdomains, each with 144 internal degrees of freedom. The interface boundary Γ_I is depicted in Figure 7.

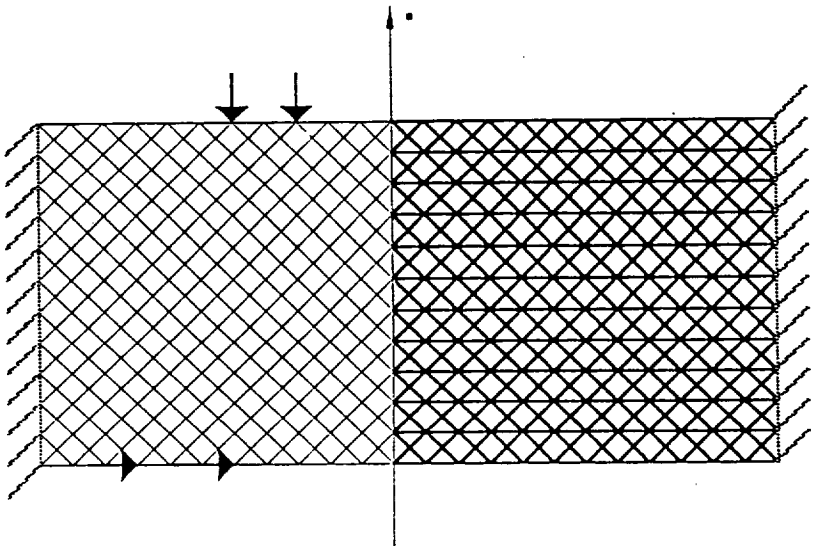


FIG. 7 *Two-subdomain decomposition of an unsymmetric fixed-fixed truss*

The size of the interface problem for the above structure is rather small ($n_I = 24$), so that polynomial approximations for the Lagrange multiplier functions are considered again. The predicted vertical and horizontal displacements using the tearing hybrid method are reported in Figures (8-9). Adequate accuracy is achieved for $N_\lambda = 6$, which corresponds to only 25% of the number of degrees of freedom along Γ_I.

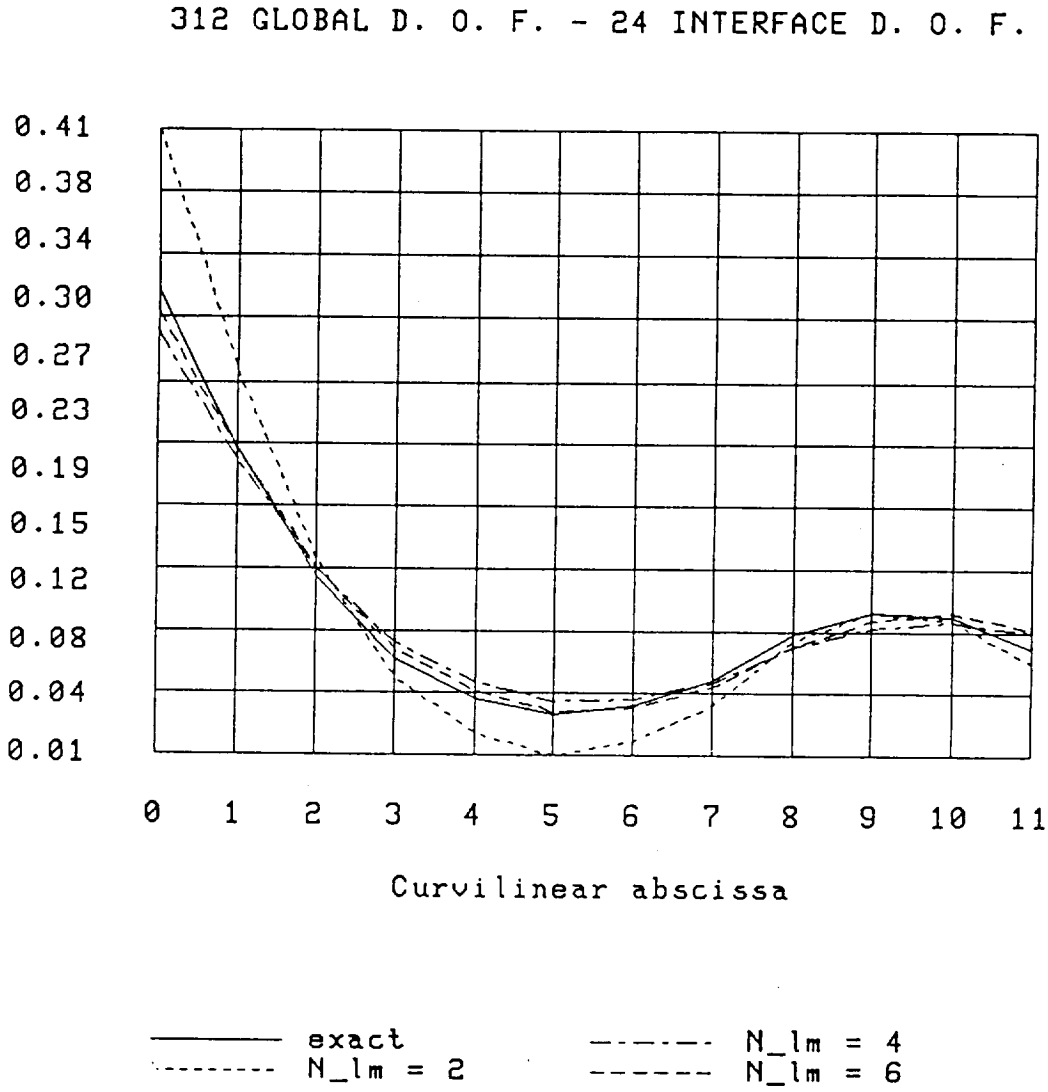


FIG. 8 *Truss problem: predicted horizontal displacement field*

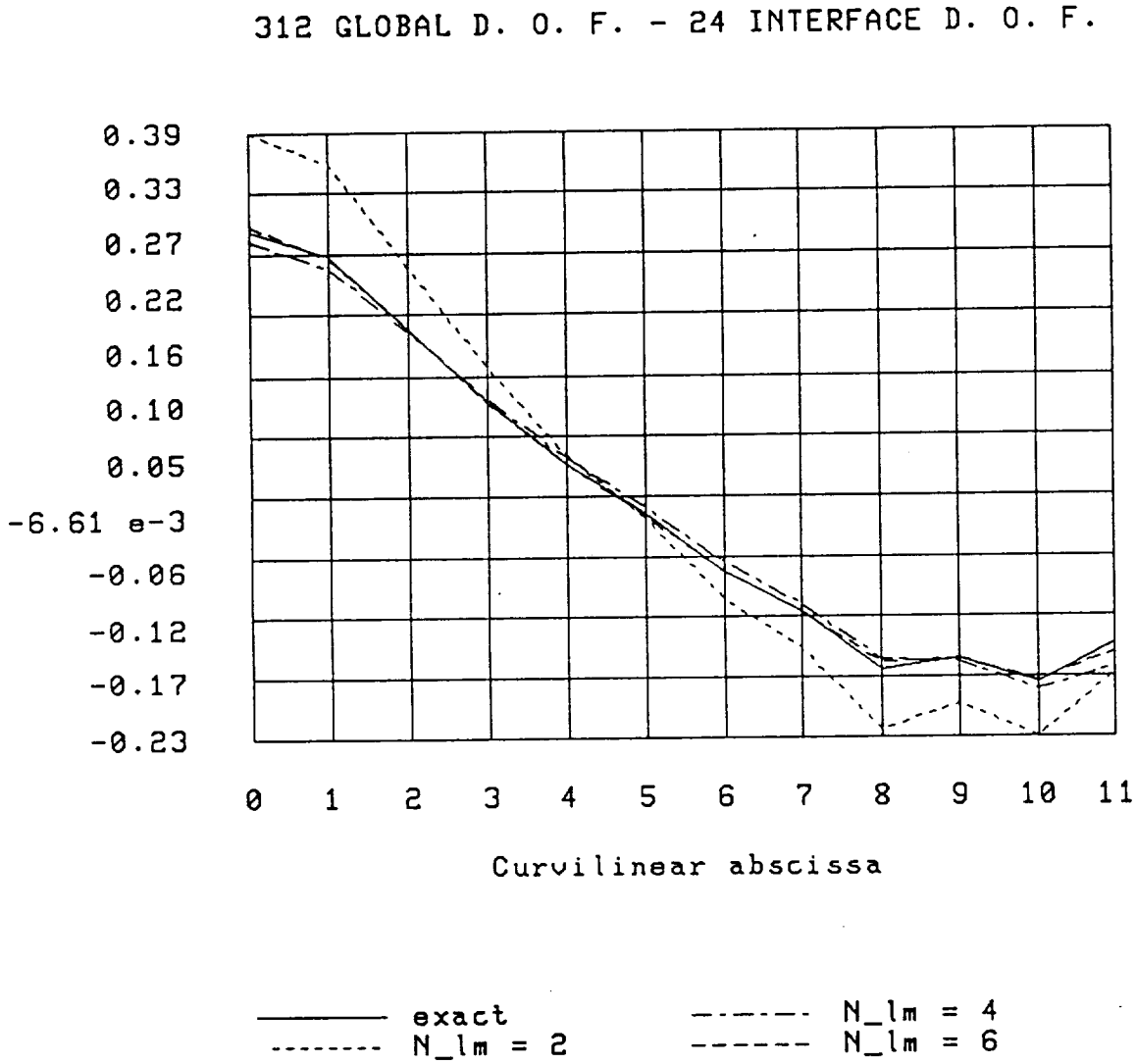


FIG. 9 *Truss problem: predicted vertical displacement field*

Finally, we select to illustrate the use of piecewise low order polynomials for the approximation of the interface tractions with the static analysis of a cantilever beam. A finite element mesh with 300 degrees of freedom is constructed using 4-node plane stress elements ($q = 4$) with two degrees of freedom per node ($d = 2$). It is partitioned in two non-floating subdomains, each with a minimum bandwidth (Fig. 10). The horizontal slicing adopted for this problem avoids the subdomain singularity but produces a larger interface than a vertical slicing. The size of the interface problem is 60.

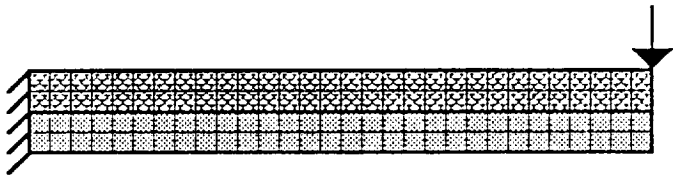


FIG. 10 *Two-subdomain decomposition of cantilever beam*

An intitial partition $\mathcal{P}^{(0)}$ of Γ_I is defined using four points ($N_\lambda = 8$), of which three are clustered towards the free end where the vertical force is applied. The iterative refinement procedure of Section 5 introduces an additional point in the subinterval that is closest to the load (Fig. 11).

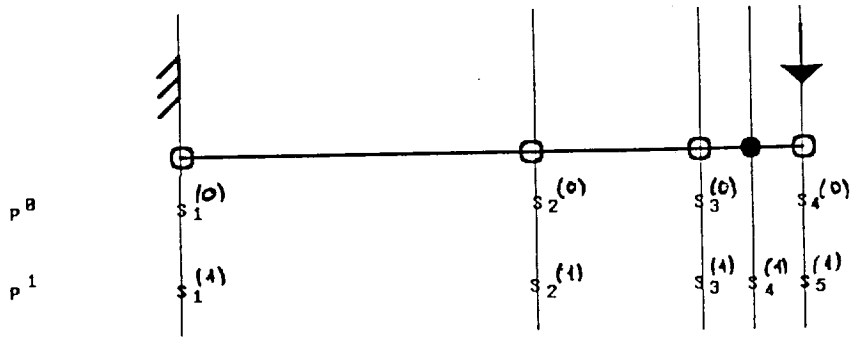


FIG. 11 *Successive partitionings of* Γ_I

Within three iterations, the tearing hybrid algorithm is shown to converge towards the exact solution (Fig. 12-13). Note however that it took only two iterations for the vertical displacement to converge. This example illustrates the need for a component-by-component convergence criterion as in (41).

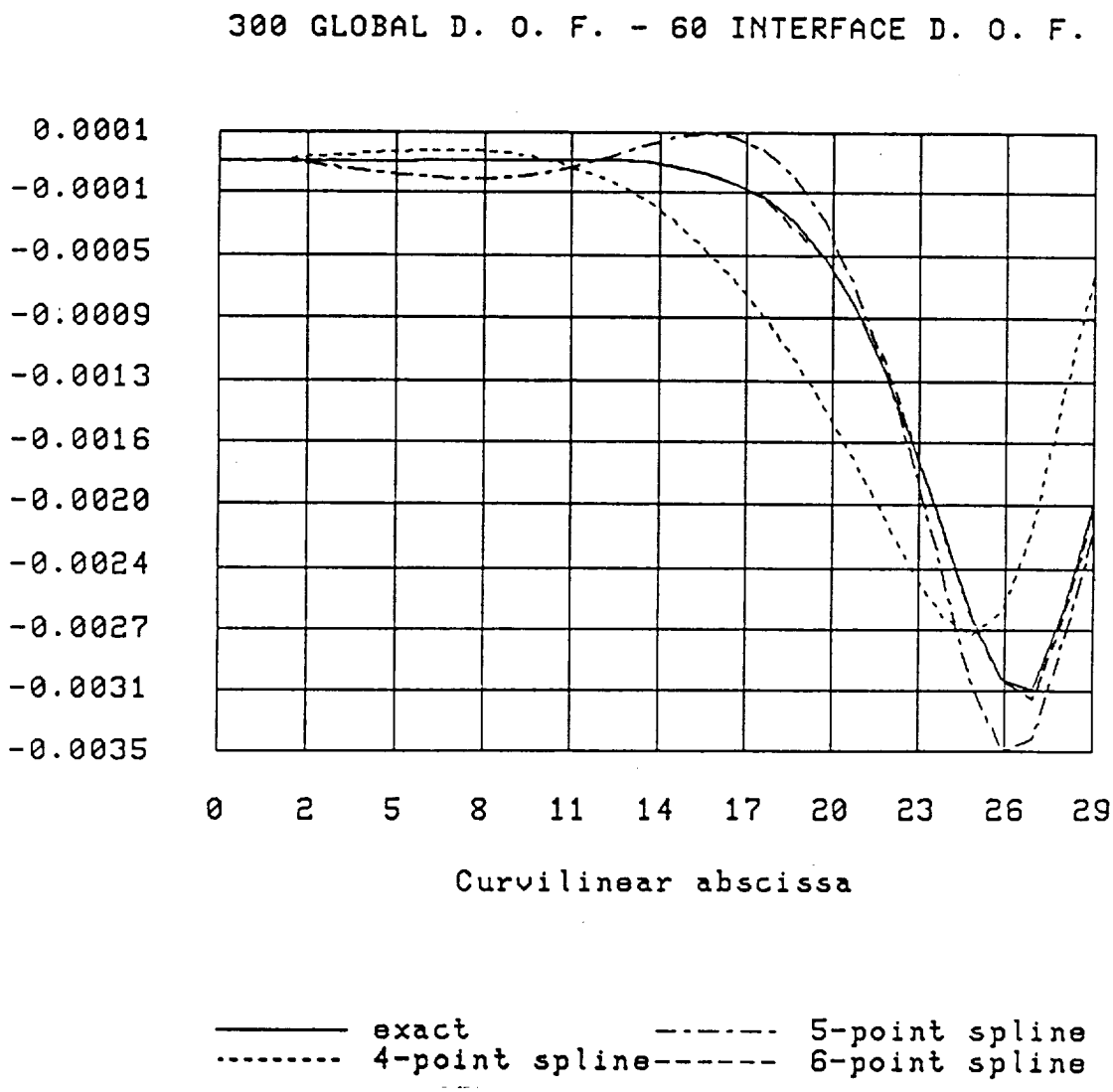


FIG. 12 *Cantilever problem: predicted horizontal displacement field*

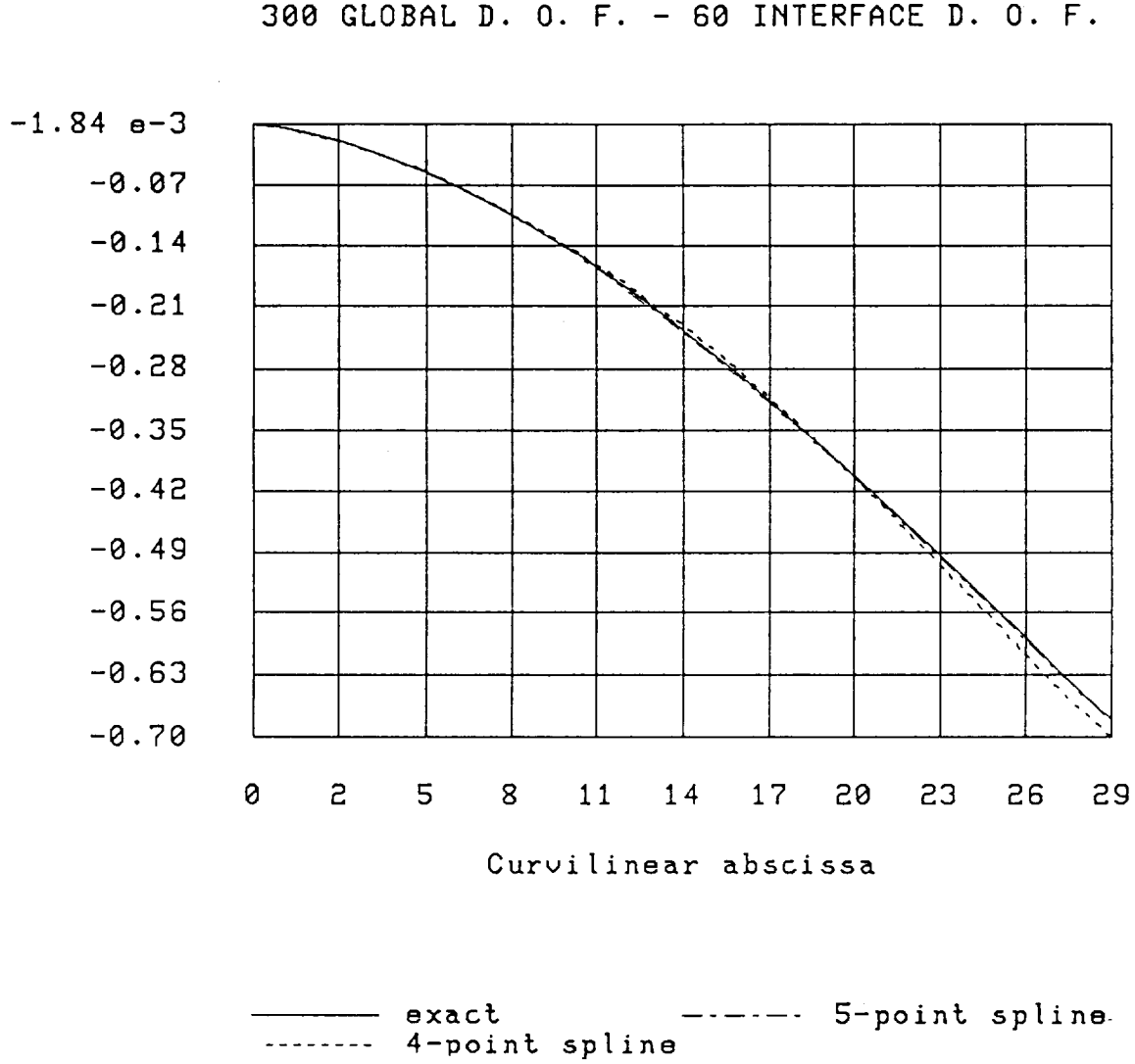


FIG. 13 *Cantilever problem: predicted vertical displacement field*

For the above problem, Figure 14 compares the condition numbers of $\mathbf{B}_1\mathbf{B}_1^T$ for various values of N_λ, when the traction forces are approximated with polynomials and piecewise low order polynomials. The advantage of the latter approximation is clearly demonstrated.

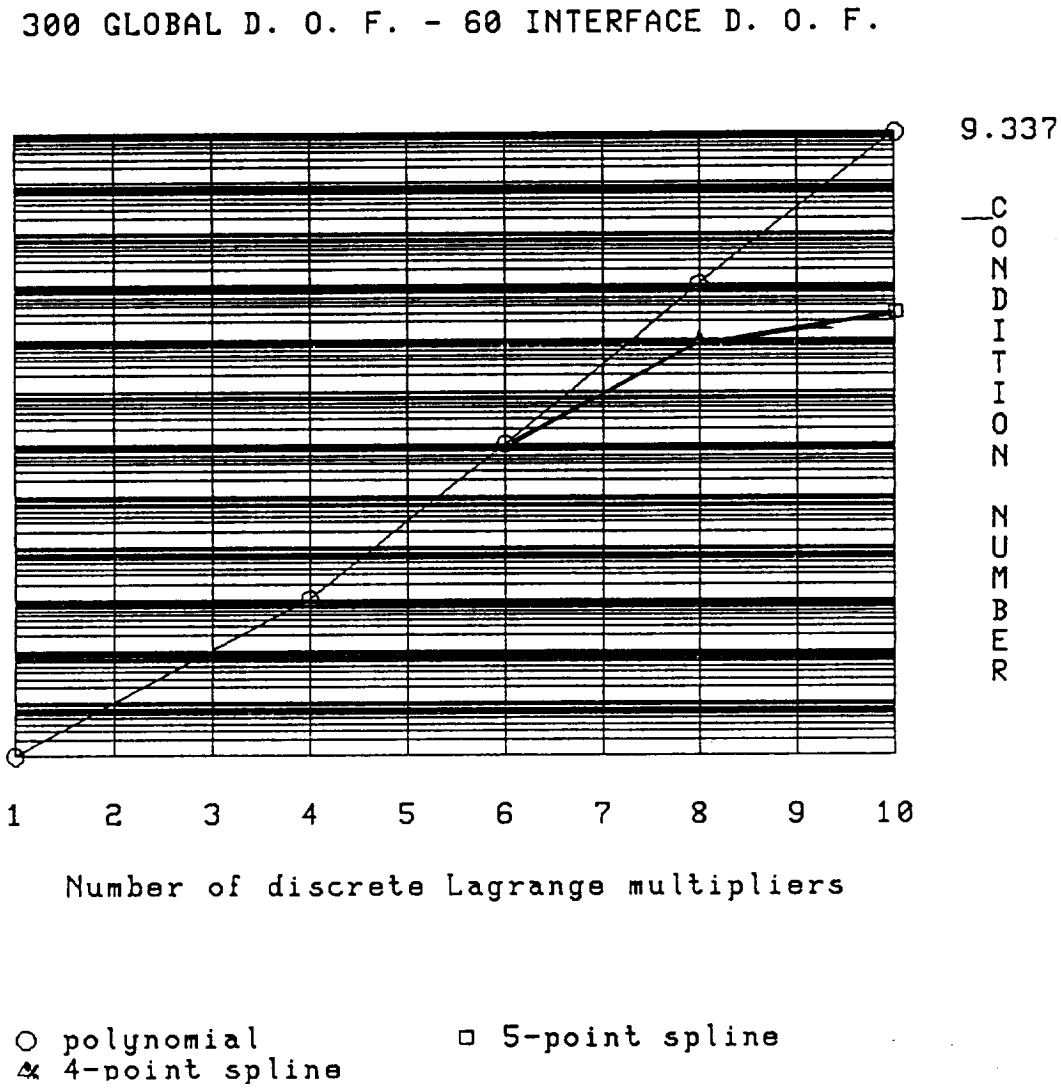


FIG. 14 *Conditioning of the constraint matrices*

6.2 Performance evaluation

Here we report on the performance of the proposed computational algorithm for a large-scale structural problem. The corresponding parallel/vector code is implemented on a CRAY Y-MP multiprocessor. Even though this system accommodates up to 8 processors, only 4 CPUs were available to us.

We consider the solution of the system of equations arising in the linear static analysis of the SRB when loaded by internal pressure in its Solid Rocket Motor (SRM) subsystem. The discretized SRB model has 10,453 elements, 9,206 nodes and 54,870 degrees of freedom (FIG. 15). After node-renumbering, the average profile bandwidth is 310. The finite element mesh is decomposed in 4 subdomains, each with approximately 2,613 elements. The decomposition is carried out along the longitudinal direction of the structure, using one-way separators only. This restriction will be removed in future developments. The optimized average profile bandwidth for each of the 4 subdomains is 91. Each of the 4 separators include approximately 920 degrees of freedom. The size of the interface problem is 3692. The tolerance ϵ for the convergence criterion (41) is set to 10^{-4}. Given the size of the interface problem, a rather large number of discrete Lagrange multipliers is anticipated. Therefore, the piecewise low order polynomial approximation switch is activated and the preconditioned *projected* conjugate gradient algorithm described in reference [1] is invoked for the solution of the interface problem. The hybrid algorithm achieves convergence after 3 iterative refinement steps with $N_\lambda = 283$. The computed results are compared with those generated by a parallel/vector skyline solver for validation. Table 1 below reports the CPU timings for the proposed algorithm and compares them with those of the fastest solutions that have been published for this problem (Storaasli, Nguyen and Agarwal [9], Farhat [10]). Clearly, the proposed algorithm is shown to be significantly faster in both serial and parallel environments.

FIG. 15 *Finite element discretization of the SRB*

TABLE 1. *Equation solving on the CRAY Y-MP*

SRB structural model - 54,870 d.o.f.

Number of processors	CPU time Skyline solver	CPU time Tearing Hybrid Algorithm
1	39 secs	20.18 secs
2	19.79 secs	10.21 secs
4	10 secs	5.19 secs

7. Conclusion

Recently, Farhat and Roux [1] have developed a domain decomposition algorithm based on a hybrid variational principle, for the parallel finite element solution of self-adjoint elliptic partial differential equations. First, the spatial domain was partitioned into a set of totally disconnected subdomains and an incomplete finite element solution was computed in each of these subdomains. Next, a set of Lagrange multipliers representing surface tractions were introduced at each degree of freedom of the discretized binding interface in order to enforce compatibility constraints between the independent local finite element approximations. For structural and mechanical problems, the resulting algorithm was shown to outperform the conventional method of substructures, especially on parallel processors. In this work, we have investigated the use of a much lower number of Lagrange multipliers, N_λ, for interconnecting the incomplete field finite element solutions. For that purpose, we have derived finite element procedures for both global and piecewise low order polynomial approximations of the interface tractions. Through simple structural examples, we have shown that a high accuracy can be reached with a value of N_λ that is only a small percentage of the total number of interface degrees of freedom. With this modification, the performance of the hybrid algorithm presented in [1] is drastically improved since it deals with a much smaller interface or reduced system. Even though we have addressed only the two-subdomain decomposition, the procedure is readily applicable to many-subdomain problems where only one-way separators are used for the mesh decomposition. We have illustrated the latter case with the large-scale static analysis of the Solid Rocket Booster (SRB) on a 4 processor CRAY Y-MP. For that problem, the modified hybrid algorithm is shown to outperform parallel skyline solvers in both serial and parallel environments. Future work will focus on the case of arbitrary mesh decompositions and on time dependent problems.

Appendix A. Piecewise-cubic Bessel interpolation

Let Γ_I^k, $k = 0, ..., N_\lambda/d - 2$ denote a partition $\mathcal{P}$ of the interface boundary Γ_I defined as:

$$\Gamma_I^k = [s_k, s_{k+1}] \qquad k = 0, ..., N_\lambda/d - 2$$

Within each subinterval Γ_I^k, d cubic polynomials are defined as:

$$\begin{aligned}
\tilde{\lambda}_k^1(s) &= c_{1\,k}^1 + c_{2\,k}^1(s-s_k) + c_{3\,k}^1(s-s_k)^2 + c_{4\,k}^1(s-s_k)^3 \\
\tilde{\lambda}_k^2(s) &= c_{1\,k}^2 + c_{2\,k}^2(s-s_k) + c_{3\,k}^2(s-s_k)^2 + c_{4\,k}^2(s-s_k)^3 \\
&\cdot \\
&\cdot \\
&\cdot \\
\tilde{\lambda}_k^d(s) &= c_{1\,k}^d + c_{2\,k}^d(s-s_k) + c_{3\,k}^d(s-s_k)^2 + c_{4\,k}^d(s-s_k)^3
\end{aligned}$$

The coefficients $c_{i\,k}^j$, $i = 1, ..., 4$, $j = 1, ..., d$ are determined by imposing:

$$\begin{aligned}
\tilde{\lambda}_k^j(s_k) &= \lambda_k^j \quad ; \quad \tilde{\lambda}_k^j(s_{k+1}) = \lambda_{k+1}^j \\
\frac{d\tilde{\lambda}_k^j}{ds}(s_k) &= \frac{d\lambda^j}{ds}(s_k) \; ; \; \frac{d\tilde{\lambda}_k^j}{ds}(s_{k+1}) = \frac{d\lambda^j}{ds}(s_{k+1}) \\
\frac{d\tilde{\lambda}_k^j}{ds}(s_k) &= \frac{\frac{\Delta s_{k-1}}{\Delta s_k}(\lambda_{k+1}-\lambda_k) + \frac{\Delta s_k}{\Delta s_{k-1}}(\lambda_k - \lambda_{k-1})}{\Delta_2 s_k} \\
k &= 0, ..., N_\lambda - 2 \\
j &= 1, ..., d
\end{aligned}$$

where Δs_k and $\Delta_2 s_k$ are defined as:

$$\begin{aligned}
\Delta s_k &= s_{k+1} - s_k \\
\Delta_2 s_k &= s_{k+1} - s_{k-1}
\end{aligned}$$

The solution of the above equations yield:

$$\begin{aligned}
c_{1\,k}^j &= \lambda_k^j \\
c_{2\,k}^j &= \xi_{2k}\lambda_{k+1}^j + \eta_{2k}\lambda_k^j + \zeta_{2k}\lambda_{k-1}^j \\
c_{3\,k}^j &= \xi_{3k}\lambda_{k+2}^j + \eta_{3k}\lambda_{k+1}^j + \zeta_{3k}\lambda_k^j + \nu_{3k}\lambda_{k-1}^j \\
c_{4\,k}^j &= \xi_{4k}\lambda_{k+2}^j + \eta_{4k}\lambda_{k+1}^j + \zeta_{4k}\lambda_k^j + \nu_{4k}\lambda_{k-1}^j
\end{aligned}$$

where

$$
\begin{aligned}
\xi_{2k} &= \frac{\Delta s_{k-1}}{\Delta s_k \Delta_2 s_k} \\
\eta_{2k} &= \frac{\Delta s_k}{\Delta s_{k-1} \Delta_2 s_k} - \frac{\Delta s_{k-1}}{\Delta s_k \Delta_2 s_k} \\
\zeta_{2k} &= \frac{-\Delta s_k}{\Delta s_{k-1} \Delta_2 s_k} \\
\xi_{3k} &= \frac{-1}{\Delta s_{k+1} \Delta_2 s_{k+1}} \\
\eta_{3k} &= \frac{3}{\Delta s_k^2} - \frac{\Delta s_{k-1}}{\Delta s_k^2 \Delta_2 s_k} + \frac{1}{\Delta s_{k+1} \Delta_2 s_{k+1}} - \frac{\Delta s_{k+1}}{\Delta s_k^2 \Delta_2 s_{k+1}} - \frac{\Delta s_{k-1}}{\Delta s_k^2 \Delta_2 s_k} \\
\zeta_{3k} &= \frac{-3}{\Delta s_k^2} - \frac{2}{\Delta s_{k-1} \Delta s_k} + \frac{\Delta s_{k-1}}{\Delta s_k^2 \Delta_2 s_k} + \frac{\Delta s_{k+1}}{\Delta s_k^2 \Delta_2 s_{k+1}} + \frac{\Delta s_{k-1}}{\Delta s_k^2 \Delta_2 s_k} \\
\nu_3 k &= \frac{2}{\Delta s_{k-1} \Delta_2 s_k} \\
\xi_{4k} &= \frac{1}{\Delta s_k \Delta s_{k+1} \Delta_2 s_{k+1}} \\
\eta_{4k} &= \frac{-1}{\Delta s_k \Delta s_{k+1} \Delta_2 s_{k+1}} + \frac{\Delta s_{k+1}}{\Delta s_k^3 \Delta_2 s_{k+1}} + \frac{\Delta s_{k-1}}{\Delta s_k^3 \Delta_2 s_k} - \frac{2}{\Delta s_k^3} \\
\zeta_{4k} &= \frac{-\Delta s_{k+1}}{\Delta s_k^3 \Delta_2 s_{k+1}} - \frac{\Delta s_{k-1}}{\Delta s_k^3 \Delta_2 s_k} + \frac{1}{\Delta s_k \Delta s_{k-1} \Delta_2 s_k} + \frac{2}{\Delta s_k^3} \\
\nu_{4k} &= \frac{-1}{\Delta s_{k-1} \Delta s_k \Delta_2 s_k}
\end{aligned}
$$

Acknowledgments

The first author would like to thank C. Militello at the Center for Space Structures and Controls, Boulder, for his valuable suggestions. He also acknowledges partial support by the National Science Foundation under Grant ASC-8717773, partial support by NASA Langley under Grant NAGI-756, and partial support by the Air Force Office of Scientific Research under Grant AFOSR-89-0422.

References

[1] C. Farhat and F. X. Roux, "A Method of Finite Element Tearing and Interconnecting and its Parallel Solution Algorithm," *(submitted for publication)*

[2] B. Nour-Omid, A. Raefsky and G. Lyzenga, "Solving Finite Element Equations on Concurrent Computers," *Parallel Computations and Their Impact on Mechanics*, ed. by A. K. Noor, ASME, New York, (1987), pp. 209-228.

[3] C. Farhat and E. Wilson, "A New Finite Element Concurrent Computer Program Architecture," *Int. J. Num. Meth. Eng.*, Vol. 24, No. 9, (1987), pp. 1771-1792.

[4] G. Kron, "A Set of Principles to Interconnect the Solutions of Physical Systems," *J. Applied Physics*, Vol. 24, No. 8, (1953), pp. 965-980.

[5] M. R. Dorr, "Domain Decomposition via Lagrange Multipliers," *UCRL-98532*, Lawrence Livermore National Laboratory, (1988).

[6] T. H. H. Pian, "Finite Element Formulation by Variational Principles with Relaxed Continuity Requirements," in *The Mathematical Foundation of the Finite Element Method with Applications to Partial Differential Equations*, Part II, ed. by A. K. Aziz, Academic Press, London, (1972), pp. 671-687.

[7] O. C. Zienkiewicz and R. L. Taylor, "The Finite Element Method," Fourth Edition, Volume 1, McGraw-Hill, (1989), pp. 373-396.

[8] S. D. Conte and C. De Boor, "Elementary Numerical Analysis, An Algorithmic Approach," McGraw-Hill, N. Y., (1980), pp. 288-289.

[9] O. O. Storaasli, D. T. Nguyen and T. K. Agarwal, "Parallel-Vector Solution of Large-Scale Structural Analysis Problems on Supercomputers," *AIAA / ASME / ASCE / AHS / ASC 30th Structures*, Structural Dynamics and Materials Conference, Mobile, Alabama, (1989), pp. 859-867.

[10] C. Farhat, "Which Parallel Finite Element Algorithm for Which Architecture and Which Problem," *Computational Structural Mechanics and Multidisciplinary Optimization*, ed. by R. V. Grandhi, W. J. Stroud and V. B. Venkayya, ASME, AD-Vol. 16, (1989), pp. 35-43.