

this document downloaded from

vulcanhammer.net

Since 1997, your complete
online resource for
information geotechnical
engineering and deep
foundations:

The Wave Equation Page for
Piling

*Online books on all aspects of
soil mechanics, foundations and
marine construction*

Free general engineering and
geotechnical software

And much more...

Terms and Conditions of Use:

All of the information, data and computer software ("information") presented on this web site is for general information only. While every effort will be made to insure its accuracy, this information should not be used or relied on for any specific application without independent, competent professional examination and verification of its accuracy, suitability and applicability by a licensed professional. Anyone making use of this information does so at his or her own risk and assumes any and all liability resulting from such use. The entire risk as to quality or usability of the information contained within is with the reader. In no event will this web page or webmaster be held liable, nor does this web page or its webmaster provide insurance against liability, for any damages including lost profits, lost savings or any other incidental or consequential damages arising from the use or inability to use the information contained within.

This site is not an official site of Prentice-Hall, Pile Buck, the University of Tennessee at Chattanooga, or Vulcan Foundation Equipment. All references to sources of software, equipment, parts, service or repairs do not constitute an endorsement.

**Visit our
companion site**

<http://www.vulcanhammer.org>



CIVIL ENGINEERING STUDIES

STRUCTURAL RESEARCH SERIES NO. 571



AD-A260 716



ISSN: 0069-4274

APPLICATIONS OF PARALLEL AND VECTOR ALGORITHMS IN NONLINEAR STRUCTURAL DYNAMICS USING THE FINITE ELEMENT METHOD

DTIC
ELECTE
JAN 26 1993
S c D

By
BRIAN E. HEALY
DAVID A. PECKNOLD
ROBERT H. DODDS, JR.

A Report on a Research Project Sponsored by
the Department of Civil Engineering and
the Center for Supercomputing Research
and Development (CSR)

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

DEPARTMENT OF CIVIL ENGINEERING
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN
URBANA, ILLINOIS
SEPTEMBER 1992

116
PP
93-01358



93 1 25 120

REPORT DOCUMENTATION PAGE	1. REPORT NO. UILU-ENG-92-2011	2.	3. Recipient's Accession No.
4. Title and Subtitle Applications of Parallel and Vector Algorithms in Nonlinear Structural Dynamics Using the Finite Element Method		5. Report Date September 1992	
		6.	
7. Author(s) B.E. Healy, D.A. Pecknold, and R.H. Dodds, Jr.		8. Performing Organization Report No. SRS 571	
9. Performing Organization Name and Address University of Illinois at Urbana-Champaign Department of Civil Engineering 205 N. Mathews Avenue Urbana, Illinois 61801		10. Project/Task/Work Unit No.	
		11. Contract(C) or Grant(G) No.	
12. Sponsoring Organization Name and Address Department of Civil Engineering and Center for Supercomputing Research and Development (CSRD)		13. Type of Report & Period Covered	
		14.	
15. Supplementary Notes			
16. Abstract (Limit: 200 words) <p>This research is directed toward the numerical analysis of large, three dimensional, nonlinear dynamic problems in structural and solid mechanics. Such problems include those exhibiting large deformations, displacements, or rotations, those requiring finite strain plasticity material models that couple geometric and material nonlinearities, and those demanding detailed geometric modeling.</p> <p>A finite element code was developed, designed around the 3D isoparametric family of elements, and using a Total Lagrangian formulation and implicit integration of the global equations of motion. The research was conducted using the Alliant FX/8 and Convex C240 supercomputers.</p> <p>The research focuses on four main areas: Development of element computation algorithms that exploit the inherent opportunities for concurrency and vectorization present in the finite element method; Comparison of the preconditioned conjugate gradient method to a representative direct solver; Investigation of various nonlinear solution algorithms, such as modified Newton-Raphson, secant-Newton, and nonlinear preconditioned conjugate gradient; and, Discovery of an accurate, robust finite strain plasticity material model.</p>			
17. Document Analysis a. Descriptors Parallel Computing, Vector Algorithms, Conjugate Gradient, Newton Raphson, Newmark Beta Method, Nonlinear, Structural Dynamics, Implicit, Finite Element, Plasticity b. Identifiers/Open-Ended Terms c. COSATI Field/Group			
18. Availability Statement Release Unlimited	19. Security Class (This Report) UNCLASSIFIED		21. No. of Pages 310
	20. Security Class (This Page) UNCLASSIFIED		22. Price

APPLICATIONS OF PARALLEL AND VECTOR ALGORITHMS
IN NONLINEAR STRUCTURAL DYNAMICS
USING THE FINITE ELEMENT METHOD

by

Brian E. Healy
David A. Pecknold
Robert H. Dodds Jr.

DATA QUALITY INDICATED 5

Urbana, Illinois
September, 1992

Accession For	
NTIS GR&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A-1	

ACKNOWLEDGEMENTS

This report is based on the dissertation of Dr. Brian E. Healy, which was submitted in June 1991 to the Graduate College of the University of Illinois at Urbana-Champaign in partial fulfillment of the requirements for the Ph. D. degree in Civil Engineering.

The research was conducted under the joint supervision of Dr. David A. Pecknold and Dr. Robert H. Dodds, Jr., Professors of Civil Engineering. Financial assistance and support was provided by the Department of Civil Engineering and by the Center for Supercomputing Research and Development (CSR D) under grant SCCA 90-82144 from the Illinois Department of Commerce and grant DE-FG02-85ER25001 from the Department of Energy.

The authors are indebted to the Department of Civil Engineering for use of the HP/Apollo network, to the Department of Computer Science for the use of the Convex C240, and to CSR D for the use of the Alliant FX/8. These computing facilities were crucial to the completion of this research.

TABLE OF CONTENTS

1 INTRODUCTION	1
1.1 PROBLEM DEFINITION	1
1.2 PRIMARY RESEARCH AREAS	4
1.3 OUTLINE	11
2 THEORY AND APPLICATION	16
2.1 FINITE ELEMENT FORMULATION	15
2.2 LINEAR AND NONLINEAR SOLUTION ALGORITHMS	24
2.3 ELEMENT COMPUTATION ALGORITHMS	46
2.4 MATERIAL MODEL	64
3 NUMERICAL RESULTS: FINITE ELEMENT ANALYSES	94
3.1 BAR EXTENSION PROBLEM	94
3.2 SHEAR BAND PROBLEM	103
3.3 3D BEND BAR PROBLEM	110
3.4 CYLINDRICAL SHELL PROBLEM	119
3.5 MATERIAL MODEL	128
4 NUMERICAL RESULTS: TIMINGS	149
4.1 PROBLEM DESCRIPTIONS	150
4.2 LINEAR SOLUTION ALGORITHMS	156
4.3 ELEMENT COMPUTATION ALGORITHMS	186
4.4 NONLINEAR SOLUTION ALGORITHMS	215
5 CONCLUSION	232
5.1 ELEMENT COMPUTATION ALGORITHMS	232
5.2 LINEAR SOLUTION ALGORITHMS	234
5.3 NONLINEAR SOLUTION ALGORITHMS	237
5.4 MATERIAL MODEL	237

5.5 IMPROVEMENTS AND EXTENSIONS	239
BIBLIOGRAPHY	249
APPENDIX A	257
APPENDIX B	263
VITA	311

1 INTRODUCTION

In the following, the purpose and goals of this work are described, the research areas which comprise the primary focus of the thesis are described, and the main body of the thesis is outlined. Throughout the introduction and the remainder of the thesis, references to previous work are cited as they are discussed in context.

1.1 PROBLEM DEFINITION

This research is directed toward the analysis of large, three dimensional, nonlinear dynamic problems in structural and solid mechanics. Such problems include those exhibiting large deformations, displacements, or rotations, those requiring finite strain plasticity material models to couple these geometric nonlinearities with material nonlinearities, and those demanding detailed geometric modeling. The finite element method is employed to model these problems. Examples of applications employing such finite element models are

- Vehicle crashworthiness
- Offshore structures
- Thin walled structures
- Seismic structure response
- Fracture mechanics and crack propagation
- Metal forming
- Composite materials

Advances in computer technology and the advent of supercomputers have made available to analysts large amounts of main memory and the ability to perform calculations at very high speeds. These advances allow the capabilities of implicit methods for structural dynamics to be realistically explored. The implicit formulation also provides a unified approach within which both nonlinear static and dynamic structural problems can be addressed.

Ultimately, this research targets the solution of very large problems, which can contain on the order of 100,000 degrees of freedom and half band widths of perhaps 10,000. Such problems can require many gigabytes of memory if a direct solver is used to solve a linear system of equations within a nonlinear equilibrium iteration; unfortunately the machines used in this research, the Convex C240 and primarily the Alliant FX/8, either do not posses

such large memory capacity or it is generally unavailable. Consequently, smaller problems which can be analyzed by the Alliant FX/8 are utilized as example problems in this thesis. These example problems reach sizes of more than 10,000 degrees of freedom and upwards of 2500 elements and are large for the Alliant. Thus, these example problems provide a microcosm of larger problems that can be solved on more powerful supercomputers and allow representative trends to be observed.

NLDFEP, a NonLinear Dynamic Finite Element Program, is designed around the three dimensional isoparametric family of elements. Isoparametric elements are attractive because of their flexible geometric modeling capability and their systematic approach to element computations. The modeling capability results from the ability to take a basic shape and orientation in parametric space and through interpolation of nodal Cartesian coordinates transform it to a deformed shape and arbitrary orientation in Cartesian space. Isoparametric elements can be approached systematically because it is necessary to integrate element equations numerically, creating a natural hierarchy of computations. The two elements currently implemented in NLDFEP are the 8 and 20 node bricks. The program is structured so that additional elements, such as the 27 node brick or another family of parametric elements can be incorporated into the program without difficulty.

Many supercomputers, such as the Alliant FX/8, employ optimizing compilers that take an existing code and optimize it for concurrency and vectorization. These compilers are capable of vectorizing loops and providing a low level of concurrency. Unfortunately, these optimizing compilers are not sufficient for dealing with complications such as data dependencies and are not equipped to recognize opportunities to restructure code to achieve a higher level of concurrency and a consistently efficient vector length. Because of this, existing finite element code written for sequential computers is inefficient when executed on supercomputers even with compiler optimization. In order to take full advantage of the computational power of a supercomputer, special element computation algorithms must be written to exploit the opportunities for designed vectorization and high level concurrency available in the finite element method. These opportunities are ample, especially when dealing with the computational hierarchy generated by the use of the isoparametric element formulation.

As noted above, the solution of a linear system of equations is generally required for each equilibrium iteration in a nonlinear analysis, whether it be static or dynamic. Traditionally, a direct solver has been used to perform this linear solution. However, given the memory requirements and computational effort characteristic of a direct solver, iterative methods may be more appropriate for large problems solved on supercomputers. Specifically, the preconditioned conjugate gradient method, due to its reliance on vector and matrix-

vector operations and the fact that it is not necessary to assemble the global stiffness matrix, is considered as an alternative to a direct solver. The stiffness matrix can be stored by elements, which mitigates the amount of memory usage and allows the matrix-vector operations to be performed in an element-by-element fashion that utilizes optimized element computation algorithms, and the vector operations are ideally suited to the capabilities of a supercomputer.

As well as discovering an efficient linear solution algorithm, an efficient nonlinear solution algorithm must also be found. Such a nonlinear solution algorithm must be able to handle combined material and geometric nonlinearities effectively. Algorithms which seem worthy of consideration are modified Newton-Raphson, some form of quasi-Newton, and a nonlinear implementation of the preconditioned conjugate gradient method.

Advances in computer technology have made practical the consideration of plasticity in conjunction with large strains and finite rotations. So that this may be accomplished, an effective finite strain plasticity material model that behaves correctly from the standpoint of theoretical and computational engineering mechanics is developed. This material model must also be suitable to the implicit dynamic formulation of NLDPE so that it accounts for the computational power available and the greater computational complexity of implicit dynamics, and should be compatible with a small strain plasticity model in the case that only material nonlinearities are present in the problem.

The preceding paragraphs highlight the four primary areas of research addressed in this thesis. Developments in these areas combine to allow the supercomputer analysis of large, three dimensional, dynamic structural engineering problems exhibiting complex behavior. These areas are

- 1) The development of element computation algorithms so that the inherent opportunities for concurrency and vectorization present in the finite element method can be exploited.
- 2) The comparison of the preconditioned conjugate gradient method to a representative direct solver.
- 3) The investigation of various nonlinear solution algorithms.
- 4) The discovery of an accurate and robust finite strain plasticity material model so that material nonlinearities can be evaluated both in the presence and the absence of geometric nonlinearities.

1.2 PRIMARY RESEARCH AREAS

In this section, the four primary areas of research addressed by this thesis are discussed in some depth. These research areas are the development of effective element computation algorithms, linear solution algorithms, nonlinear solution algorithms, and finite strain plasticity material models.

1.2.1 ELEMENT COMPUTATION ALGORITHMS

A readily exploitable computational hierarchy is created by the nature of the finite element method itself and the fact that for the isoparametric formulation it is necessary to evaluate element integrals numerically. This hierarchy spans three levels: computations flow from the structure level to the element level and finally to the material point level. As long as there is nonconflicting access to the appropriate data, calculations at the element level and subsequently calculations at the material point level can take place independently, allowing excellent opportunities for concurrency and vectorization.

Element computation algorithms are designed to capitalize on this computational hierarchy in ways that are optimal for the architecture of a given machine. Fundamental element computations, such as the calculation of the internal force vector, the tangent stiffness matrix, the tangent mass matrix, and the stress recovery, which encompass all three levels of the hierarchy are subject to the element computation algorithms. Further, computations traversing only the structure and element levels of the hierarchy, such as the matrix-vector product critical to the success of the linear preconditioned conjugate gradient algorithm, also fall under the auspices of the element computation algorithms. The machines considered in this thesis are capable of both concurrency and vectorization, and the element computation algorithms manipulate these two attributes in various ways to discover the most effective combination on a given machine. Hughes et al [39] applied a block element vectorization algorithm, where vectorization takes place across a block of simultaneously processed elements with the vector lengths equal to the number of elements in the block, on a Cray X-MP/48. However, no provision is made for concurrency in this algorithm, and it is applied only to linear isoparametric elements. These observations lead to many new questions, such as

- For parallel-vector machines such as the Alliant FX/8, which rely heavily on concurrency, should the primary emphasis be on vectorization or would an algorithm emphasizing concurrency be more efficient? What about parallel-vector machines like the Convex C240, which depend primarily on vectorization?
- How would the element computation algorithms perform when applied to quadratic isoparametric elements, which require significantly more computational effort than

linear isoparametric elements ? How does the increase in element data requirements affect the algorithms ? How do the natural vector lengths inherent in quadratic, and perhaps linear, isoparametric elements combined with higher level concurrency compare with designing vectorization ?

Element computations form the foundation of the finite element method, and should be optimized if supercomputers are to be fully exploited. Because the Alliant FX/8 and the Convex C240 are representative of the spectrum of supercomputers which utilize both concurrency and vectorization, the answers to the above questions should be applicable to other supercomputers.

1.2.2 LINEAR SOLUTION ALGORITHMS

The capabilities of the linear preconditioned conjugate gradient algorithm and a representative direct solver in terms of memory usage and computational efficiency are explored in this work. There is a limit to the amount of main and virtual memory available on a given machine and if the memory requirements of a given linear solver exceed this amount an analysis would not be possible. This is a particular concern on the Alliant FX/8, which is the primary machine used in this research. Because a direct solver requires storage of a global stiffness matrix in banded or other format, as the size and dimensionality of a given problem increase the memory space demanded by a direct solver can quickly become prohibitive. The linear preconditioned conjugate gradient method does not require storage of the global stiffness, just the element stiffness matrices. This can represent a tremendous savings in memory, enough to provide the difference between a successful solution and one that exceeds the capacity of the machine. Even if the memory usage of the direct solver does not exceed this capacity, the amount of virtual memory paging during the factorization and solution can severely hamper the efficiency of these operations and nullify the power of a supercomputer.

For large stiffness matrices, the computational efficiency of a direct solver can drop significantly, apart from the memory considerations. Given the large half band widths and number of equations characteristic of these matrices, calculations with large vector lengths can result which, when performed repeatedly and considering concurrency, can inefficiently utilize the cache memory of machines like the Alliant FX/8. Of course, the sheer computational effort required by a direct solver is the critical drawback when applied to large problems. Use of the linear preconditioned conjugate gradient method can greatly reduce the required computational effort, and the algorithm contains operations, such as a matrix-vector product, which can be performed with highly optimized element computation algorithms.

This can translate into a great savings in execution time and more effectively exploit the capabilities of a supercomputer.

The principal preconditioners employed in this research are the diagonal and element-by-element [39] preconditioners. The diagonal preconditioner, consisting in this case of the main diagonal of the global dynamic tangent stiffness matrix, is the simplest and least computationally intensive preconditioner and one which is most generally employed. The element-by-element preconditioner is more complex, but its more involved computations may be more than compensated by a reduced number of performed linear iterations. Additionally, the element-by-element formulation is implemented with the highly optimized element computation algorithms and represents a natural extension of the finite element method. A third preconditioner, the hierarchical preconditioner [13],[18] was considered, but difficulties were encountered during its implementation, and since it is only applicable to quadratic isoparametric elements, it was not pursued. An effective preconditioner is critical to performance of the linear preconditioned conjugate gradient method, and the evaluation of the above preconditioners is an important aspect of this research.

The performance of the linear preconditioned conjugate gradient method, using both the diagonal and element-by-element preconditioners, is compared in reference [39] to a direct solver in the production nonlinear stress analysis code NIKE3D on the Cray X-MP/48. This comparison deals solely with linear isoparametric elements, and many questions remain unanswered. Among these questions are

- How is the comparison affected by the use of quadratic isoparametric elements ? How is the comparison affected by the presence of concurrency ? By combined material and geometric nonlinearities ? Exactly how and why do three dimensional elements applied to one or two dimensional meshes exhibit poor performance for the linear preconditioned conjugate gradient method, as reported by Hughes ?
- What is the influence of dynamic effects ? How does varying the size of the time step affect the performance of the linear preconditioned conjugate gradient method ?

These questions and others should be answered and the comparison of the linear solvers should be completed if large, nonlinear dynamic finite element solutions are to be performed efficiently on supercomputers.

1.2.3 NONLINEAR SOLUTION ALGORITHMS

The benchmark nonlinear solution algorithm employed in this research is modified Newton-Raphson. This algorithm is chosen because of its widespread use in structural

engineering codes [1],[70], and because of the quadratic convergence rates characteristic of the algorithm. It provides a solid base for comparison of the nonlinear solution algorithms under combined material and geometric nonlinearities. The main computational expense of the modified Newton-Raphson method is the update of the dynamic tangent stiffness matrix, and the factorization of the global dynamic tangent stiffness if a direct solver is specified. Of course, in modified Newton-Raphson it is not necessary to update the dynamic tangent stiffness every equilibrium iteration, although for a highly nonlinear analysis more frequent updates are often required to insure an adequate convergence rate. This update falls within the province of the highly optimized element computation algorithms, so that modified Newton-Raphson stands to benefit from the capabilities of a supercomputer.

Quasi-Newton methods are the extrapolation of the one degree of freedom secant method for finding the roots of an equation [19]. In quasi-Newton, an approximate dynamic tangent stiffness matrix at the beginning of a time step is updated by satisfying a secant relationship at each equilibrium iteration in order to obtain a better approximation. This approximation approaches the true tangent stiffness in the limit of convergence. The updating can be accomplished through a series of vector operations suitable for supercomputers and can be intertwined with the solution of the linear set of equations [53]. These operations rely heavily on the calculation of the internal force vector and the stress recovery [31], which are performed with the highly optimized element computation algorithms. Also, only one computation of the dynamic tangent stiffness matrix is required per time step [31],[53] and if a direct solver is employed only one factorization of the global matrix is required per time step [31],[53] although storage of the global matrix would still be necessary, and one forward reduction and back substitution is combined with the series of vector operations above to produce the new search direction for each equilibrium iteration [53]. It is necessary to store the updating vectors, an additional two per equilibrium iteration, and if so many equilibrium iterations are performed that the number of updating vectors becomes too large, then the process can be restarted with a new approximation to the final dynamic tangent stiffness, which must be recalculated and factorized if a direct solver is used [31],[53]. Quasi-Newton is the nonlinear solution algorithm employed in reference [39].

Unfortunately, time constraints did not allow enough effort to be put into the development of a quasi-Newton algorithm to do it justice. Consequently, quasi-Newton was implemented in a qualitative sense by the adoption of the method termed secant-Newton, attributed to Crisfield [17],[19]. In this method, the secant relationship of quasi-Newton is satisfied by modifying the iterative displacement increment produced by modified Newton-Raphson. This represents an acceleration of modified Newton-Raphson and is accomplished

through a small number of vector operations suitable for supercomputers. Secant-Newton also has the advantage that it is very compatible with modified Newton-Raphson and can be incorporated as an easily implemented option into a modified Newton-Raphson approach.

As a nonlinear solution algorithm, the preconditioned conjugate gradient method is promising for use on supercomputers for a variety of reasons. As with the linear algorithm, it is composed mainly of vector operations. A key computational expense is the line search to find the step length necessary to force the calculated residual to be orthogonal to the current search direction. The line search requires repeated stress recovery and computation of the internal force vector. Since these calculations can be performed using the highly optimized element computation algorithms, it is possible that the power of a supercomputer may significantly mitigate the cost of the line search. The preconditioning step can be accomplished in one of two ways. First, some form of the global dynamic tangent stiffness matrix, such as the main diagonal, may be employed, eliminating the need for a complete solution of the linear set of equations [64]. However, the number of nonlinear equilibrium iterations thus required may be prohibitive, as for a highly nonlinear solution more frequent dynamic tangent stiffness updates are often necessary, in addition to the always mandatory line search. A second approach to the preconditioning step is to use the entire dynamic tangent stiffness matrix as the preconditioner. The nonlinear preconditioned conjugate gradient algorithm then resembles the conjugate-Newton method proposed by Irons and Elsawaf [42]. The difference between the two methods lies primarily in the manner that the search direction multiplier is calculated and consequently the manifestation of the global dynamic tangent stiffness matrix to which the search directions are orthogonal. The advantage of the nonlinear preconditioned conjugate gradient method under the second preconditioning approach resides in the possibility that far fewer global equilibrium iterations may be necessary, as compared to the related Newton-Raphson method, given the convergence characteristics of the conjugate gradient method that result from the enforcement of orthogonality and the line search. This would decrease the number of dynamic tangent stiffness updates, and global matrix factorizations if a direct solver is employed. An advantage of the first approach, dispensing with the need to assemble the full dynamic tangent stiffness matrix, can be realized in the second approach if the preconditioned conjugate gradient algorithm is used as the linear solver.

Each of the nonlinear solution algorithms discussed here have qualities and characteristics amenable to supercomputer analysis. Testing and evaluation is necessary to determine which of these algorithms performs best under combined material and geometric nonlinearities and makes best use of the resources of a supercomputer.

1.2.4 MATERIAL MODELS

The type of plasticity considered in this thesis is isotropic, rate independent J_2 flow theory including both isotropic and kinematic hardening. When combined with large strain and finite rotations, the situation is greatly complicated. Finite rotations of material axes, those axes attached to material points that deform and rotate with the continuum in a local sense, complicate the definition of stress-strain rates and their numerical integration to advance the material response over a time step. Traditionally, constitutive models for large strain plasticity in finite element codes are cast in a spatial setting which mandates use of an objective stress rate to remove that part of the total stress rate due to simple rigid rotation of the material. In a spatial setting, the components of Cauchy (true) stress are defined relative to a fixed, Cartesian system; thus rigid rotation alone alters the stress components. While numerous objective stress rates may be constructed [3], the Jaumann stress rate has been widely implemented in both explicit and implicit codes given its apparent simplicity [1],[48].

Over the past ten years, serious objections to constitutive models employing the Jaumann rate have developed as more complex material behaviour is considered, for example kinematic hardening and viscoplasticity, and as the magnitude of the deformations experienced in the applications has increased, up to plastic strains of 50 to 100%. The first objection addresses the increased complexity of numerical algorithms to accommodate the spatial setting; tensorial state variables within the plasticity model, for example the back stress in kinematic hardening, must also be expressed using an objective rate and modified to reflect rigid rotations. Processing of the purely kinematic effects due to finite rotations is thus interwoven with the integration of evolution equations for internal state variables. Consequently, development of each new material model potentially requires individual treatment of finite rotations. The second objection to use of the Jaumann rate concerns the physically unacceptable stresses predicted at large strains under certain conditions. The problem of finite simple shear illustrates this deficiency [20]. When an incremental, linear elastic material law is used to relate the Jaumann stress rate to the rate of deformation tensor in a fixed Cartesian system, the predicted Cauchy stresses oscillate in an unrealistic manner, with the shear stress actually reversing sign. Nagtegaal and de Jong [56] and Johnson and Baumann [44] noted such stress oscillations with kinematic hardening in elasto-plasticity for a material which strain hardens monotonically in tension. Atluri [3] later showed that similar oscillations exist for isotropic hardening unless the elastic strains are vanishingly small.

Atluri [3] demonstrated that removal of the oscillatory response in simple shear may be accomplished through definition of alternate stress rates or through a more general construction of the hypoelastic material law. In a desire to retain the simplest hypoelastic ma-

terial law as a direct generalization of the conventional small strain forms, Green and Naghdi [35] introduced an objective stress rate that has been discussed extensively by Dienes [20], Johnson and Baumann [44], and Atluri [3]. A Cauchy stress measure and its objective rate are defined on an unrotated orthogonal reference frame established through a polar decomposition of the deformation gradient at each material point. This constitutive model predicts monotonically increasing stresses in finite simple shear for incremental linear elasticity. Using this concept of an unrotated reference frame for constitutive modeling, Flanagan and Taylor [26] developed the PRONTO 2-D and 3-D [80],[81] codes for transient dynamic analysis with explicit time integration. Constitutive computations are performed using strains, stresses, and their objective rates defined on the unrotated reference frame. Effects of finite rotations are thus uncoupled from and transparent to the integration algorithms for stresses and material state variables, so that the numerical architecture of existing small strain plasticity models is fully retained. In their formulation, Flanagan and Taylor developed an efficient algorithm for the approximate evolution of the polar decomposition with time in the globally explicit solution.

In this research, the concept of the unrotated reference frame is implemented, and as this research concerns implicit time integration this implementation represents an extension of the work of Flanagan and Taylor. Additional requirements generated by implicit time integration are

- 1) An efficient and accurate algorithm for the polar decomposition that accounts for the large strain increments and fewer time steps characteristic of implicit methods. The approximate evolution of the polar decomposition developed by Flanagan and Taylor for explicit time integration may not be suitable for implicit methods.
- 2) An equally efficient and accurate scheme for the integration of the plasticity rate equations over the large strain increments. This scheme should contain the small strain plasticity model as a kernel around which the finite strain operations act.
- 3) A consistent tangent operator which maintains the desired quadratic convergence rates of the global equilibrium iterations. In keeping with the Total Lagrangian approach of this research, this tangent operator relates the rate of second Piola-Kirchhoff stress to the Green strain rate.

A formulation revolving around the Truesdell stress rate is also implemented in NLDFEP. Such a formulation also avoids the oscillatory response noted above for the Jaumann stress rate in finite simple shear employing incremental linear elasticity. The Truesdell stress rate formulation enjoys the advantage that no polar decompositions are required, but

labors under the burden of requiring that the initial Cauchy stresses, and the initial back stresses if kinematic hardening is specified, at the beginning of a time step must be transformed to reflect the nature of the Truesdell stress rate. It is thus more difficult to uncouple the small strain plasticity model from the finite strain effects. While requirement 1 listed above is not applicable to the Truesdell formulation, requirements 2 and 3 remain imperative.

In this thesis, both the unrotated and Truesdell formulations are tested and evaluated to discover which provides the most robust, accurate, and efficient finite strain plasticity model.

1.3 OUTLINE

This thesis is organized into seven chapters. The first chapter is, of course, the introduction. The remaining chapters are, in order, theory and application, numerical results of finite element analyses, numerical timing results, conclusions, bibliography, and appendices. In the balance of this section the contents of the remaining chapters are discussed.

1.3.1 THEORY AND APPLICATION

The chapter on theory and application contains four sections. In the first section, the foundation of the finite element method is developed from the standpoint of three dimensional isoparametric elements and geometric nonlinearity. Subsections include discussions on

- The equations of motion
- The finite element formulation, including geometric nonlinearity
- The determination of element quantities, including the internal force vector, the tangent stiffness matrix, and the tangent mass matrix

In the second section, the linear and nonlinear solution algorithms implemented in this research are presented and developed, as well as the basis of the implicit integration of the equations of motion. Subsections involve

- The Newmark Beta method for implicit time integration
- The modified Newton-Raphson and secant-Newton nonlinear solution algorithms, including convergence criteria and the direct solvers employed
- The linear preconditioned conjugate gradient algorithm, including a discussion of various preconditioners

- The nonlinear preconditioned conjugate gradient algorithm, including discussion of the line search algorithm and the calculation of the search direction multiplier

The third section concerns the development and implementation of the element computation algorithms. Subsections deal with

- The element computational hierarchy
- Parallel and vector algorithms, the interactions of vectorization and concurrency, gather/scatter operations, element blocking algorithms, and element data structures
- Possible element computation algorithms
- Element computation algorithms implemented in NLDFEP on both the Alliant FX/8 and the Convex C240

The final section in the theory and application chapter deals with the finite strain plasticity material models. The two finite strain formulations, based on the unrotated and Truesdell stress rates, and the small strain kernel are presented and developed. Subsections concern

- Definitions of stress and strain for finite deformations and rotations
- Stress rates and constitutive relationships
- The performance of the polar decomposition, including an exact and approximate approach and a numerical comparison of the two
- The small strain stress recovery and tangent operator
- The finite strain stress recovery and tangent operators for both the unrotated and Truesdell formulations

1.3.2 NUMERICAL RESULTS: FINITE ELEMENT ANALYSES

Numerical results are presented to establish the validity of NLDFEP as a nonlinear dynamic finite element program, and to explore the capabilities of the finite strain plasticity material models. Four example problems, which are also examined in the chapter on numerical timing results, are analyzed to demonstrate the efficacy of NLDFEP in various circumstances. These are the bar extension problem, the shear band problem, the 3D bend bar prob-

lem, and the cylindrical shell problem. The results of each of these analyses are discussed in separate sections. The final section in this chapter deals with the investigation of the finite strain plasticity models. Various problems are studied, including finite simple shear, finite extension, and severe blunting at a crack tip within a boundary layer small scale yield problem.

1.3.3 NUMERICAL RESULTS: TIMINGS

Timing results are presented based on the analysis of various example problems. The example problems represent a broad spectrum of behaviours, from one dimensional to three dimensional problems, employing both linear and quadratic three dimensional isoparametric elements, exhibiting curved and crack tip geometries, combining material and geometric nonlinearities, and ranging from static to highly dynamic problems. The example problems, the first four of which are treated in the finite element results chapter above, are the bar extension problem, the shear band problem, the 3D bend bar problem, the cylindrical shell problem, the 3D linear finite extension problem, and the 3D quadratic finite extension problem. The timing results are arranged in sections concerning descriptions of the performed timing runs, the linear solution algorithms, the element computation algorithms, and the nonlinear solution algorithms.

The timing runs are intended to be representative of the behaviour of each problem solved. In addition, they should be feasible to execute within the parameters of the available machine time on the Convex C240 and especially the Allaint FX/8. In the section describing the timing runs, subsections are provided which describe the slices of each example problem analyzed for the purposes of comparing the linear solution, element computation, and nonlinear solution algorithms.

In the section presenting the timing results concerning the linear solution algorithms, the results are divided into subsections discussing

- The memory requirements of the various linear solvers
- The comparison of the direct solver with linear preconditioned conjugate gradients
- A study of the LPCG convergence characteristics. The study is separated into two aspects: the variation of the condition number and the convergence rate of linear preconditioned conjugate gradients with the size of the time step, and the effect of the element computation algorithm employed and nonlinearity on the the convergence rate of linear preconditioned conjugate gradients.

The section pertaining to the element computation algorithms is partitioned into subsections concerning

- The blocking and memory requirements of the various element computation algorithms
- Speedups of the element computations performed by the various algorithms, including those versus the sequential application and the compiler optimized sequential application.

The section comparing of the nonlinear solution algorithms is organized by the example problems analyzed for this comparison, namely the 3D bend bar and 3D finite extension problems.

A summary is provided at the end of each section.

1.3.4 CONCLUSION

The conclusion chapter is partitioned into five sections, four of which are dedicated to the primary research areas of this thesis: the element computation algorithms, the linear solution algorithms, the nonlinear solution algorithms, and the finite strain plasticity material models. The fifth section concerns possible improvements and extensions to the research.

The section on the element computation algorithms draws conclusions in two principal areas:

- The algorithms implemented on the Alliant FX/8
- The algorithms implemented on the Convex C240

The section on the linear solution algorithms draws conclusions on

- meshes comprised of linear three dimensional isoparametric elements
- meshes comprised of quadratic three dimensional isoparametric elements

The section on nonlinear solution algorithms contains recommendations concerning the preferred algorithm while the section on the material models contains recommendations regarding which finite strain plasticity material model to employ in various situations.

The section on improvements and extensions is organized by the four principal areas of research.

2 THEORY AND APPLICATION

In this chapter, the theory supporting NLDfEP is presented and discussed. Based on this theoretical foundation, application algorithms are developed and examined. The four principal areas of concentration targeted in this chapter include the finite element formulation, solution algorithms for nonlinear systems of equations, element computation algorithms, and the numerical model pertaining to material behaviour.

2.1 FINITE ELEMENT FORMULATION

In NLDfEP, the structure is assumed to occupy the reference configuration B_0 at time $t = 0$ and to evolve through time to the deformed configuration B at time t . In the reference configuration, the structure may not be undeformed, and may not be at rest. In reaching the deformed configuration, the structure may displace in any manner, including simple rigid body translation or rotation in the absence of true deformation. This situation is illustrated in Fig. 2.1-1. The position vector \mathbf{X} identifies a point in the reference (undeformed) configuration and \mathbf{x} denotes the position vector of the same point in the deformed (current) configuration. The vector \mathbf{d} is the displacement vector that takes the point from the reference to the

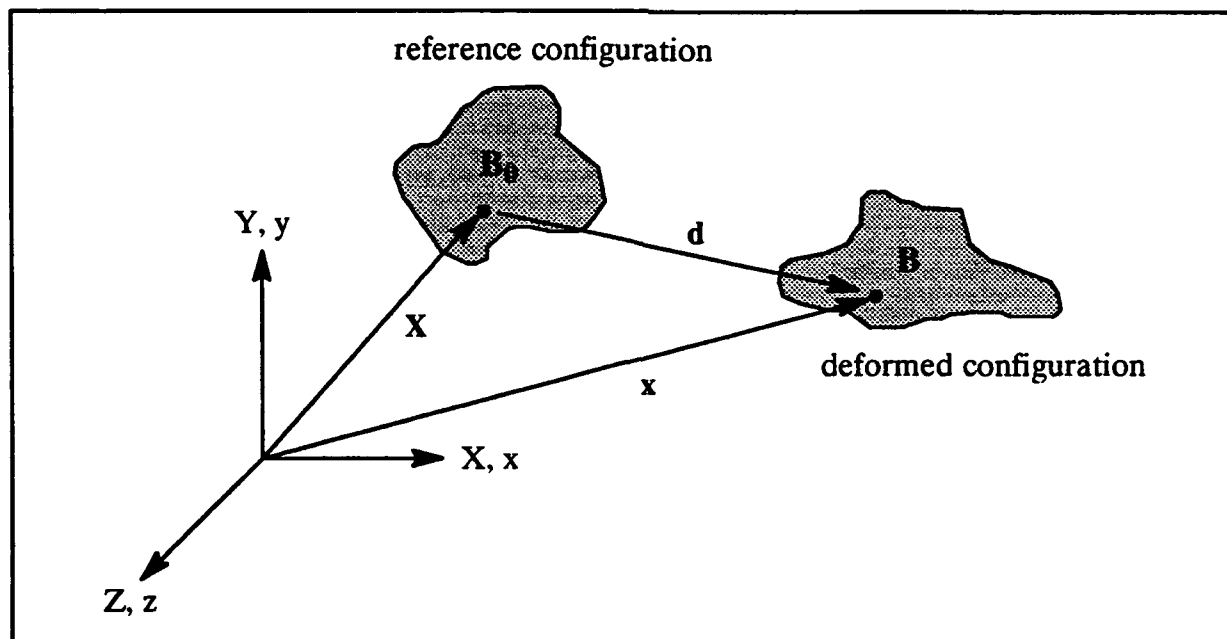


Fig. (2.1-1), definition of reference and deformed configurations

deformed configuration. The coordinates of the structure in the reference configuration represent the initial geometry interpolated from the parametric coordinates in the isoparametric formulation [16]. Following the development of Zienkiewicz and Nayak [86], the nonlinear

implementation of the finite element method in NLDFEP is a Total Lagrangian formulation so that the expression of virtual work defining equilibrium and the equation of motion are defined and solved in the reference configuration [16]. Throughout the deformation history of the structure, the choice of reference configuration remains constant; it is never updated.

In the remainder of this section, the equations of motion are derived, the basic development of the finite element formulation for three dimensional isoparametric elements is presented, and key element quantities such as the internal force vector are determined.

2.1.1 EQUATIONS OF MOTION

The virtual work expression in the reference configuration [86] is given by

$$\int_V \delta \mathbf{e}^T \mathbf{S} dV - \int_V \delta \mathbf{d}^T \mathbf{f} |\mathbf{F}| dV - \sum_{i=1}^m \delta \mathbf{d}_i^T \mathbf{p}_i = 0 \quad (2.1-1)$$

where \mathbf{e} and \mathbf{S} are the Green strain and second Piola-Kirchoff stress vectors, \mathbf{f} is the body force vector per unit volume in the deformed configuration, and each \mathbf{p}_i is a 3x1 vector of concentrated external forces acting at discrete points. The operator δ denotes a small, arbitrary virtual variation. The Green strain and the second Piola-Kirchoff stress are chosen to represent the internal virtual work in the reference configuration because they are work conjugate measures of stress and strain [36]. The matrix \mathbf{F} is the deformation gradient defined by

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} ; \quad |\mathbf{F}| > 0 \quad (2.1-2)$$

The external force vectors are assumed to be conservative in that the magnitude and direction of the loads are constant over a load step. In NLDFEP the only loads acting on the structure are assumed to be concentrated loads at the structural nodes. The only body forces in NLDFEP are the inertial D'Alembert forces defined by

$$\mathbf{f} = -\rho \ddot{\mathbf{d}} \quad (2.1-3)$$

where ρ is the mass density in the deformed configuration. Equation (1) then becomes

$$\int_V \delta \mathbf{e}^T \mathbf{S} dV + \int_V \delta \mathbf{d}^T \rho \ddot{\mathbf{d}} |\mathbf{F}| dV - \sum_{i=1}^m \delta \mathbf{d}_i^T \mathbf{p}_i = 0 \quad (2.1-4)$$

In accordance with the work of Zienkiewicz and Nayak [86], in the finite element formulation equation (4) evolves as

$$\begin{aligned}
& \int_V \delta \mathbf{e}^T \mathbf{S} dV + \int_V \delta \mathbf{d}^T \rho \ddot{\mathbf{d}} |\mathbf{F}| dV - \sum_{i=1}^m \delta \mathbf{d}_i^T \mathbf{p}_i = \\
& \sum_{j=1}^{\#elem} \int_{V_e} \delta \mathbf{e}^T \mathbf{S} dV_e + \sum_{j=1}^{\#elem} \int_{V_e} \delta \mathbf{d}^T \rho \ddot{\mathbf{d}} |\mathbf{F}| dV_e - \sum_{i=1}^m \delta \mathbf{d}_i^T \mathbf{p}_i = \\
& \sum_{j=1}^{\#elem} (\delta \mathbf{u}_e^T \mathbf{I}_e)_j + \sum_{j=1}^{\#elem} (\delta \mathbf{u}_e^T \mathbf{M}_e \ddot{\mathbf{u}}_e)_j - \sum_{i=1}^m \delta \mathbf{d}_i^T \mathbf{p}_i = \\
& \delta \mathbf{u}^T (\sum \mathbf{I}_e + (\sum \mathbf{M}_e) \ddot{\mathbf{u}} - \mathbf{P}) = 0
\end{aligned} \tag{2.1-5}$$

where \mathbf{u} is the global nodal displacement vector, \mathbf{u}_e is an element nodal displacement vector, \mathbf{I}_e is an element internal force vector, \mathbf{M}_e is an element mass matrix, and \mathbf{P} is the global external force vector. The determination of the element internal force vector and element mass matrix as well as the element tangent stiffness matrix is presented later. The summation in the last of equations (5) denotes the global assembly process. Since the $\delta \mathbf{u}$ are arbitrary in nature,

$$\sum \mathbf{I}_e + (\sum \mathbf{M}_e) \ddot{\mathbf{u}} - \mathbf{P} = 0 \tag{2.1-6}$$

Performing the global assembly of equation (6) gives the global equation of motion

$$\mathbf{I} + \mathbf{M} \ddot{\mathbf{u}} = \mathbf{P} \tag{2.1-7}$$

The solution of the nonlinear system of equations represented by equation (7) is discussed in Section 2.2.

2.1.2 BASIC DEVELOPMENT

The basic development of the finite element formulation using three dimensional isoparametric elements begins with the interpolation of the element displacements and coordinates. The displacement of a point from the reference configuration to the current configuration is interpolated from the nodal displacements through the use of the element shape function matrix, yielding

$$\mathbf{d} = \begin{Bmatrix} u \\ v \\ w \end{Bmatrix}_{3 \times 1} = \hat{\mathbf{N}} \begin{Bmatrix} (\mathbf{u}_e^X)_{n \times 1} \\ (\mathbf{u}_e^Y)_{n \times 1} \\ (\mathbf{u}_e^Z)_{n \times 1} \end{Bmatrix}_{3n \times 1} = \hat{\mathbf{N}} \mathbf{u}_e \tag{2.1-8}$$

The coordinates of a point in the reference configuration are interpolated from the nodal reference coordinates using the same shape functions, resulting in the similar equation

$$\mathbf{X} = \begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix}_{3 \times 1} = \hat{\mathbf{N}} \begin{Bmatrix} (\mathbf{c}_e^X)_{n \times 1} \\ (\mathbf{c}_e^Y)_{n \times 1} \\ (\mathbf{c}_e^Z)_{n \times 1} \end{Bmatrix}_{3n \times 1} = \hat{\mathbf{N}} \mathbf{c}_e \quad (2.1-9)$$

The element shape functions, one for each element node, are functions of the parametric variables ξ , η , and ζ . They are grouped in the row vector

$$\mathbf{N} = \langle N_1 \ N_2 \ \dots \ N_n \rangle_{1 \times n} \quad (2.1-10)$$

where n is the number of element nodes. The shape function derivatives with respect to the parametric variables are represented by the row vectors

$$\begin{aligned} \mathbf{N}_{,\xi} &= \langle N_{1,\xi} \ N_{2,\xi} \ \dots \ N_{n,\xi} \rangle_{1 \times n} \\ \mathbf{N}_{,\eta} &= \langle N_{1,\eta} \ N_{2,\eta} \ \dots \ N_{n,\eta} \rangle_{1 \times n} \\ \mathbf{N}_{,\zeta} &= \langle N_{1,\zeta} \ N_{2,\zeta} \ \dots \ N_{n,\zeta} \rangle_{1 \times n} \end{aligned} \quad (2.1-11)$$

The element shape functions are collected in the element shape function matrix defined by

$$\hat{\mathbf{N}} = \begin{bmatrix} \mathbf{N} & 0 & 0 \\ 0 & \mathbf{N} & 0 \\ 0 & 0 & \mathbf{N} \end{bmatrix}_{3 \times 3n} \quad (2.1-12)$$

The Jacobian matrix relating infinitesimal increments in the reference configuration to those in parametric space is given by

$$\mathbf{J} = \begin{bmatrix} \frac{\partial X}{\partial \xi} & \frac{\partial Y}{\partial \xi} & \frac{\partial Z}{\partial \xi} \\ \frac{\partial X}{\partial \eta} & \frac{\partial Y}{\partial \eta} & \frac{\partial Z}{\partial \eta} \\ \frac{\partial X}{\partial \zeta} & \frac{\partial Y}{\partial \zeta} & \frac{\partial Z}{\partial \zeta} \end{bmatrix}_{3 \times 3} \quad (2.1-13)$$

with the inverse of the Jacobian matrix denoted by

$$\Gamma = J^{-1} \quad (2.1-14)$$

The displacement gradients in the reference configuration are contained in the vector defined by

$$\Theta = \begin{bmatrix} d_{,X} \\ d_{,Y} \\ d_{,Z} \end{bmatrix} \equiv \begin{Bmatrix} \Theta_X \\ \Theta_Y \\ \Theta_Z \end{Bmatrix} = \begin{Bmatrix} u_{,X} \\ v_{,X} \\ w_{,X} \\ u_{,Y} \\ v_{,Y} \\ w_{,Y} \\ u_{,Z} \\ v_{,Z} \\ w_{,Z} \end{Bmatrix}_{9 \times 1} \quad (2.1-15)$$

The displacement gradients in parametric space constitute the vector

$$\Phi = \begin{bmatrix} d_{,\xi} \\ d_{,\eta} \\ d_{,\zeta} \end{bmatrix} = \begin{Bmatrix} u_{,\xi} \\ v_{,\xi} \\ w_{,\xi} \\ u_{,\eta} \\ v_{,\eta} \\ w_{,\eta} \\ u_{,\zeta} \\ v_{,\zeta} \\ w_{,\zeta} \end{Bmatrix}_{9 \times 1} \quad (2.1-16)$$

The two displacement gradient vectors are related by the equation

$$\Theta = \hat{\Gamma} \Phi \quad (2.1-17)$$

where

$$\hat{\Gamma} = \begin{bmatrix} \Gamma_{11}I_3 & \Gamma_{12}I_3 & \Gamma_{13}I_3 \\ \Gamma_{21}I_3 & \Gamma_{22}I_3 & \Gamma_{23}I_3 \\ \Gamma_{31}I_3 & \Gamma_{32}I_3 & \Gamma_{33}I_3 \end{bmatrix}_{9 \times 9} \quad (2.1-18)$$

The displacement gradient vector in parametric space is expressed in terms of the nodal displacements by

$$\Phi = Gu_e \quad (2.1-19)$$

where

$$\mathbf{G} = \begin{bmatrix} \hat{N}_{,\xi} \\ \hat{N}_{,\eta} \\ \hat{N}_{,\zeta} \end{bmatrix} = \begin{bmatrix} N_{,\xi} & 0 & 0 \\ 0 & N_{,\xi} & 0 \\ 0 & 0 & N_{,\xi} \\ N_{,\eta} & 0 & 0 \\ 0 & N_{,\eta} & 0 \\ 0 & 0 & N_{,\eta} \\ N_{,\zeta} & 0 & 0 \\ 0 & N_{,\zeta} & 0 \\ 0 & 0 & N_{,\zeta} \end{bmatrix}_{9 \times 3n} \quad (2.1-20)$$

The Green strain vector is defined by

$$\mathbf{e} = \begin{bmatrix} \epsilon_X \\ \epsilon_Y \\ \epsilon_Z \\ \gamma_{XY} \\ \gamma_{YZ} \\ \gamma_{XZ} \end{bmatrix} = \begin{bmatrix} u_{,X} \\ v_{,Y} \\ w_{,Z} \\ u_{,Y} + v_{,X} \\ v_{,Z} + w_{,Y} \\ w_{,X} + u_{,Z} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(u_{,X}^2 + v_{,X}^2 + w_{,X}^2) \\ \frac{1}{2}(u_{,Y}^2 + v_{,Y}^2 + w_{,Y}^2) \\ \frac{1}{2}(u_{,Z}^2 + v_{,Z}^2 + w_{,Z}^2) \\ u_{,X}u_{,Y} + v_{,X}v_{,Y} + w_{,X}w_{,Y} \\ u_{,Y}u_{,Z} + v_{,Y}v_{,Z} + w_{,Y}w_{,Z} \\ u_{,X}u_{,Z} + v_{,X}v_{,Z} + w_{,X}w_{,Z} \end{bmatrix} \quad (2.1-21)$$

The Green strain vector and Green strain rate vector are written in terms of the nodal displacements and velocities by

$$\mathbf{e}_{6 \times 1} = \mathbf{B}_s \mathbf{u}_e \quad (2.1-22)$$

and

$$\dot{\mathbf{e}}_{6 \times 1} = \mathbf{B} \dot{\mathbf{u}}_e \quad (2.1-23)$$

Defining the matrix A in terms of the reference configuration displacement gradients as

$$\mathbf{A} = \begin{bmatrix} \Theta_X^T & 0 & 0 \\ 0 & \Theta_Y^T & 0 \\ 0 & 0 & \Theta_Z^T \\ \Theta_Y^T & \Theta_X^T & 0 \\ 0 & \Theta_Z^T & \Theta_Y^T \\ \Theta_Z^T & 0 & \Theta_X^T \end{bmatrix}_{6 \times 9} \quad (2.1-24)$$

and the matrix

$$\tilde{\mathbf{B}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}_{6 \times 9} \quad (2.1-25)$$

permits the strain-displacement matrices \mathbf{B}_s and \mathbf{B} to be determined by the equations

$$(\mathbf{B}_s)_{6 \times 3n} = (\bar{\mathbf{B}} + \frac{1}{2} \mathbf{A}) \hat{\Gamma} \mathbf{G} \quad (2.1-26)$$

and

$$\mathbf{B}_{6 \times 3n} = (\bar{\mathbf{B}} + \mathbf{A}) \hat{\Gamma} \mathbf{G} \quad (2.1-27)$$

The vectors and matrices presented in this section form the building blocks of the key element quantities determined below.

2.1.3 DETERMINATION OF ELEMENT QUANTITIES

The key element quantities and calculations vital to the solution of the equations of motion are the internal force vector, the tangent stiffness matrix, the mass matrix, and the recovery of the second Piola-Kirchoff stress vector. The stress recovery is explored in depth in Section 2.4. The remaining element quantities are examined in the following development.

2.1.3.1 INTERNAL FORCE VECTOR

The element internal force vector is derived from the internal virtual work term in equation (1) given by

$$\int_{V_e} \delta \mathbf{e}^T \mathbf{S} dV_e \quad (2.1-28)$$

Assuming small virtual displacements, the virtual Green strains are related to the virtual element nodal displacements as in equation (23) by

$$\delta \mathbf{e} = \mathbf{B} \delta \mathbf{u}_e \quad (2.1-29)$$

Substituting equation (29) into equation (28) yields

$$\int_{V_e} \delta \mathbf{e}^T \mathbf{S} dV_e = \delta \mathbf{u}_e^T \int_{V_e} \mathbf{B}^T \mathbf{S} dV_e \quad (2.1-30)$$

Comparison with the third of equations (5) shows the element internal force vector to be defined as

$$\mathbf{I}_e = \int_{V_e} \mathbf{B}^T \mathbf{S} dV_e = \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 \mathbf{B}^T \mathbf{S} |J| d\xi d\eta d\zeta \quad (2.1-31)$$

The global internal force vector is obtained through global assembly of the element internal force vectors. It is calculated using the element computation algorithms of Section 2.3.

2.1.3.2 TANGENT STIFFNESS MATRIX

The element tangent stiffness matrix is defined in terms of the rate of the element internal force vector by

$$\dot{\mathbf{I}}_e = (\mathbf{K}_t)_e \dot{\mathbf{u}}_e \quad (2.1-32)$$

From equation (31) the rate of the element internal force vector is

$$\dot{\mathbf{I}}_e = \int_{V_e} \dot{\mathbf{B}}^T \mathbf{S} \, dV_e + \int_{V_e} \mathbf{B}^T \dot{\mathbf{S}} \, dV_e \quad (2.1-33)$$

The first term in equation (33) can be manipulated to the form

$$\int_{V_e} \dot{\mathbf{B}}^T \mathbf{S} \, dV_e = \left(\int_{V_e} \mathbf{G}^T \hat{\Gamma}^T \mathbf{M}_\sigma \hat{\Gamma} \mathbf{G} \, dV_e \right) \dot{\mathbf{u}}_e \quad (2.1-34)$$

where

$$\mathbf{M}_\sigma = \begin{bmatrix} S_1 \mathbf{I}_3 & S_4 \mathbf{I}_3 & S_6 \mathbf{I}_3 \\ S_4 \mathbf{I}_3 & S_2 \mathbf{I}_3 & S_5 \mathbf{I}_3 \\ S_6 \mathbf{I}_3 & S_5 \mathbf{I}_3 & S_3 \mathbf{I}_3 \end{bmatrix}_{9 \times 9} \quad (2.1-35)$$

The term in parentheses in equation (34) is the geometric tangent stiffness matrix

$$(\mathbf{K}_t^g)_e = \int_{V_e} \mathbf{G}^T \hat{\Gamma}^T \mathbf{M}_\sigma \hat{\Gamma} \mathbf{G} \, dV_e \quad (2.1-36)$$

The second term in equation (33) resolves to

$$\int_{V_e} \mathbf{B}^T \dot{\mathbf{S}} \, dV_e = \left(\int_{V_e} \mathbf{B}^T \mathbf{C} \mathbf{B} \, dV_e \right) \dot{\mathbf{u}}_e \quad (2.1-37)$$

where \mathbf{C} is the constitutive matrix relating the rate of second Piola–Kirchoff stress to the Green strain rate, as in

$$\dot{\mathbf{S}} = \mathbf{C}\dot{\mathbf{e}} \quad (2.1-38)$$

Combining equations (34) and (37) permits the element tangent stiffness matrix to be written as

$$\begin{aligned} (\mathbf{K}_T)_e &= \int_{V_e} (\mathbf{G}^T \hat{\Gamma}^T \mathbf{M}_o \hat{\Gamma} \mathbf{G} + \mathbf{B}^T \mathbf{C} \mathbf{B}) dV_e \\ &= \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 (\mathbf{G}^T \hat{\Gamma}^T \mathbf{M}_o \hat{\Gamma} \mathbf{G} + \mathbf{B}^T \mathbf{C} \mathbf{B}) |\mathbf{J}| d\xi d\eta d\zeta \end{aligned} \quad (2.1-39)$$

Again, the global tangent stiffness matrix can be obtained through global assembly of the element matrices. The tangent stiffness matrix is also evaluated using the element computation algorithms of Section 2.3.

2.1.3.3 MASS MATRIX

The element consistent mass matrix is derived from the inertial virtual work term in the second of equations (5) given by

$$\int_{V_e} \delta \mathbf{d}^T \rho \ddot{\mathbf{d}} |\mathbf{F}| dV_e \quad (2.1-40)$$

Substituting equation (8) and its second time derivative, noting that the shape functions are independent of time, into equation (40) yields

$$\int_{V_e} \delta \mathbf{d}^T \rho \ddot{\mathbf{d}} |\mathbf{F}| dV_e = \delta \mathbf{u}_e^T \left(\int_{V_e} \hat{\mathbf{N}}^T \hat{\mathbf{N}} \rho |\mathbf{F}| dV_e \right) \ddot{\mathbf{u}}_e \quad (2.1-41)$$

Comparison with the third of equations (5) reveals the element consistent mass matrix to be

$$\mathbf{M}_e = \int_{V_e} \hat{\mathbf{N}}^T \hat{\mathbf{N}} \rho |\mathbf{F}| dV_e = \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 \hat{\mathbf{N}}^T \hat{\mathbf{N}} \rho |\mathbf{F}| |\mathbf{J}| d\xi d\eta d\zeta \quad (2.1-42)$$

In practice, considering the block diagonal structure of equation (12), the element consistent mass matrix is also block diagonal, and it is only necessary to compute the block diagonal mass matrix corresponding to one of the three continuum degrees of freedom and to assign this matrix to the other two.

The mass density ρ used in equation (42) corresponds to the current configuration, as the inertial body force acts there. It is defined in terms of the original mass density in the reference configuration by

$$\rho_0 = \rho |F| \quad (2.1-43)$$

Substitution of equation (43) into (42) yields

$$M_e = \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 \dot{N}^T \dot{N} \rho_0 |J| d\xi d\eta d\zeta \quad (2.1-44)$$

The element consistent mass matrix defined by equation (44) is independent of time, so that the element tangent and secant consistent mass matrices are seen to be equal.

It is also possible to define a diagonal element lumped mass matrix. This is accomplished in the following manner:

- 1) Compute the diagonal terms of the block diagonal consistent mass matrix corresponding to one of the continuum degrees of freedom.
- 2) Accumulate the mass of these diagonal terms. Scale the diagonal terms by the ratio of the total element mass related to the continuum degree of freedom to the accumulated mass so that the total mass of the diagonal terms is correct. Assign the diagonal terms to the other two continuum degrees of freedom. This is the element lumped mass matrix.

Once again, either the global consistent or lumped mass matrix is found through assembly of the element matrices. The consistent or lumped mass matrix is once more computed using the element computation algorithms of Section 2.3.

2.2 LINEAR AND NONLINEAR SOLUTION ALGORITHMS

A main thrust of this research effort is to explore the suitability of various linear and nonlinear solution algorithms to supercomputer applications. In this section the nonlinear solution algorithms implemented in NLDFEP, as well as the linear solvers employed by these algorithms, are presented and developed. In addition, the particulars of the numerical integration technique pertaining to the dynamic analysis is discussed to provide necessary background information for the nonlinear solution algorithms.

2.2.1 DYNAMIC ANALYSIS: NEWMARK BETA METHOD

The numerical integration of the equations of motion in NLDFEP is based on a method attributed to Newmark [59]. The method relies upon a two parameter family of equations that define the displacement, velocity, and acceleration at time t_{n+1} in terms of the displacement increment from t_n to t_{n+1} and the kinematic state at time t_n . These equations result from successive application of the extended mean value theorem of differential calculus [82]. Consider first the velocities at time t_n and t_{n+1} . Use of the extended mean value theorem for the first derivative leads to the equation

$$\dot{u}_{n+1} = \dot{u}_n + \Delta t \ddot{u}_\gamma ; \quad \ddot{u}_\gamma \in [\ddot{u}_n , \ddot{u}_{n+1}] \quad (2.2-1)$$

Using the relationship

$$\ddot{u}_\gamma = (1 - \gamma)\ddot{u}_n + \gamma\ddot{u}_{n+1} ; \quad 0 \leq \gamma \leq 1 \quad (2.2-2)$$

equation (1) can be rewritten as

$$\dot{u}_{n+1} = \dot{u}_n + (1 - \gamma)\Delta t\ddot{u}_n + \gamma\Delta t\ddot{u}_{n+1} \quad (2.2-3)$$

Equation (3) would be exact for a given time interval if the parameter γ happened to be chosen correctly. Even so, the constant acceleration \ddot{u}_γ upon integration of equation (1) would not necessarily produce the correct displacement at time t_{n+1} in terms of the displacement and velocity at time t_n . Accordingly, the extended mean value theorem for the second derivative is invoked to yield

$$u_{n+1} = u_n + \Delta t\dot{u}_n + \frac{\Delta t^2}{2}\ddot{u}_\beta ; \quad \ddot{u}_\beta \in [\ddot{u}_n , \ddot{u}_{n+1}] \quad (2.2-4)$$

Again, a relationship such as

$$\ddot{u}_\beta = (1 - 2\beta)\ddot{u}_n + 2\beta\ddot{u}_{n+1} ; \quad 0 \leq 2\beta \leq 1 \quad (2.2-5)$$

is employed to recast equation (4) as

$$u_{n+1} = u_n + \Delta t\dot{u}_n + \frac{(1 - 2\beta)}{2}\Delta t^2\ddot{u}_n + \beta\Delta t^2\ddot{u}_{n+1} \quad (2.2-6)$$

Equation (6) would also be exact for a given time interval as long as the choice of the parameter β proved to be correct. Of course, in general it is impossible to choose either γ or β correctly without knowing the solution in advance, so that the approximation in the Newmark meth-

od lies in the choice of γ and β . Newmark [59] showed that to avoid spurious damping in linear systems, the parameter γ should equal $1/2$. The pertinent equations of the Newmark method then become

$$\begin{aligned}\ddot{\mathbf{u}}_{n+1} &= \ddot{\mathbf{u}}_n + \frac{\Delta t}{2} (\ddot{\mathbf{u}}_n + \ddot{\mathbf{u}}_{n+1}) \\ \mathbf{u}_{n+1} &= \mathbf{u}_n + \Delta t \dot{\mathbf{u}}_n + \frac{(1-2\beta)}{2} \Delta t^2 \ddot{\mathbf{u}}_n + \beta \Delta t^2 \ddot{\mathbf{u}}_{n+1}\end{aligned}\tag{2.2-7}$$

A wide variety of values for the parameter β are possible. For instance, setting β equal to zero leads to the well known second central difference method. A choice of β equal to $1/6$ results in the linear acceleration method, where the acceleration is assumed to vary linearly over the time increment. Selecting β equal to $1/4$ produces the constant average acceleration method. Newmark [59] demonstrated that β equal to $1/4$ renders the method unconditionally stable for linear problems; other choices must satisfy a time step constraint to maintain stability throughout the solution. For materially nonlinear problems, Schoeberle and Belytschko [75] established that the use of β equal to $1/4$ leads to unconditional stability as long as enough nonlinear equilibrium iterations are performed to satisfy an energy convergence criterion, and for nonlinear elastic problems Hughes [38] found much the same situation. Consequently, dynamic applications in NLD FEP generally specify β to be $1/4$, and rely on a sufficiently stringent convergence criterion for the nonlinear equilibrium iterations to maintain stability.

Use of the Newmark method leads to an implicit dynamic formulation in that the solution of a nontrivial system of equations is required to compute a displacement increment. Toward this end (and assuming that β does not equal zero), equations (7) are manipulated to the form

$$\begin{aligned}\Delta \mathbf{u}_{n+1} &= \mathbf{u}_{n+1} - \mathbf{u}_n \\ \dot{\mathbf{u}}_{n+1} &= \frac{1}{2\beta\Delta t} \Delta \mathbf{u}_{n+1} - \frac{(1-2\beta)}{2\beta} \dot{\mathbf{u}}_n - \frac{(1-4\beta)}{4\beta} \Delta t \ddot{\mathbf{u}}_n \\ \ddot{\mathbf{u}}_{n+1} &= \frac{1}{\beta\Delta t^2} \Delta \mathbf{u}_{n+1} - \frac{1}{\beta\Delta t} \dot{\mathbf{u}}_n - \frac{(1-2\beta)}{2\beta} \ddot{\mathbf{u}}_n\end{aligned}\tag{2.2-8}$$

Equations (8) are substituted into the equations of motion and the chosen iterative nonlinear solution algorithm, the total displacement increment for the current iteration is computed,

and that increment is back substituted into equations (8) to produce the velocity and acceleration for the current estimate of the solution at time t_{n+1} .

2.2.2 MODIFIED NEWTON-RAPHSON

Recalling the basic equation of motion, the residual load vector at any time is expressed as

$$\mathbf{R} = \mathbf{P} - \mathbf{I} - \mathbf{M}\ddot{\mathbf{u}} \quad (2.2-9)$$

Where \mathbf{P} is the external load vector, \mathbf{I} is the internal force vector, \mathbf{M} is the mass matrix, and \mathbf{u} is the nodal displacement vector. The residual is the out-of-balance force vector that results from the iterative solution of a nonlinear problem, an iterative solution designed to drive the residual as close to zero as desired. The Newton-Raphson algorithm for nonlinear equations, illustrated in Fig. 2.1-1 for a static problem, can be derived by assuming that there exists an approximate displacement state in the neighborhood of which a linear mapping represented by

$$\mathbf{R}(\mathbf{u}) = \mathbf{R}(\bar{\mathbf{u}}) + d\mathbf{R}(\mathbf{u}) = \mathbf{R}(\bar{\mathbf{u}}) + \frac{\partial \mathbf{R}}{\partial \mathbf{u}} d\mathbf{u} \quad (2.2-10)$$

is a good approximation to the residual load vector. The partial derivative in equation (10) represents the Jacobian matrix mapping the displacement vector to the residual load vector. Presumably, a better approximation is obtained by setting equation (10) to zero. The differential increment of the residual load vector, remembering that the mass matrix for a given time step is constant, is given by

$$d\mathbf{R} = d\mathbf{P} - d\mathbf{I} - \mathbf{M}d\ddot{\mathbf{u}} \quad (2.2-11)$$

The external loads in NLD FEP are nodal loads, assumed to be constant in direction and magnitude during the solution of a time step. Accordingly, utilizing equations (8) and recalling the derivation of the tangent stiffness, the terms of equation (11) are defined as

$$d\mathbf{P} = 0$$

$$d\mathbf{I} = \mathbf{K}_t d\mathbf{u} \quad (2.2-12)$$

$$\mathbf{M}d\ddot{\mathbf{u}} = \frac{1}{\beta \Delta t^2} \mathbf{M}d\mathbf{u}$$

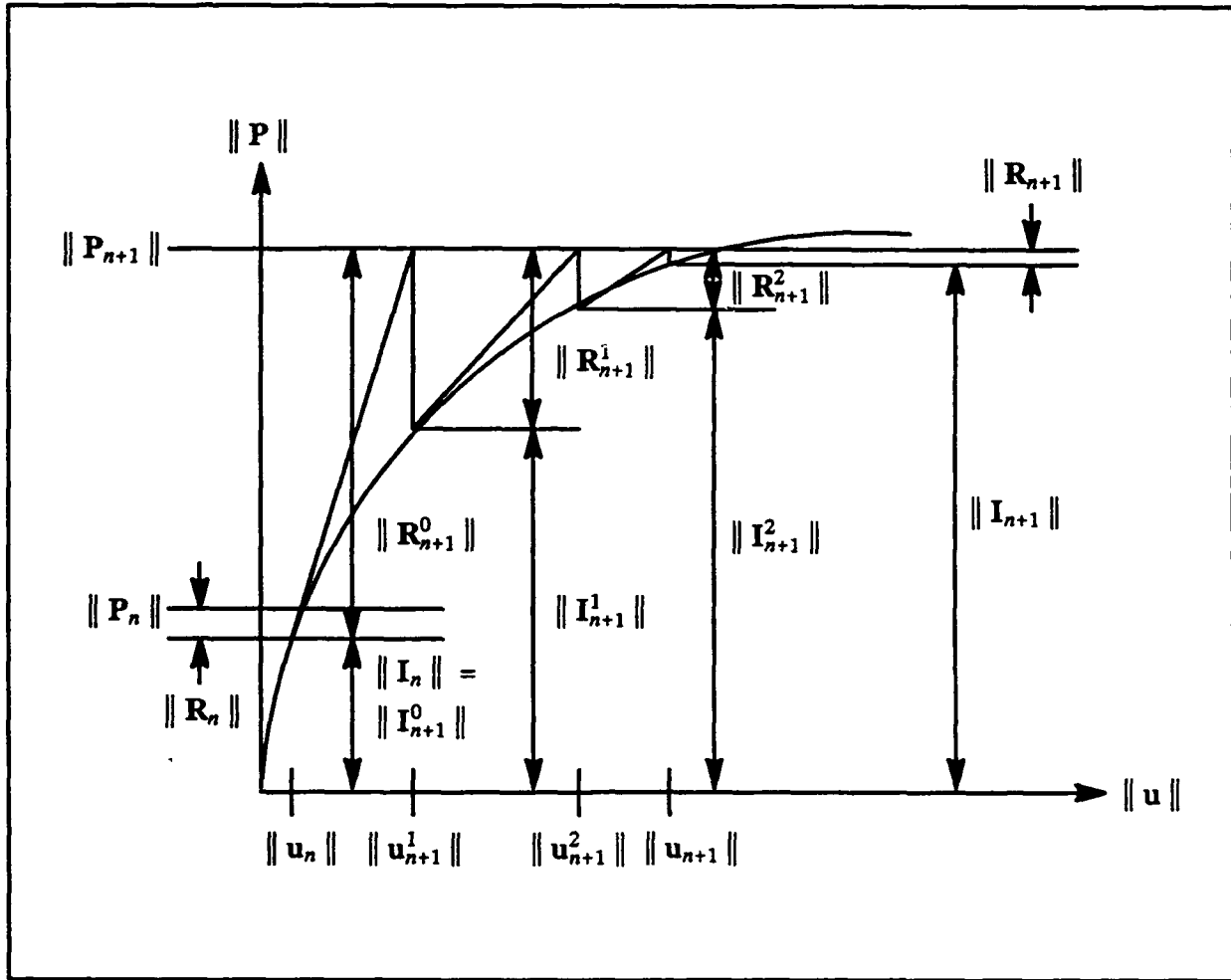


Fig. (2.2-1), illustration of the Newton-Raphson algorithm for static problems

Equation (11) can then be written in the form

$$dR = -K_t^d du \quad (2.2-13)$$

where

$$K_t^d = K_t + \frac{1}{\beta \Delta t^2} M \quad (2.2-14)$$

defines the dynamic tangent stiffness. Substituting equation (13) into equation (10) yields

$$R(u) = R(\bar{u}) - K_t^d du \quad (2.2-15)$$

so that the dynamic tangent stiffness is seen to be the negative of the Jacobian matrix relating the residual load vector to the displacement vector:

$$\mathbf{K}_t^d = -\frac{\partial \mathbf{R}}{\partial \mathbf{u}} \quad (2.2-16)$$

Setting equation (15) to zero and rearranging gives

$$\mathbf{K}_t^d d\mathbf{u} = \mathbf{R}(\dot{\mathbf{u}}) \quad (2.2-17)$$

For finite increments, equation (17) becomes

$$\mathbf{K}_t^d \delta \mathbf{u}_{n+1}^i = \mathbf{R}_{n+1}^{i-1} \quad (2.2-18)$$

where $\delta \mathbf{u}_{n+1}^i$ is the increment of displacement for the current iteration of the time step advancing the solution from time t_n to time t_{n+1} and \mathbf{R}_{n+1}^{i-1} is the residual load after the previous iteration. This residual is defined as

$$\mathbf{R}_{n+1}^{i-1} = \mathbf{P}_{n+1} - \mathbf{I}_{n+1}^{i-1} - \mathbf{M}\ddot{\mathbf{u}}_{n+1}^{i-1} \quad (2.2-19)$$

or, after substitution of equations (8), alternately as

$$\mathbf{R}_{n+1}^{i-1} = \mathbf{P}_{n+1}^d - \mathbf{I}_{n+1}^{i-1} - \frac{1}{\beta \Delta t^2} \mathbf{M} \Delta \mathbf{u}_{n+1}^{i-1} \quad (2.2-20)$$

where \mathbf{P}_{n+1}^d is the applied load vector at time t_{n+1} modified by terms associated with equations (8):

$$\mathbf{P}_{n+1}^d = \mathbf{P}_{n+1} + \frac{1}{\beta \Delta t} \mathbf{M} \ddot{\mathbf{u}}_n + \frac{(1-2\beta)}{2\beta} \mathbf{M} \ddot{\mathbf{u}}_n \quad (2.2-21)$$

The displacement vector for the current iteration is obtained from the displacement vector after the previous time step and the total increment of the displacement vector for the current time step; the total increment is the sum of the displacement increments from all iterations:

$$\mathbf{u}_{n+1}^i = \mathbf{u}_n + \Delta \mathbf{u}_{n+1}^i; \quad \Delta \mathbf{u}_{n+1}^i = \sum_1^i \delta \mathbf{u}_{n+1}^k \quad (2.2-22)$$

Combining equations (18) and (20) generates the basic equation driving the iterative solution associated with the Newton-Raphson method:

$$\mathbf{K}_t^d \delta \mathbf{u}_{n+1}^i = \mathbf{P}_{n+1}^d - \mathbf{I}_{n+1}^{i-1} - \frac{1}{\beta \Delta t^2} \mathbf{M} \Delta \mathbf{u}_{n+1}^{i-1} \quad (2.2-23)$$

In the modified Newton-Raphson method, it is not necessary to update the dynamic tangent stiffness matrix every iteration; the matrix is updated when required for adequate convergence. Iterations continue until specified convergence criteria are met or until a specified limit on iterations is reached.

The residual load vector, the dynamic tangent stiffness, and the mass matrix are computed using the element computation algorithms discussed in Section 2.3. The solution of equation (23) for the iterative displacement increment is performed by solvers discussed subsequently.

2.2.2.1 CONVERGENCE CRITERIA

Four convergence criteria are provided in NLDFEP for the modified Newton-Raphson method. They are:

- 1) $\| \delta \mathbf{u}_{n+1}^i \| \leq \delta_1 \| \delta \mathbf{u}_{n+1}^1 \|$
- 2) $\| \mathbf{R}_{n+1}^i \| \leq \delta_2 \| \mathbf{R}_{n+1}^0 \|$
- 3) $\max(|(\delta u_{n+1}^i)_k|, k = 1, N_{eq}) \leq \delta_3 \| \delta \mathbf{u}_{n+1}^1 \|$
- 4) $\max(|(\mathbf{R}_{n+1}^i)_k|, k = 1, N_{eq}) \leq \delta_4 \| \mathbf{R}_{n+1}^0 \|$

Possible alternatives to tests 2) and 4) would include the initial reactions for constrained degrees of freedom in the initial residual norm, which normally contains zero for constrained degrees of freedom. This would aid convergence in situations where the initial residual is small compared to the magnitude of the overall loading experienced by the structure. Additionally, there is no mechanism to control loading in the vicinity of limit points or to otherwise improve performance in such situations. Such techniques were considered peripheral to this research and were not explored. The numerical examples presented in Sections 3.1 to 3.5 employ test 2) with a tolerance of 0.01% for dynamic applications, and no stability problems in the dynamic solution were observed.

2.2.2.2 SECANT-NEWTON

The secant-Newton method is essentially an accelerated modified Newton-Raphson technique [19]. It proceeds by modifying the incremental displacement vector calculated by equation (23). The modified incremental displacement vector for the current iteration is

$$\delta \ddot{\mathbf{u}}_{n+1}^i = \mathbf{e}_i \delta \ddot{\mathbf{u}}_{n+1}^{i-1} + f_i \delta \mathbf{u}_{n+1}^i \quad (2.2-24)$$

where

$$\delta \mathbf{u}_{n+1}^i = \mathbf{K}_t^{d-1} \mathbf{R}_{n+1}^{i-1} \quad (2.2-25)$$

as before and the scalars e_i and f_i are defined as

$$\begin{aligned} f_i &= -\frac{a_i}{b_i} \\ e_i &= f_i\left(1 - \frac{c_i}{b_i}\right) - 1 \end{aligned} \quad (2.2-26)$$

The scalar inner products a_i , b_i , and c_i are given by

$$\begin{aligned} a_i &= (\delta \mathbf{u}_{n+1}^{i-1})^T \mathbf{R}_{n+1}^{i-1} \\ b_i &= (\delta \mathbf{u}_{n+1}^{i-1})^T (\mathbf{R}_{n+1}^i - \mathbf{R}_{n+1}^{i-1}) \\ c_i &= (\delta \mathbf{u}_{n+1}^i)^T (\mathbf{R}_{n+1}^i - \mathbf{R}_{n+1}^{i-1}) \end{aligned} \quad (2.2-27)$$

The total increment of displacement for the current step is now the sum of the modified displacement increments for all iterations:

$$\Delta \mathbf{u}_{n+1}^i = \sum_1^i \delta \mathbf{u}_{n+1}^k \quad (2.2-28)$$

Equation (24) represents the secant relationship characteristic of the variable metric or quasi-Newton nonlinear solution method, given in a more recognizable form by

$$(\delta \mathbf{u}_{n+1}^i)^T (\mathbf{R}_{n+1}^i - \mathbf{R}_{n+1}^{i-1}) = -(\delta \mathbf{u}_{n+1}^{i-1})^T \mathbf{R}_{n+1}^i \quad (2.2-29)$$

Equation (29) can be derived by taking the dot product of equation (24) with $\mathbf{R}_{n+1}^i - \mathbf{R}_{n+1}^{i-1}$. This relationship is illustrated in one dimension in Fig. 2.2-2, and is akin to the search direction update in the preconditioned conjugate gradient method that, in part, enforces the conjugate condition, and allows for the faster convergence for which quasi-Newton is noted. In fact, a secant-Newton iteration is equivalent to a quasi-Newton iteration when

- 1) The estimate of the dynamic tangent stiffness used in equation (25) is identical to the approximate dynamic tangent stiffness used in the previous iteration of the quasi-Newton method.
- 2) The previous iteration of modified Newton-Raphson is not accelerated.

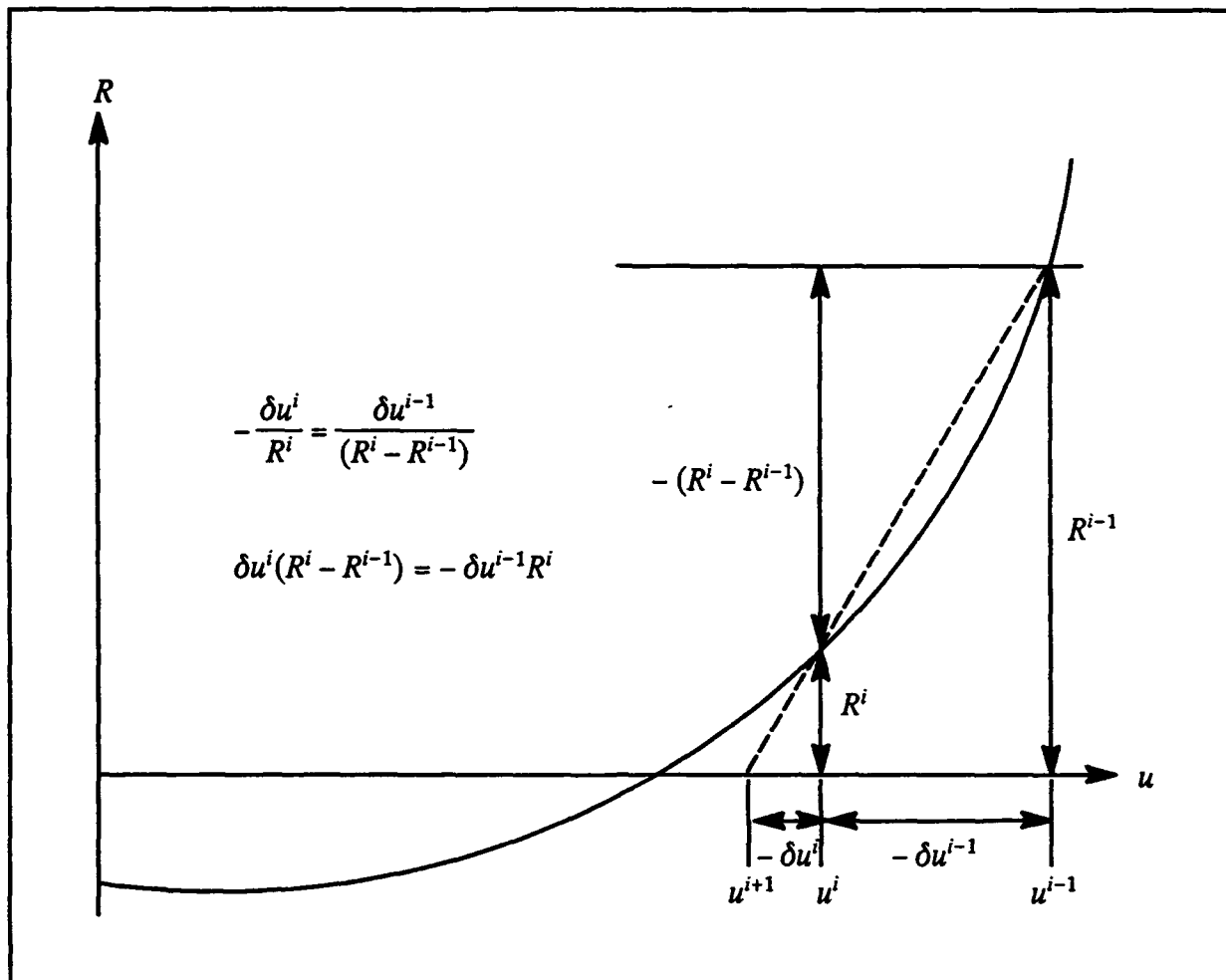


Fig. (2.2-2), illustration of the secant relationship in one dimension

In practice, all iterations of modified Newton-Raphson are accelerated, and the stiffnesses do not coincide. However, the influence of the quasi-Newton method is still manifest. In this way, the convergence characteristics of the quasi-Newton method can be studied in relation to modified Newton-Raphson and the preconditioned conjugate gradient method yet to be discussed, without actually implementing quasi-Newton.

2.2.2.3 SOLVERS

In NLDfEP, the resolution of the linear set of equations described by equation (23) is handled either by a direct solver or by a linear preconditioned conjugate gradient algorithm. The direct solver used is generally a highly optimized version of the LINPACK Cholesky factorization and back substitution routines that is provided by the computer manufacturer. The direct solver is used as a benchmark against which to compare the linear preconditioned conjugate gradient solver. Although the manufacturer optimized LINPACK

routines may not be the fastest available, they are representative of direct solvers and are a worthy benchmark. Linear preconditioned conjugate gradient as an alternative to a direct solver is one of the main focuses of this research and is discussed in detail presently.

2.2.3 LINEAR PRECONDITIONED CONJUGATE GRADIENT

As stated above, the linear preconditioned conjugate gradient algorithm can be used to solve the linear system of equations in a nonlinear iteration of modified Newton–Raphson. It can also perform the same function for the nonlinear preconditioned conjugate gradient algorithm as will be examined shortly. In the following development, the linear system of equations is denoted by $\mathbf{Ax} = \mathbf{b}$, where \mathbf{A} is understood to be the current estimate of the dynamic tangent stiffness and \mathbf{b} the nonlinear residual. The matrix \mathbf{B} represents the preconditioning matrix. The linear preconditioned conjugate gradient algorithm proceeds as follows:

1) Initialize:

$$\mathbf{x}_0 = \mathbf{0}$$

for $j = 1, N_{eq}$; if j is a constrained dof,

$$r_j = 0$$

else

$$r_j = b_j$$

end if

$$k = 1$$

note: to facilitate linear and nonlinear preconditioned conjugate gradient and modified Newton–Raphson using Crisfield acceleration, non-zero displacement constraints are placed in the total increment of displacement vector at the beginning of each step and corresponding residual entries are set to zero. Otherwise, the non-zero constraints will be propagated throughout the solution by equation (24) above and equations (32) and (60) below.

2) Compute:

$$\mathbf{z}_{k-1} = \mathbf{B}^{-1}\mathbf{r}_{k-1} \quad (2.2-30)$$

$$\beta_k = \frac{\mathbf{z}_{k-1}^T \mathbf{r}_{k-1}}{\mathbf{z}_{k-2}^T \mathbf{r}_{k-2}} \quad (\beta_1 = 0) \quad (2.2-31)$$

$$\mathbf{p}_k = \mathbf{z}_{k-1} + \beta_k \mathbf{p}_{k-1} \quad (\mathbf{p}_0 = \mathbf{0}) \quad (2.2-32)$$

$$\alpha_k = \frac{\mathbf{z}_{k-1}^T \mathbf{r}_{k-1}}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k} \quad (2.2-33)$$

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_k \quad (2.2-34)$$

$$\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{A} \mathbf{p}_k \quad (2.2-35)$$

3) Check convergence:

```

if  $\|\mathbf{r}_k\| \leq \delta_r \|\mathbf{r}_0\|$  then
    Return
else
    if  $k > \text{iteration limit}$  then
        Return
    else
         $k = k + 1$ 
        Go to 2)
    end if
end if

```

note: a second convergence test exists that compares the square root of the dot product between \mathbf{z}_{k-1} and \mathbf{r}_{k-1} to the square root of the initial dot product. This test is evaluated after equation (30) of step 2). If this test is activated and true, then the algorithm is exited. If false or deactivated, the algorithm continues as indicated.

The costly operations in the above algorithm are represented by the preconditioning step, equation (30), and the matrix-vector product required by equations (33) and (35). Performance of the preconditioning step is discussed in conjunction with the examination of available preconditioning matrices made below. Because the matrix \mathbf{A} is never formed on the global level, the matrix-vector product is computed in blocks of similar, nonconflicting elements using the element computation algorithms of Section 2.3. FORTRAN code for the step

length calculation defined by equation (33) is given in Appendix B for each of the fully implemented element computation algorithms.

The key to the performance of the linear preconditioned conjugate algorithm is the choice of a preconditioning matrix, represented by the matrix \mathbf{B} in equation (30). Defining the "A" norm as

$$\| \mathbf{x} \|_A = \mathbf{x}^T \mathbf{A} \mathbf{x} \quad (2.2-36)$$

the rate of convergence in this norm is given by

$$\| \mathbf{x} - \mathbf{x}_k \|_A = \| \mathbf{x} - \mathbf{x}_0 \|_A \left[\frac{1 - \sqrt{\kappa}}{1 + \sqrt{\kappa}} \right]^{2k} \quad (2.2-37)$$

where κ is the condition number

$$\kappa = \lambda_{\max}(\mathbf{B}^{-1}\mathbf{A}) / \lambda_{\min}(\mathbf{B}^{-1}\mathbf{A}) \quad (2.2-38)$$

and λ_{\max} and λ_{\min} are the maximum and minimum eigenvalues of $\mathbf{B}^{-1}\mathbf{A}$. [15],[34],[39]. The preconditioning matrix should resemble the inverse of \mathbf{A} so that κ approaches unity and convergence is enhanced, and it should also be a relatively trivial matter to invert the preconditioning matrix. Below are presented the preconditioning matrices available in NLDfEP.

2.2.3.1 DIAGONAL PRECONDITIONER

The first and simplest preconditioning matrix in NLDfEP is the diagonal preconditioner

$$\mathbf{B} = \text{diag}(\mathbf{A}) \quad (2.2-39)$$

which represents diagonal scaling or an acceleration of the Jacobi iterative method [34]. Instead of using the current estimate of the dynamic tangent stiffness \mathbf{A} , it is also possible to employ the diagonal elements of the current estimate of the tangent stiffness or the mass matrix as the preconditioner, although no real advantage results since \mathbf{A} must be available in some form (in the case of NLDfEP upper triangular storage by element) to calculate the step length and the linear residual in equations (33) and (35). Solution of equation (30) using the diagonal preconditioner is accomplished on the global level, as it consists merely of a simple vector multiply.

2.2.3.2 EBE PRECONDITIONER

The second preconditioner available in NLDfEP is the Crout element-by-element preconditioner [39]. This preconditioner is an attractive one because it conforms to the

element format of data storage inherent in the finite element method and it provides an ideal vehicle for the element computation algorithms of Section 2.3. The preconditioner consists of the product decomposition

$$\mathbf{B} = \mathbf{W}^{1/2} \times \prod_{e=1}^{N_{el}} \mathbf{L}_p^e \times \prod_{e=1}^{N_{el}} \mathbf{D}_p^e \times \prod_{e=N_{el}}^1 \mathbf{U}_p^e \times \mathbf{W}^{1/2} \quad (2.2-40)$$

where

$$\mathbf{W} = \text{diag}(\mathbf{A}) \quad (2.2-41)$$

and \mathbf{L}_p^e , \mathbf{D}_p^e , \mathbf{U}_p^e are the lower triangular, diagonal, and upper triangular matrices of the Crout factorization of the corresponding Winget regularized element matrix defined by

$$\bar{\mathbf{A}}^e = \mathbf{I} + \mathbf{W}^{-1/2} (\mathbf{A}^e - \mathbf{W}^e) \mathbf{W}^{-1/2}; \quad \mathbf{W}^e = \text{diag}(\mathbf{A}^e) \quad (2.2-42)$$

The reverse element ordering in the upper triangular product of equation (40) insures that \mathbf{B} is symmetric, and the Winget regularization dictates that the regularized element matrix be positive-definite. Since the regularized element matrix is also symmetric, \mathbf{L}_p^e is the transpose of \mathbf{U}_p^e and need not be computed or require additional storage. The upper triangular and diagonal matrix factors for a given element are computed by equations (43)–(46), for each matrix column k as k goes from one to the number of element degrees of freedom. The fac-

$$\mathbf{U}_{ik}^* = \bar{\mathbf{A}}_{ik}^e; \quad i = 1, k-1 \quad (2.2-43)$$

$$\mathbf{U}_{ik}^* = \bar{\mathbf{A}}_{ik}^e - \sum_{j=1}^{i-1} \mathbf{U}_{ji}^* \mathbf{U}_{jk}^*; \quad i = 2, k-1 \quad (2.2-44)$$

$$\mathbf{U}_{ik} = \frac{\mathbf{U}_{ik}^*}{\mathbf{D}_{ii}^e}; \quad i = 1, k-1 \quad (2.2-45)$$

$$\mathbf{D}_{kk}^e = \bar{\mathbf{A}}_{kk}^e - \sum_{j=1}^{k-1} \mathbf{U}_{jk}^* \mathbf{U}_{jk}^*; \quad i = 1, k-1 \quad (2.2-46)$$

torization is performed for all elements each time the matrix \mathbf{A} is recomputed in the course of the nonlinear iterative solution. In practice, all element matrices are stored and manipulated in a compact upper triangular vector form illustrated in Appendix A. Performance of the element regularizations and factorizations are accomplished in blocks of similar, nonconflicting elements using the element computation algorithms of Section 2.3. FORTRAN code for the regularization and factorization process is given in Appendix B for each of the fully implemented element computation algorithms.

The steps required to solve equation (30) given the preconditioning matrix of equation (40) are listed as follows:

- 1) Global diagonal scaling

$$\mathbf{z}_0^* = \mathbf{W}^{-1/2} \mathbf{r}_{k-1} \quad (2.2-47)$$

- 2) Element forward reduction

$$\mathbf{z}_i^* = (\mathbf{L}_p^i)^{-1} \mathbf{z}_{i-1}^* ; \quad i = 1, N_{el} \quad (2.2-48)$$

- 3) Element diagonal scaling

$$\hat{\mathbf{z}}_0 = \mathbf{z}_{N_{el}}^* \rightarrow \hat{\mathbf{z}}_i = (\mathbf{D}_p^i)^{-1} \hat{\mathbf{z}}_{i-1} ; \quad i = 1, N_{el} \quad (2.2-49)$$

- 4) Element back substitution

$$\hat{\mathbf{z}}_{N_{el}+1} = \hat{\mathbf{z}}_{N_{el}} \rightarrow \hat{\mathbf{z}}_i = (\mathbf{U}_p^i)^{-1} \hat{\mathbf{z}}_{i+1} ; \quad i = N_{el}, 1 \quad (2.2-50)$$

- 5) Global diagonal scaling

$$\mathbf{z}_{k-1} = \mathbf{W}^{-1/2} \hat{\mathbf{z}}_1 \quad (2.2-51)$$

The element operations implied by equations (48) and (50) are again executed in blocks of similar, nonconflicting elements. FORTRAN code for these operations is also given in Appendix B for each of the fully implemented element computation algorithms. Element diagonal scaling is achieved at the global level through the equation

$$\hat{\mathbf{z}}_{N_{el}} = \hat{\mathbf{W}}^{-1} \mathbf{z}_{N_{el}}^* \quad (2.2-52)$$

where

$$\hat{\mathbf{W}}^{-1} = \prod_{e=1}^{N_{el}} \mathbf{D}_p^{e-1} \quad (2.2-53)$$

is premultiplied during the regularization and factorization procedure. FORTRAN code for the premultiplication is included with the code for regularization and factorization.

As with the diagonal preconditioner, it is also possible to employ the Crout factorization of the current estimate of the tangent stiffness or the mass matrix as the preconditioner, although again no real advantage will accrue.

2.2.3.3 HIERARCHICAL PRECONDITIONER

The final preconditioner considered in NLDfEP is the hierarchical preconditioner. Strictly speaking, this preconditioner is not suitable for eight node isoparametric

elements, as in this case the preconditioning matrix reduces to the matrix **A** itself. The hierarchical preconditioner is based on the use of hierarchical shape functions and variables and is applicable to higher order elements. The only higher order element in NLD FEP is the twenty node isoparametric element; in this case the hierarchical displacement variables are the corner node displacements and midside variables that represent departures from linearity. In contrast, the displacement variables associated with the serendipity approach normally used in NLD FEP are displacements at all nodes. The relationship between the midside displacements and the midside hierarchical displacement variables along an edge of a twenty node isoparametric element is given by

$$\begin{aligned} U_3 &= \frac{(U_1 + U_2)}{2} - U_3^h \\ V_3 &= \frac{(V_1 + V_2)}{2} - V_3^h \\ W_3 &= \frac{(W_1 + W_2)}{2} - W_3^h \end{aligned} \quad (2.2-54)$$

where the subscripts 1 and 2 refer to corner nodes, the subscript 3 to the midside node, the superscript h to a hierarchical variable, and U, V, and W to the x, y, and z displacements. The foundation for this relationship is given in Fig. 2.2-3 [85]. Defined in this way, the hierarchical displacement variables actually represent a departure to linearity, as when they are added to the actual displacements they produce a linear variation along the edge. Based on this relationship, the transformations between the element nodal displacement vector and the element hierarchical displacement vector are

$$\begin{aligned} U^h &= T^h U \\ U &= T^h U^h \end{aligned} \quad (2.2-55)$$

Note that the same transformation matrix acts in both directions. This property results from the definition of the hierarchical displacement variables as a departure to linearity. The structure of the transformation matrix is given in Appendix A. Applying the principle of contragradience, the transformations between element nodal and hierarchical force vectors are

$$\mathbf{F}^h = (\mathbf{T}^h)^T \mathbf{F} \quad (2.2-56)$$

$$\mathbf{F} = (\mathbf{T}^h)^T \mathbf{F}^h$$

and element hierarchical mass and tangent stiffness matrices are obtained from their serendipity counterparts through

$$\mathbf{M}^h = (\mathbf{T}^h)^T \mathbf{M} \mathbf{T}^h \quad (2.2-57)$$

$$\mathbf{K}^h = (\mathbf{T}^h)^T \mathbf{K} \mathbf{T}^h$$

Once the element hierarchical mass and tangent stiffness matrices are formed, combined, and assembled into the global hierarchical dynamic tangent stiffness matrix $(\mathbf{K}_t^d)^h$, the hierarchical preconditioning matrix is defined as

$$\mathbf{B} = \begin{bmatrix} (\mathbf{K}_t^d)_{ll}^h & 0 \\ 0 & (\mathbf{D}_t^d)_{hh}^h \end{bmatrix} \quad (2.2-58)$$

where the subscript l refers to lower order or corner nodes, and the subscript h refers to higher order or midside nodes. The matrix $(\mathbf{D}_t^d)_{hh}^h$ contains only the diagonal elements of $(\mathbf{K}_t^d)_{hh}^h$. The advantage of the hierarchical preconditioning matrix is that, while it still must be formed globally and factorized, its size and bandwidth is significantly smaller than that of the original problem. Also, it has been shown that for two dimensional problems the number of iterations required for convergence of the linear preconditioned conjugate gradient algorithm is bounded as the mesh is refined when using the hierarchical preconditioner [18].

There are drawbacks to using the hierarchical preconditioner. First, one still has to form a global stiffness matrix. One of the primary advantages of the linear preconditioned conjugate gradient method is that it is not necessary to assemble and factorize a global stiffness matrix, which can become so large that the speed of the supercomputer is offset by prolific disk I/O. The size of the global stiffness matrix can even become so great that the virtual memory capacity of the supercomputer is exceeded. Although the size and bandwidth of the corner node stiffness will be smaller than that of the overall stiffness, for large problems they could still be prohibitive. Consider also that even though a limit on iterations may exist as the

mesh is refined, there is no assurance that this limit will be low enough to insure that the cost of using a direct solver on the corner node stiffness will be offset by greatly reduced iterations.

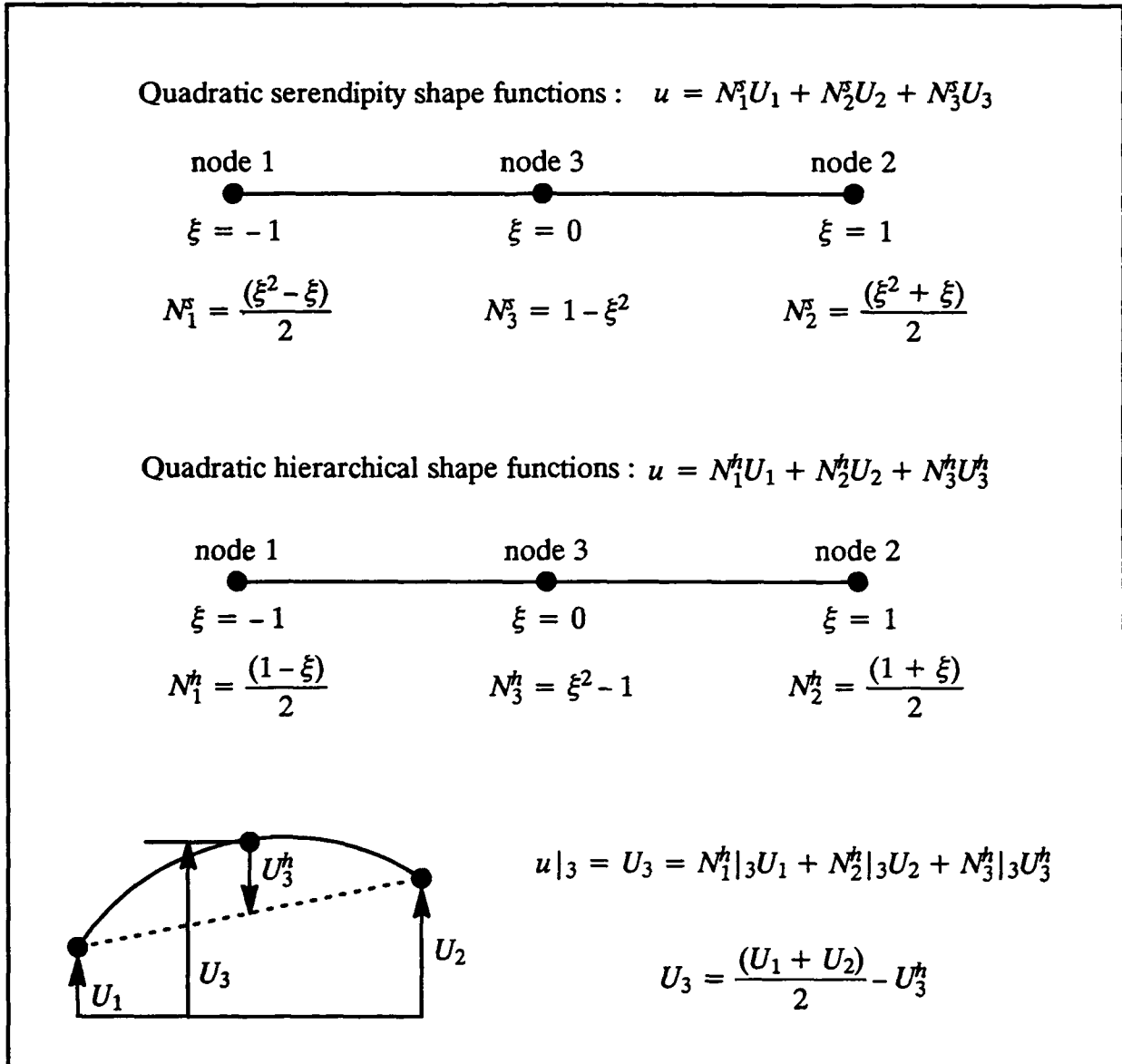


Fig. (2.2-3), relationship between serendipity and hierarchical variables, 1D element

The second drawback is a simple one mentioned above: hierarchical preconditioning is useless for lower order elements such as the eight node brick. A great many problems can be solved with these elements efficiently, and an effective preconditioner should encompass these problems as well.

The third drawback is that the hierarchical preconditioner is fundamentally incompatible with programs such as NLDFEP that are built upon serendipity elements and which employ other preconditioners. In order to use the hierarchical preconditioner, the equations of motion have to be couched in terms of hierarchical variables. This means that widespread transformations must be performed selectively when use of the hierarchical preconditioner is specified. This is a process that is ripe for error. Currently, implementation of the hierarchical preconditioner in NLDFEP is in a state of disarray; many roaches must be exterminated. Consequently its use has not been investigated in this thesis.

Although there are problems with its implementation, the hierarchical preconditioner merits investigation – a topic for further research. Probably a separate code should be created to explore its use, instead of attempting to integrate it into the existing code. Of course, this separate code would only apply to 20 node isoparametric elements. An intriguing possibility would be to use a linear preconditioned conjugate gradient algorithm with a simple preconditioner (say diagonal) to invert the corner node stiffness within the outer algorithm. In this way, a global stiffness matrix would not be required and perhaps the corner node stiffness would be well conditioned enough to make the nested system efficient.

2.2.4 NONLINEAR PRECONDITIONED CONJUGATE GRADIENT

Whereas the linear preconditioned conjugate gradient algorithm is employed to solve the linear system of equations resulting from the nonlinear solution algorithm, the nonlinear preconditioned conjugate gradient algorithm is a nonlinear solution method. Originally it was designed to employ one of the preconditioners discussed above, but preliminary testing showed that this was hopelessly costly in light of the number of iterations that were required for the convergence of the first time step and the calculations necessary for each iteration, such as a stress recovery, an internal force vector update, and possibly a dynamic stiffness matrix update. It was then adapted to use the current estimate of the dynamic tangent stiffness as the nonlinear preconditioner and to invert the preconditioning matrix using the linear preconditioned conjugate gradient algorithm. In this form, the nonlinear algorithm becomes an accelerated Newton–Raphson method while retaining the benefits of a conjugate gradient algorithm. Belatedly, it was discovered to closely resemble the conjugate–Newton method developed by Irons and Elsawaf [42], but with definite differences. The nonlinear preconditioned conjugate gradient algorithm to advance the solution from t_n to t_{n+1} thus proceeds as follows:

- 1) Initialize:

update initial residual (equation (20))

for $j = 1, N_{eq}$; if j is a constrained dof,

$$(\Delta u_{n+1}^0)_j = c_j$$

$$(R_{n+1}^0)_j = 0$$

else

$$(\Delta u_{n+1}^0)_j = 0$$

end if

$k = 1$

note: as stated previously, non-zero displacement constraints (c_j) are placed in the total increment of displacement vector at the beginning of each step and corresponding residual entries are set to zero.

2) Compute:

update the dynamic tangent stiffness, if necessary

$$z_{n+1}^{k-1} = (K_t^d)^{-1} R_{n+1}^{k-1} \quad (2.2-59)$$

compute β_k ($\beta_1 = 0$)

$$p_{n+1}^k = z_{n+1}^{k-1} + \beta_k p_{n+1}^{k-1} \quad (p_{n+1}^0 = 0) \quad (2.2-60)$$

compute α_k with line search

note: the residual for iteration k is updated during the line search operation.

$$\Delta u_{n+1}^k = \Delta u_{n+1}^{k-1} + \delta u_{n+1}^k ; \quad \delta u_{n+1}^k = \alpha_k p_{n+1}^k \quad (2.2-61)$$

3) Check convergence:

```

if converged then
  Return
else
  if k > iteration limit then
    Return
  else
    k = k + 1
    Go to 2)
  end if
end if

```

note: the convergence tests used here are identical to those given for the modified Newton-Raphson algorithm above.

Again, equation (59) is solved using the linear preconditioned conjugate gradient algorithm. It remains to discuss the computation of the search direction multiplier β_k and the step length α_k .

2.2.4.1 SEARCH DIRECTION MULTIPLIER

The search direction multiplier β_k is determined by either the equation

$$\beta_k = \frac{(z_{n+1}^{k-1})^T (R_{n+1}^{k-1} - R_{n+1}^{k-2})}{(z_{n+1}^{k-2})^T R_{n+1}^{k-2}} \quad (2.2-62)$$

or

$$\beta_k = - \frac{(z_{n+1}^{k-1})^T (R_{n+1}^{k-1} - R_{n+1}^{k-2})}{(p_{n+1}^{k-1})^T (R_{n+1}^{k-1} - R_{n+1}^{k-2})} \quad (2.2-63)$$

Equation (62) represents the Polak-Ribiere form of β_k [69] while equation (63) follows the form suggested by Sorenson [79] to ensure the orthogonality of the current search direction and the last residual increment. By substituting equation (63) into equation (60) the relationship

$$(p_{n+1}^k)^T (R_{n+1}^{k-1} - R_{n+1}^{k-2}) = 0 \quad (2.2-64)$$

is satisfied, establishing the desired orthogonality. Applying the mean value theorem to equation (13) and accounting for equation (61) leads to

$$\mathbf{R}_{n+1}^{k-1} - \mathbf{R}_{n+1}^{k-2} = -\alpha_k \bar{\mathbf{K}}_t^d \mathbf{p}_{n+1}^{k-1} \quad (2.2-65)$$

where $\bar{\mathbf{K}}_t^d$ is the the actual (not the estimate) dynamic tangent stiffness evaluated at some point between iteration $k-1$ and iteration $k-2$. Combining (64) and (65) gives, assuming a nonzero step length,

$$(\mathbf{p}_{n+1}^k)^T \bar{\mathbf{K}}_t^d \mathbf{p}_{n+1}^{k-1} = 0 \quad (2.2-66)$$

Thus, use of the Sorenson β_k assures that consecutive search directions will be orthogonal with respect to some dynamic tangent stiffness matrix intermediate to the previous iterations. Use of a closed form expression for β_k is the primary difference between the nonlinear preconditioned conjugate gradient method presented here and the conjugate-Newton method. In the conjugate-Newton method, a numerical procedure based on information gathered during the line search is used to force equation (66) to be true with respect to the dynamic tangent stiffness matrix at iteration $k-1$. This difference may or may not be significant. The numerical procedure could be integrated into NLDfEP as an alternative to equations (62) and (63).

2.2.4.2 STEP LENGTH: PERFORMING THE LINE SEARCH

The line search computes a step length α_k such that the current search direction is orthogonal to the residual resulting from the displacement update in equation (61), or

$$(\mathbf{p}_{n+1}^k)^T \mathbf{R}_{n+1}^k |_{\alpha_k} = 0 \quad (2.2-67)$$

For the linear preconditioned conjugate gradient algorithm, this operation represents the minimizing of a functional over all previous search directions and α_k can be expressed in the closed form of equation (33). For the nonlinear algorithm, equation (67) provides for a zero residual in the current search direction, or a mild statement of equilibrium in that direction.

The line search algorithm proceeds first by bracketing the root, or finding a step length such that the sign of the dot product of equation (67) is reversed. This is done by assuming an initial step length of one, and successively doubling the step length until the root is bracketed. If the root cannot be bracketed after a reasonable number of doublings a step length of one is adopted and the line search is exited. For the linear preconditioned conjugate gradient algorithm it can be shown that the final step length and the dot product for zero step length are nonnegative. For a nonlinear algorithm where the preconditioning matrix is not constant and material nonlinearities exist so that a potential energy functional is not defined,

the situation is somewhat muddled. However, the assumption is that the above statement of nonnegativity holds.

Once bracketing is achieved, the root is located by using a false position-bisection technique [64]. A first estimate of the step length is calculated by bisecting the current bracket values according to

$$a_k^b = \frac{(a_k^R + a_k^L)}{2} \quad (2.2-68)$$

where the superscript b refers to the bisection estimate, the superscript R to the right bracket, and the superscript L to the left bracket. The brackets are then reset so that the calculated step length becomes the new left or right bracket, depending on whether the dot product associated with the calculated step length is positive or negative. A second estimate of the step length is then obtained using the false position relation

$$a_k^i = a_k^L - \frac{(\mathbf{p}_{n+1}^k)^T \mathbf{R}_{n+1}^k |_{a_k^L} (a_k^R - a_k^L)}{(\mathbf{p}_{n+1}^k)^T \mathbf{R}_{n+1}^k |_{a_k^R} - (\mathbf{p}_{n+1}^k)^T \mathbf{R}_{n+1}^k |_{a_k^L}} \quad (2.2-69)$$

where the superscript i refers to the line search iteration number. Convergence is tested based on the new estimate. If convergence has not been attained, the brackets are reset and the process is repeated until either convergence has been met or a limit on the number of iterations has been reached. The convergence tests used in the line search algorithm are

- 1) $\| a_k^i - a_k^{i-1} \| \leq \delta_1 (a_k^R - a_k^L) |_0$
- 2) $\| (\mathbf{p}_{n+1}^k)^T \mathbf{R}_{n+1}^k |_{a_k} \| \leq \delta_2 \| (\mathbf{p}_{n+1}^k)^T \mathbf{R}_{n+1}^k |_0 \|$

The quantity on the right hand side of the first convergence test represents the initial bracket length.

Each time a new step length is calculated and a new residual is required, the stress recovery and internal force vector computations must be performed. These computations utilize the element computation algorithms of Section 2.3. As is discussed in detail in Section 2.4, the stress recovery produces a trial state of stress at the current estimate of the solution at time t_{n+1} , this estimate corresponding to the varying step length in the line search, that is then used to calculate the internal force vector. The stress recovery strategy is "path independent" and the material history, outside of the trial state at the current estimate, is not updated until there is global convergence to an acceptable estimate of the solution at time t_{n+1} . The line search can be an extremely costly process if many iterations are required to reduce equation (67)

sufficiently close to zero. This is the trade off inherent in the nonlinear preconditioned conjugate gradient algorithm: fewer global equilibrium iterations may be executed but the line search may prove excessively expensive.

2.3 ELEMENT COMPUTATION ALGORITHMS

Element computations form the backbone of nonlinear dynamic structural analysis using the finite element method. Element computations include the stress recovery process, matrix-vector products, and the determination of the internal force vector, the tangent stiffness matrix, and the mass matrix. In order to optimize a finite element solution on a supercomputer, these calculations must be done as efficiently as possible. This involves exploiting the opportunities for concurrency and vectorization inherent in the finite element method to the utmost degree. This section deals with the development and implementation of various algorithms designed to accomplish this task.

2.3.1 COMPUTATIONAL HIERARCHY

Use of three dimensional isoparametric elements within the finite element method allows for multiple levels of computation that can be readily optimized for concurrency and vectorization. This echelon consists of a structure or global level, an element level, and a material point level which is a consequence of the isoparametric formulation. These levels comprise a computational hierarchy to which concurrency and vectorization may be applied in various combinations to discover the best combination for a given machine architecture. The remainder of this subsection describes the levels of the hierarchy in more detail.

2.3.1.1 LEVEL ONE: STRUCTURE

Computations at the structure level include the factorization and back substitution operations on a symmetric, positive definite banded matrix for those nonlinear solution algorithms that require a direct solver, and global vector operations, mandated by the nonlinear solution algorithm, of the form $\mathbf{a} + \alpha \mathbf{b}$, where \mathbf{a} and \mathbf{b} are vectors and α is a scalar. An operation of this type is termed a linked triad [63]. It is generally profitable to perform these global vector operations at the structure level because the global vectors are typically more than large enough to make concurrency and vectorization applied at the structure level efficient. The alternative would be to perform the global linked triads at the element level, which is considered in the following discussion of element level calculations.

The structure level is a factor in all element and material point level calculations through the communication of data accomplished by gather/scatter operations. In a gather

operation, data is transferred from global storage to a form usable at the element or material point level; in a scatter operation, the result of the element or material point level calculation is transferred back to global storage. Gather/scatter operations are illustrated in Fig. 2.3-1. In the element computation algorithms, this is the role that the structure level plays.

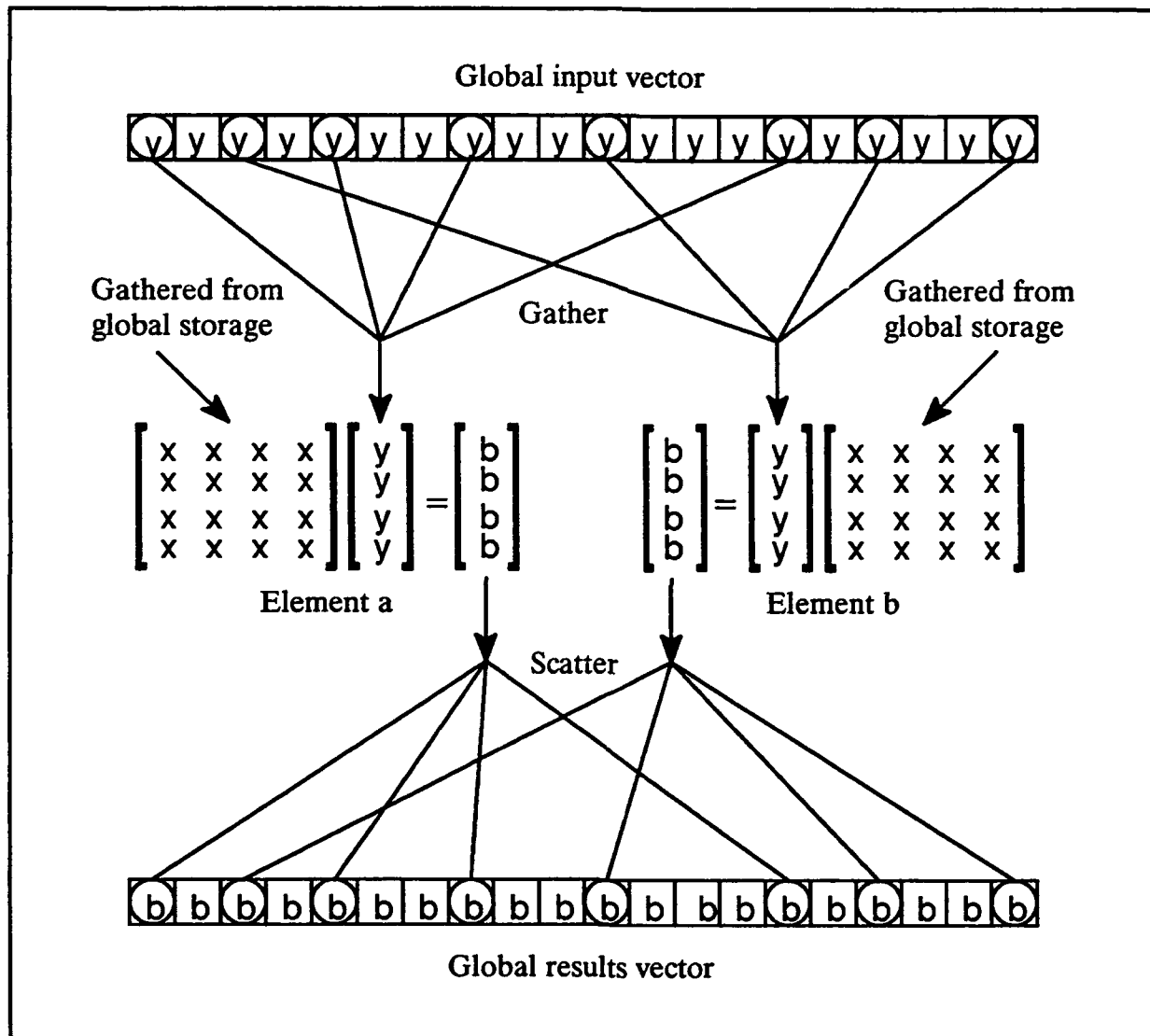


Fig. (2.3-1), element-by-element matrix-vector multiply

2.3.1.2 LEVEL TWO: ELEMENT

The finite element method provides a natural domain decomposition technique, where the structure is partitioned into discrete subdomains, such as elements, for the purpose of reducing computations at the structure level. Calculations performed solely at the element level can take place independently as long as each element has unimpeded access to the

appropriate element data. Interaction with the structure level involves the gather/ scatter of the element data. An example of such a calculation is a global matrix-vector product. Instead of multiplying the global matrix by the global vector, the computation can be done element-by-element with each element matrix-vector product being done independently. As shown in Fig 2.3-1, the element matrix and input vector are first gathered, then multiplied, and finally the result is scattered back to the structure level. This type of calculation is ripe for concurrency and vectorization, provided that the elements involved are nonconflicting in the sense that they have no global degrees of freedom in common. Further, global matrix-vector operations performed at the element level are compatible with storing such global matrices as the tangent stiffness and tangent mass matrices by elements, which simplifies matrix gather/scatter operations and reduces memory usage.

As mentioned above, it would be possible to perform global vector operations such as the linked triad at the element level, but there are at least two key drawbacks. First, if the global vectors are stored in their natural global format, the necessary gather/scatter operations to and from the element level would be overly costly compared to the simplicity of a structure level computation. Second, if the global vectors are stored by elements, making gather/scatter much simpler, then the amount of storage required will generally be greater than that required by a structure level storage format. This disparity will increase as the size of the problem increases. Thus, in this research, global vector operations are performed at the structure level.

Element level matrix-vector products occur in the linear preconditioned conjugate gradient algorithm and in updates of the global residual when the consistent mass matrix is employed. In general, element level computations are burdened with the added complexity of coping with the need for subdivision to the material point level. This level of the computational hierarchy is discussed next.

2.3.1.3 LEVEL THREE: MATERIAL POINT

The isoparametric element formulation provides for a third level of computation, namely that at the material points. Material point calculations arise from the necessity for numerical integration of element integrals. Numerical integration is required because of the mapping inherent in the isoparametric formulation. There are two types of mapping, one which concerns the determination of the initial geometry and the other which refers to the resolution of the current configuration. The initial geometry is established by interpolation of the initial coordinates in a parametric space. The current configuration is constructed from

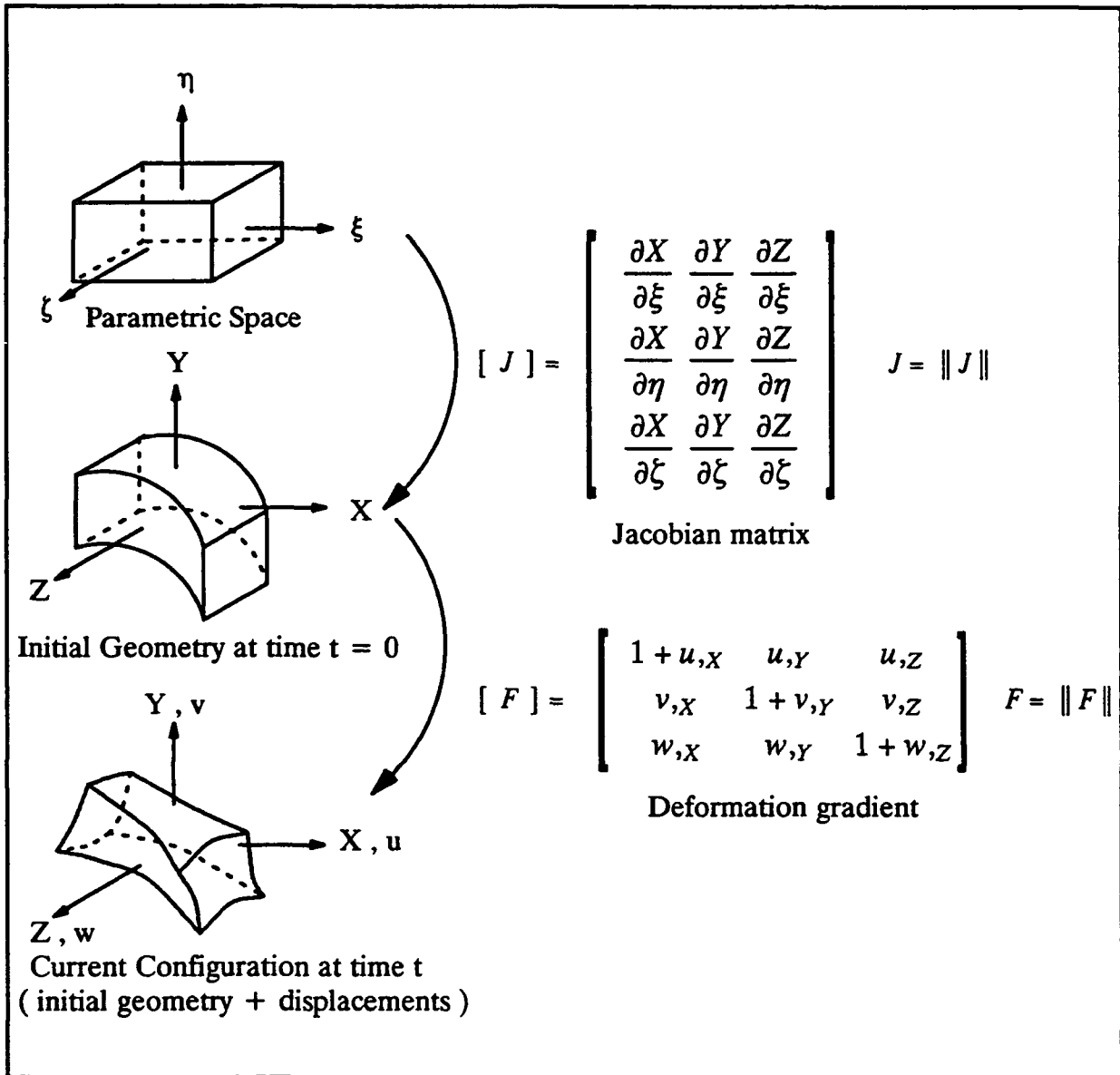


Fig. (2.3-2), mapping from parametric to reference to current configuration

the initial geometry and through interpolation of the element displacements in the same parametric space. Consequently, as seen in Fig 2.3-2, in order to perform an element integral information concerning the mapping of both initial coordinates and element displacements is required. This information takes the form of the Jacobian matrix J , the deformation gradient F , and their respective determinants. In general, both of these quantities will consist of a quotient of polynomials and will render the element integral impractical or impossible to resolve in a closed form. Element integrals thus have to be determined numerically, and the numerical integration technique used is Gauss quadrature. At each of a series of integration

or material points, the integrand is evaluated and multiplied by an appropriate weighting factor. The contributions of each material point are summed to produce the final element result. This process is demonstrated in Fig 2.3-3. In effect, each material point becomes an element itself, with element type calculations at the material points being performed independently of each other as long as each material point has unimpeded access to the appropriate element and material point data. An example of a calculation requiring material point computations and thus utilizing the entire spectrum of the computational hierarchy is

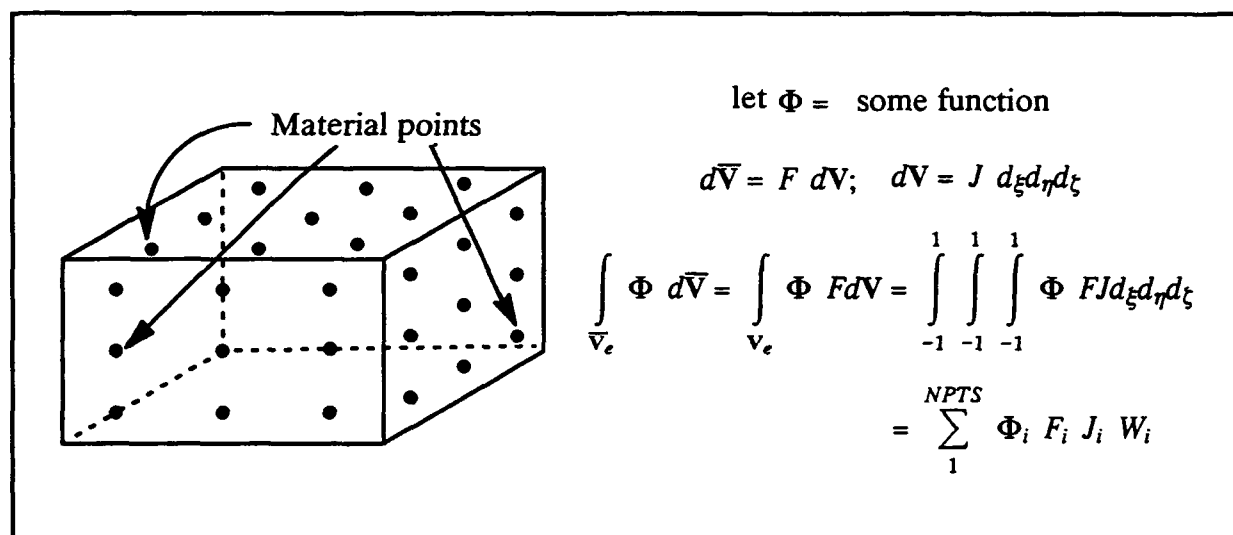


Fig. (2.3-3), Gauss quadrature

the determination of the global tangent stiffness matrix, for which the hierarchy of computations is shown in Fig. 2.3-4. Calculations represented by Fig. 2.3-4 are the primary focus of element computation algorithms.

2.3.2 CONSIDERATIONS FOR PARALLEL AND VECTOR ALGORITHMS

There are a number of considerations that bear directly upon the implementation and performance of parallel and vector algorithms suited to element computations. Among these are such concerns as whether to favor vectorization or concurrency, execution of gather/scatter operations, element reordering or blocking schemes, and appropriate element data structures. This subsection explores these considerations in some detail.

2.3.2.1 VECTORIZATION VS. CONCURRENCY

The nature of the optimizing algorithms employed are related directly to the architecture of the machines utilized. If the machine on which the application is to be run is a

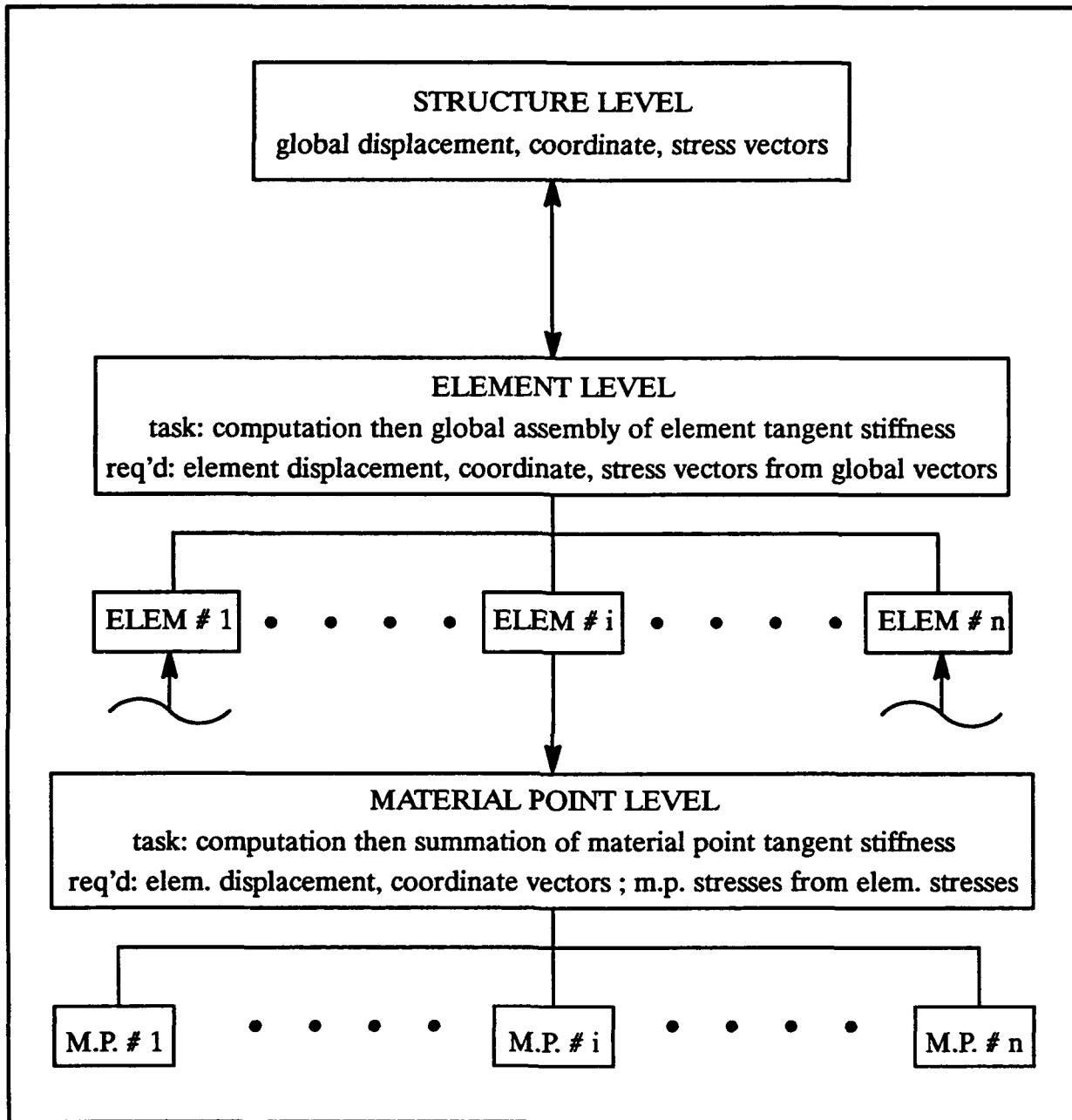


Fig. (2.3-4), computation hierarchy for the tangent stiffness

vector machine, then there is certainly no use in pondering the role of concurrency. The machines used in this research have the capability of both concurrency and vectorization, being comprised of a given number of vector processors that perform in concert. For such machines, the programmer must first contemplate whether to apply concurrency in a multitasking or microtasking mode. When multitasking, the overall computation is divided

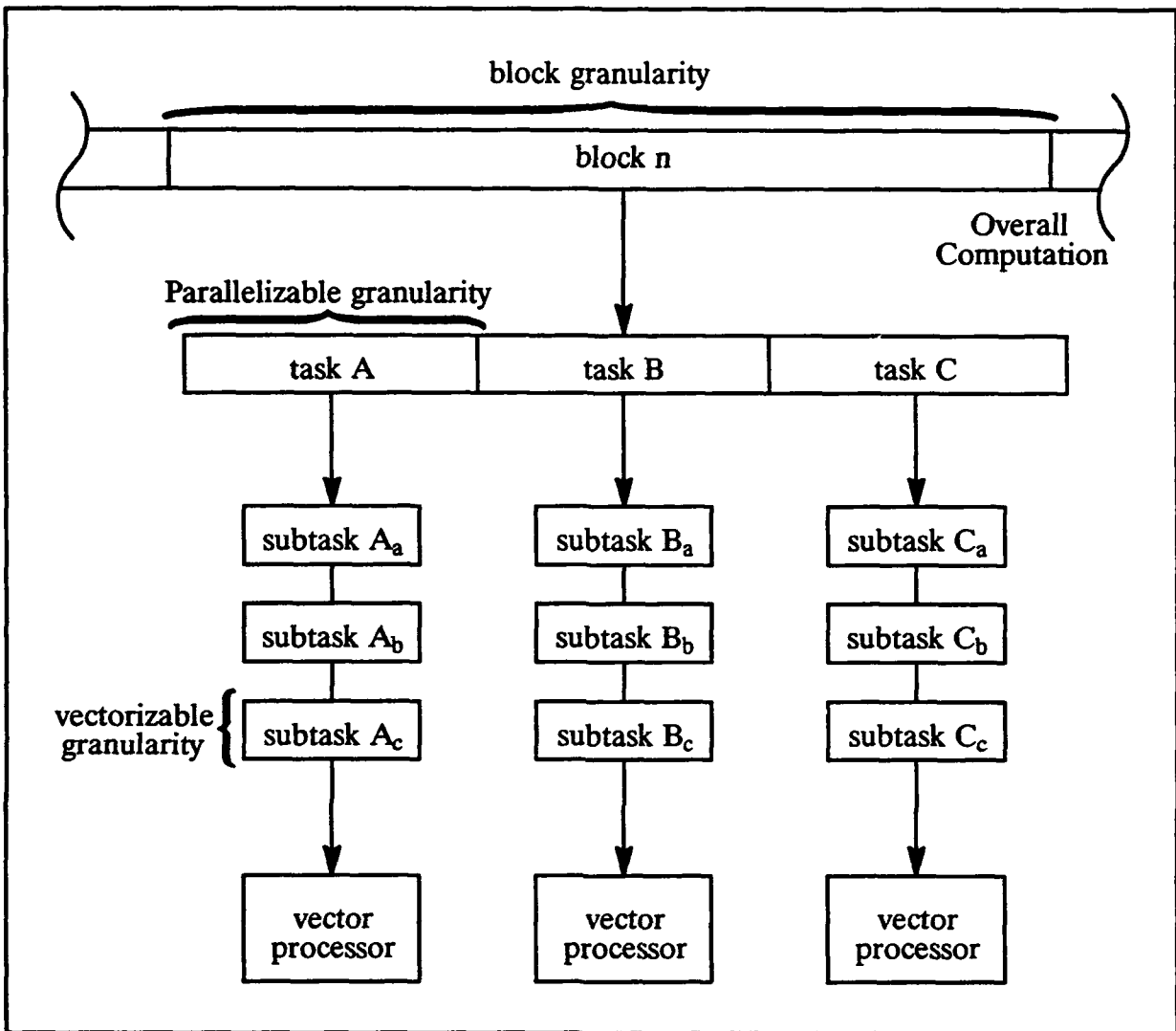


Fig. (2.3-5), symbolic representation of multitasking

into substantial, comparably sized tasks which are assigned to individual vector processors for independent processing. If there are more tasks than vector processors, then the tasks are executed in blocks. The size of the tasks is termed the parallelizable granularity, and the aggregate size of the tasks in a block is called the block granularity. Each of the tasks should exhibit an exploitable vectorizable granularity, which is a subset of the task appropriate for vectorization. Multitasking is symbolically represented in Fig. 2.3-5. When microtasking, the parallelizable granularity is much smaller and concurrency is often applied incidentally in the sense that it results from the automatic optimization of the compiler rather than programmer design. Vector-concurrent loops and concurrent outer-vector inner nested loops in

FORTTRAN are examples of incidental concurrent microtasking. The main focus then is a significant vectorizable granularity. Microtasking is symbolically represented in Fig. 2.3-6.

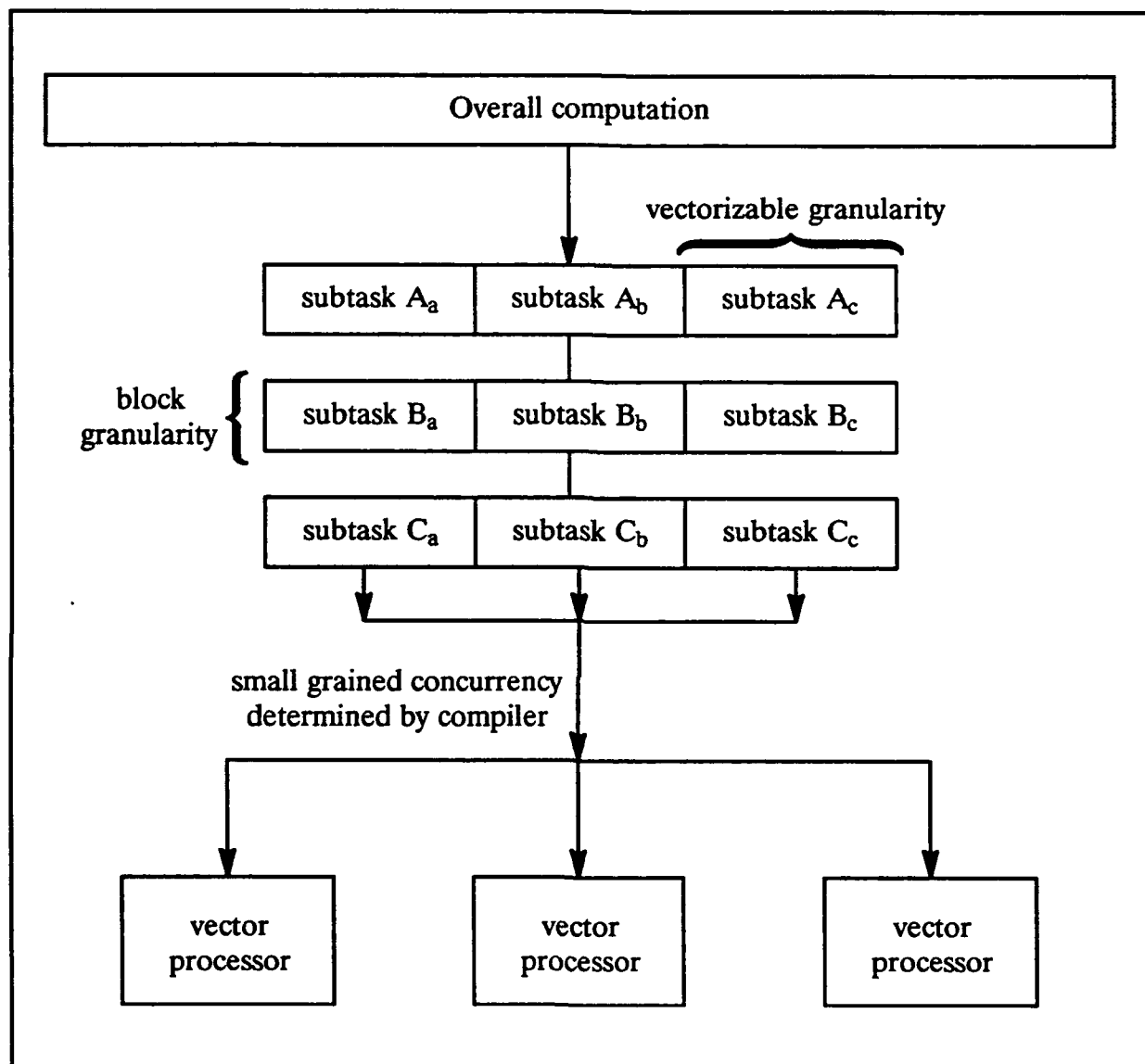


Fig. (2.3-6), symbolic representation of microtasking

If the programmer stipulates a large multitasking parallelizable granularity with a designed vectorizable granularity, then vectorization is favored over concurrency. Likewise, if the overall computation is subdivided into large granularity, sequentially processed blocks of tasks with a designed vectorizable granularity and applying incidental concurrency, then vectorization is favored. On the other hand, if a smaller multitasking parallelizable granularity is specified over a larger number of vector processors while relying on an incidental vectorizable granularity, then concurrency is favored over vectorization. Again,

which emphasis is the most effective is dependent on the architecture of a given machine. For a machine like the Convex C240, which has four relatively powerful vector processors, then it is logical that vectorization should be emphasized over concurrency for best performance. Conversely, for a computer such as the Alliant FX/8, which possesses eight less powerful vector processors, one would expect to promote concurrency above vectorization to achieve the best results. All reasonable possibilities must be explored for a given architecture to determine which is most efficient.

2.3.2.2 GATHER/SCATTER OPERATIONS

Gather/scatter operations arise from the need to perform computations at the element level dictated by the finite element method. For a sequential application, the pertinent global data is gathered to the element level, the element computation is executed, and the result is scattered to the global level for each element in turn. To facilitate parallel and vector processing, however, special attention must be paid to the gather/scatter.

The main concern for gather operations is to avoid congestion during the data retrieval. As will be discussed later, the parallel and vector algorithms for element computations are based on processing blocks of elements. If the elements in the block have degrees of freedom in common, then the elements are said to be conflicting and signals referencing the global data will interfere with each other. This interference can cause a noticeable slowdown in the performance of the gather, and in some cases may lead to bus errors. To eliminate this problem, the element blocks should be nonconflicting with no degrees of freedom in common.

The main concern for scatter operations is similar to but more serious than that for gather operations. For scatter operations, one must prevent data dependencies. Data dependencies are also the consequence of conflicting element blocks, but where in most cases congestion simply slows down the gather, data dependencies can cause the scatter to be incorrect. If a result from more than one element is to be accumulated into the same global location, then if the scatter is done in a parallel or vector mode a lack of synchronization can allow some of the results to be ignored. This is illustrated in Fig. 2.3-7. In this figure, the contribution of element a is disregarded because both elements attempt to augment the same global degree of freedom and element a lags slightly behind element b. To dispense with data dependencies, element blocks should again be nonconflicting. Alternatively, to scatter an element vector for all the elements in the structure to the global level, an approach that concentrates on the nodal connectivity of the structure can be adopted [27]. The algorithm relies primarily on vectorization over the nodes. Considering for example the assembly of the

internal force vector, the scatter from the existing element vectors is vectorized over all the structural nodes with at least one element framing in, then those with at least two, etc, until finished. It is possible to scatter the x , y , and z degrees of freedom independently, so they can be processed in parallel. This algorithm has the advantage that it generally maintains long vector lengths and – if global storage by element exists for the quantity of interest – it can dispense with the necessity of some gather operations, as element blocks are no longer required to be nonconflicting and can be processed in the order of the element numbering. It has a number of disadvantages. First, unless the nodes are renumbered, a high degree of indirect addressing is required. Indirect addressing leads to reduced vector performance, and renumbering the nodes would destroy the half band width if a direct solver is used. Second, if there are large numbers of elements framing into the nodes, then a correspondingly large number of vector operations must be performed sequentially and the efficiency of the algorithm is diminished. Third, the results for all elements in the structure have to be calculated before the algorithm can proceed, forcing the results to be stored locally and wasting memory. Finally, the algorithm is even more complicated and wasteful for the scatter of matrices such as the tangent stiffness. In light of the above, it was decided to arrange blocks of nonconflicting elements.

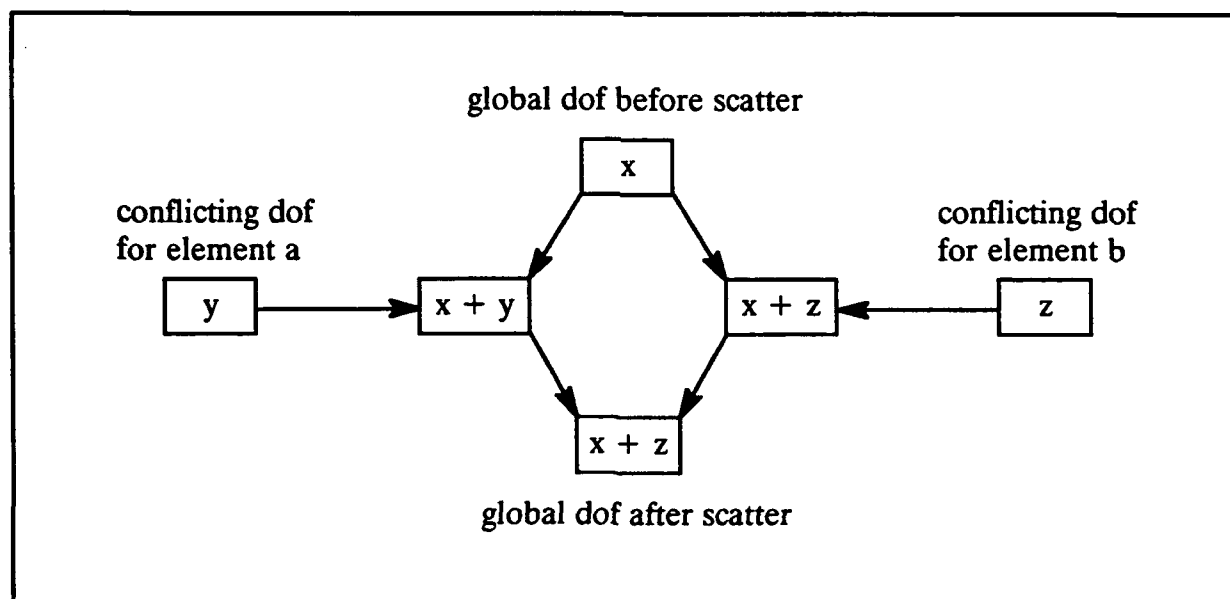


Fig. (2.3-7), concurrent scatter operation illustrating the effects of data dependencies

2.3.2.3 ELEMENT BLOCKING

Element blocking may be viewed as a symbolic reordering of the elements so that successive groups of nonconflicting elements can be processed simultaneously. The size and

arrangement of the element blocks depend upon the type of parallel-vector algorithm, the number of vector processors, and the length of the vector registers. In addition to the nonconflicting criterion, the elements in a block must be similar if vectorization over the block is to be performed. To be similar, the elements in the block must be of the same type, they must have the same number of material points, they must be uniformly geometrically linear or nonlinear, and if geometrically nonlinear their behaviour must be governed by the same stress rate. The condition of similarity need not be imposed if the parallel-vector algorithm calls for concurrent processing of the element block with incidental vectorization. In practice, as the condition is not overly restrictive, similarity is in general enforced for all parallel-vector algorithms.

2.3.2.4 DATA STRUCTURES

Special attention must also be paid to the element block data structure in order to promote concurrent and vector optimization. One issue to address is that of data locality. The basic memory architecture of the machines used in this research is hierarchical, as depicted in

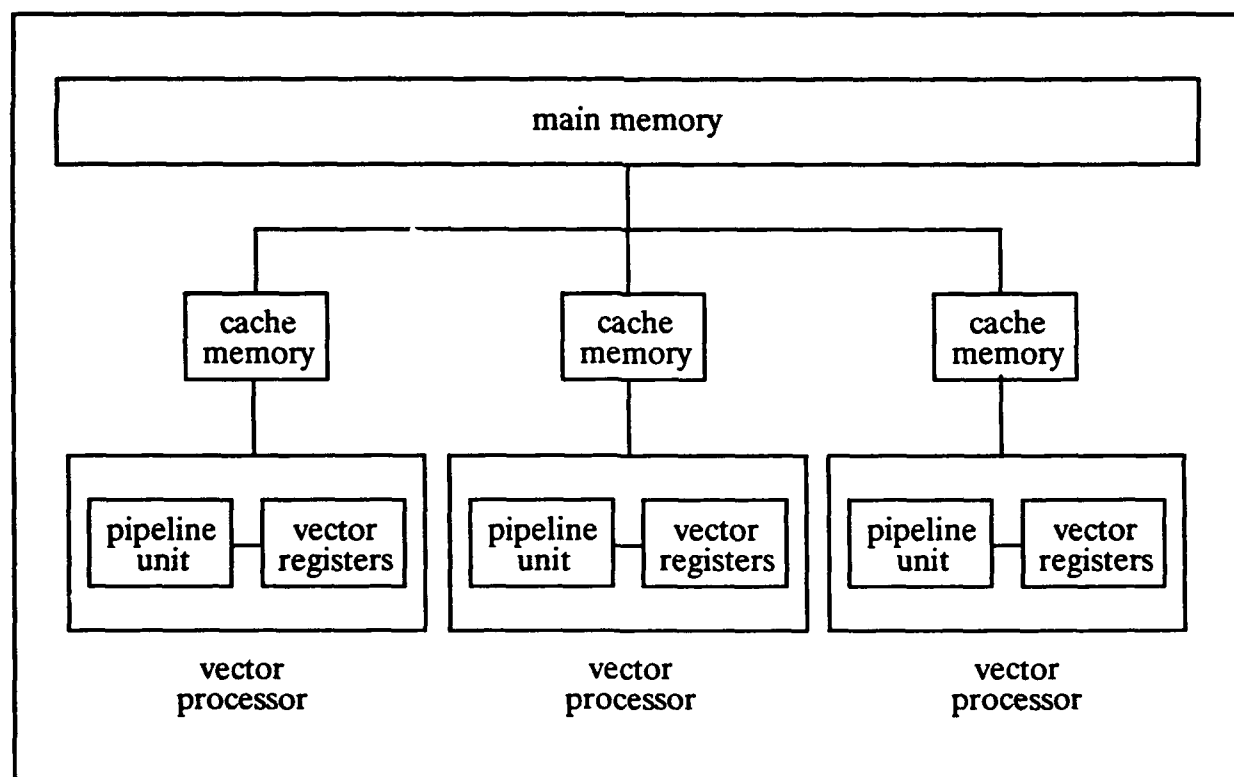


Fig. (2.3-8), symbolic hierarchical memory structure

Fig. 2.3-8. There is the main memory, a fast cache memory accessible by the vector processors, and the vector registers of the processors. The cache memory may be local to each

processor, or may be shared by multiple processors. For best performance, the data involved in an ongoing calculation on a vector processor should fit as nearly as possible into the fast cache memory so that the transfer time of data to and from the vector registers and the cache misses to fetch data not in the cache from main memory are minimized. In addition, vector lengths and thus the negotiable size of the element block data structure for those parallel-vector algorithms which rely on designed vectorization should correspond to the size of the vector registers in order to maximize efficiency.

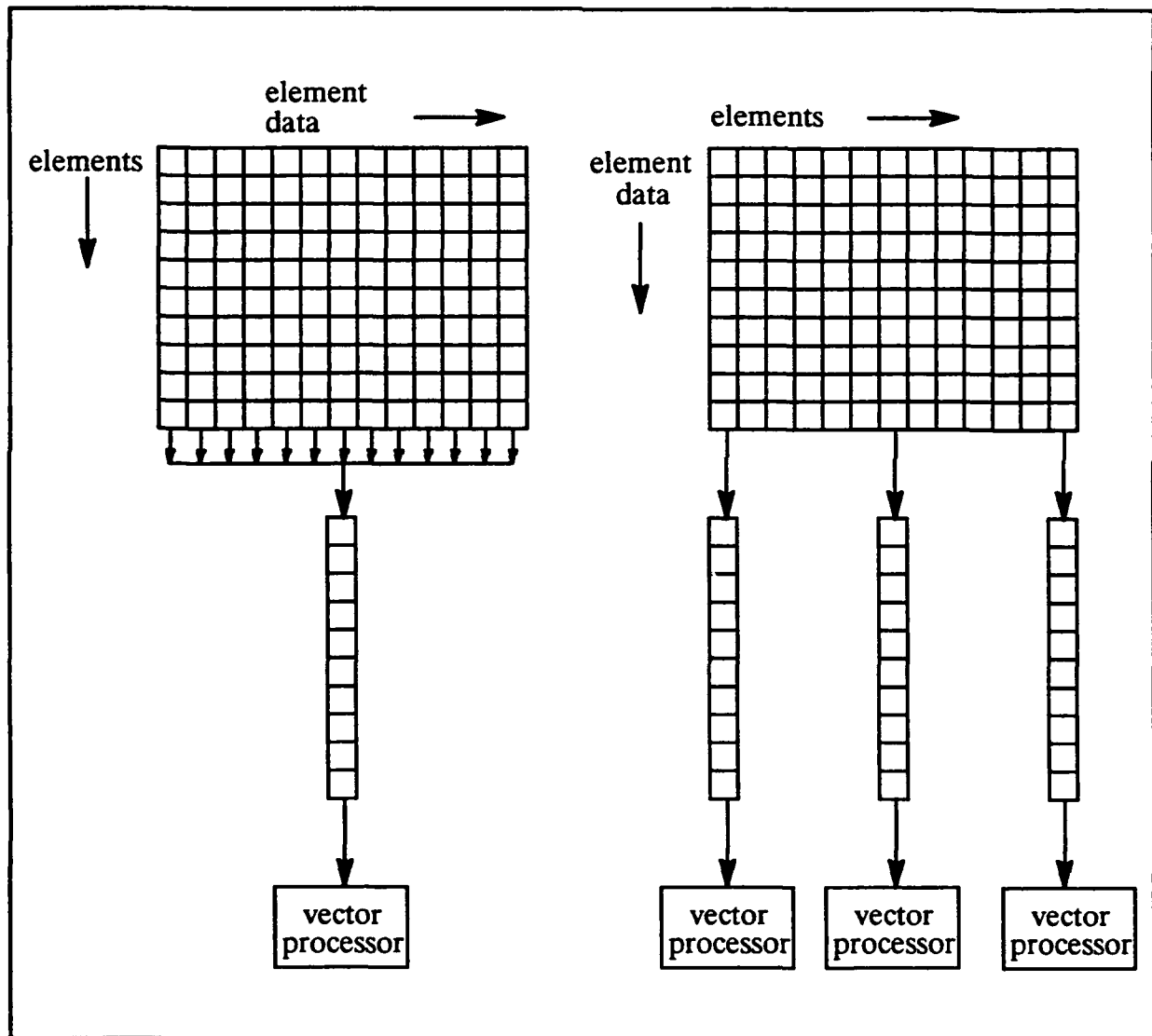


Fig. (2.3-9), elem. block data structures favoring vectorization (l) and concurrency (r)

Element block data structures should be arranged to make parallel and vector calculations both feasible and efficient. Obviously, such calculations and their related gather/scatter functions cannot proceed if data dependencies exist, and just as obviously,

independent data structures for each element in the block are needed to prevent them. Provisions to make parallel and vector calculations more efficient are less obvious. For instance, vector calculations progress most proficiently when the elements of the vectors being processed are offset in memory by a unit stride. This reduces the overhead associated with moving the vectors in and out of the vector registers. Since FORTRAN stores data by columns, if vectorization over the element block is desired then data for different elements should be stored by columns. Conversely, if the element calculations in a block are to be handled concurrently, then the data should be arranged in reverse; data for a given element should be stored by columns. Then, data for various elements can be passed to separate subroutines and treated as an array of dimension one less than in the calling routine. This is highly convenient and expedites vectorization in the subroutines. The above arrangements for element block data structures are represented in Fig. 2.3-9. If both concurrency and vectorization are applied by design to the element block, then the above arrangements need to be combined to work in unison.

2.3.3 POSSIBLE ALGORITHMS

There are numerous ways that concurrency and vectorization can be employed to exploit the three tiered computational hierarchy. Below are discussed seven possible combinations of concurrency and vectorization likely to result in reasonable computational efficiency. FORTRAN code for the computation of the internal force vector employing all of the element computation algorithms discussed below is listed in Appendix B. The internal force vector is a basic example of a level three, or material level, computation that runs the gamut of the element computational hierarchy. Given as well are common blocks manifesting the local block data structures associated with each of the element computation algorithms. Finally, FORTRAN code outlining the various blocking algorithms pertinent to a given element computation algorithm is also provided in Appendix B.

2.3.3.1 BLOCK ELEMENT CONCURRENCY

Block element concurrency (BEC) consists of executing the computations for a block of similar, nonconflicting elements in parallel, one block element per vector processor. Incidental vectorization of the calculations for a given element is performed, with these calculations being arranged so that the largest loops are vectorized, if possible. This often results in vector lengths of twenty-four for eight node bricks and sixty for twenty node bricks. The local block data structures follow the concurrent form of Fig. 2.3-9. BEC represents an algorithm where concurrency is favored over vectorization by virtue of its small grained designed parallelism and moderate incidental vectorizable granularity. It is a relatively

streamlined algorithm to code as the subroutines to be executed in parallel deal with only one element and thus are uncluttered and straightforward to write. It has the advantage that the data required by a vector processor often fits or nearly fits into the cache memory, thereby reducing overhead by minimizing cache misses. Overhead is also reduced by the relatively small amount of bus traffic necessary to supply the vector processors with the required data. One would expect BEC to be most efficient for twenty node isoparametric elements due to the greater incidental vector length.

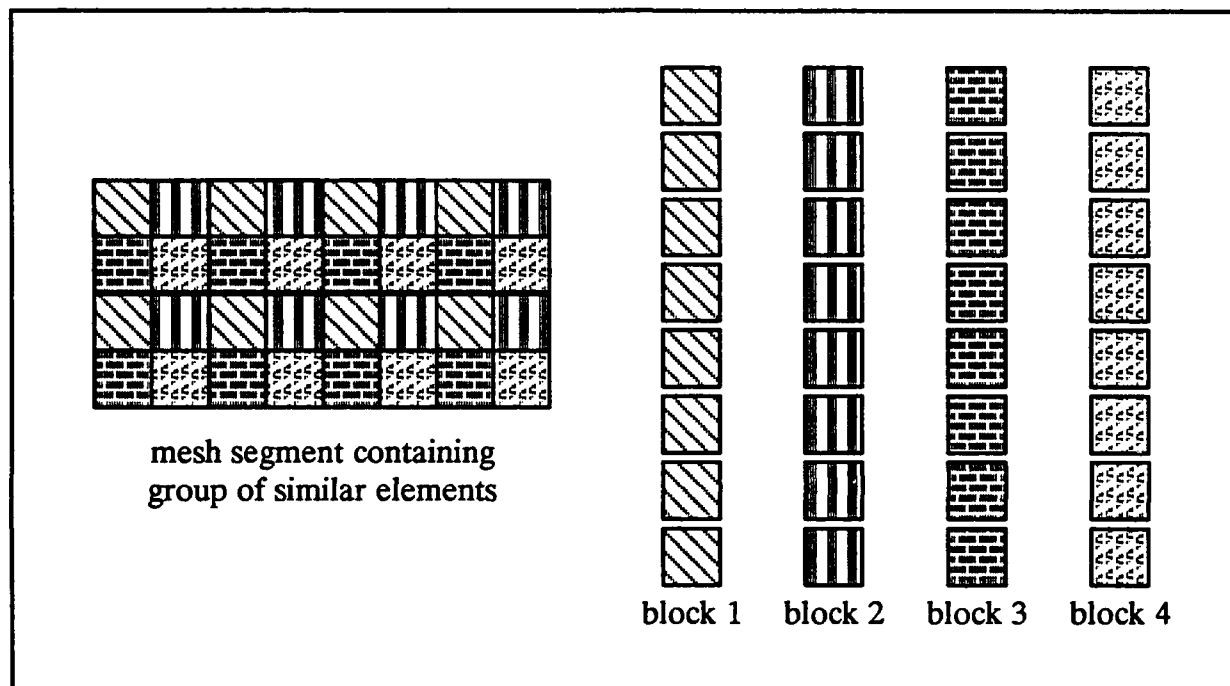


Fig. (2.3-10), element blocking for a group of similar elements

The element blocking algorithm for BEC is adopted from Hughes et al [39]. The structure is first divided into groups of similar elements. Similar elements are of the same type, they employ the same number of material points, and they are uniformly geometrically linear or nonlinear. Similarity is generally necessary to insure that all of the elements in a block can be processed identically. Each group of similar elements is then subdivided into nonconflicting element blocks. This process is illustrated in Fig. 2.3-10, where the group of similar elements in the mesh segment are numbered from left to right and top to bottom. Each element in a group is placed in an open element block if it is nonconflicting with all the elements of the group currently in the block. If the element conflicts with all open blocks, a new block is opened. An element block is closed when it reaches the maximum allowable size, in this case the number of available vector processors, or when all elements in the group are processed. This algorithm generally provides for full or nearly full block sizes for a majority of

the created element blocks. The deleterious effects of insufficient element block sizes decreases as the number of elements in the group increases, and is not that significant in any case as the number of vector processors is usually a relatively small number (8 for the Alliant FX/8, 4 for the Convex C240). As the focus of this research is problems of at least a reasonably large size, inadequate element block sizes are not a major concern and no attempt is made to balance the element block sizes.

2.3.3.2 BLOCK ELEMENT VECTORIZATION

Block element vectorization (BEV) vectorizes element computations over a block of similar, nonconflicting elements. This means that if the result for a given element degree of freedom is required, then that result is found for all block elements in a vector calculation. Incidental concurrency is applied on machines where it is effective, and its application is examined below. The local block data structures follow the vector form of Fig. 2.3-9. BEV represents an algorithm where vectorization is favored over concurrency due to its large block granularity, its designed vectorizable granularity, and its small or nonexistent parallelizable granularity. Coding BEV is more complicated than BEC due to the contrivance of element span loops, the arrangement of the local block data structures, and the manipulation of loops when applying incidental concurrency. BEV also has the drawback that the local block data structures are in general larger than for BEC so that cache misses increase in the absence of concurrency, and incidental concurrency is handicapped by escalated overhead and bus traffic.

When administered, incidental concurrency is arranged to provide for as many concurrent-outer vector-inner (COVI) nested loops as possible. These loops are in general faster than vector-concurrent loops as long as the outer loop is of sufficient size. If data dependencies exist that prohibit COVI execution, then the loops are interchanged so that the element span is the outer concurrent loop if doing so eliminates the data dependencies. This preserves COVI execution and is faster than normal vector-concurrent execution if the vectors of the formerly outer loop are of constant stride and do not use indirect addressing. This interchanging of loops represents a fan-in algorithm. If incidental concurrency is not practised, COVI loops are unrolled if possible and there is no loop interchanging.

The element blocking algorithm for BEV is identical to that for BEC with the maximum allowable block size equal to the size of the vector registers (32 for the Alliant FX/8, 128 for the Convex C240) rather than the number of available vector processors. The size of the vector registers makes the effect of inadequate element block size more critical but

still not significant for sufficiently large problems, so that again no balancing of element block sizes is attempted.

2.3.3.3 CONCURRENT BLOCK ELEMENT VECTORIZATION

Concurrent block element vectorization (CBEV) processes multiple blocks of similar, nonconflicting elements in parallel, and vectorizes over the element blocks themselves. For the calculations pertaining to a single element block on a vector processor, outer loops are unrolled as much as is feasible, such as in BEV with no incidental concurrency. The local block data structures are a merger of the concurrent and vector forms of Fig 2.3-9. The first dimension refers to the element block span and the final dimension refers to the vector processor number, as illustrated in Fig. 2.3-11. CBEV is an algorithm that generally favors vectorization as a result of its large parallelizable granularity and significant designed vectorizable granularity, although the element blocking algorithm may enforce primarily either concurrency or vectorization. CBEV is actually less complicated to code than BEV with incidental concurrency because the concurrency is accounted for early on and then the programming reduces to BEV without incidental concurrency. CBEV operates under the burden of a very large local block data structure that places heavy traffic on the bus and aggravates the cache miss problem.

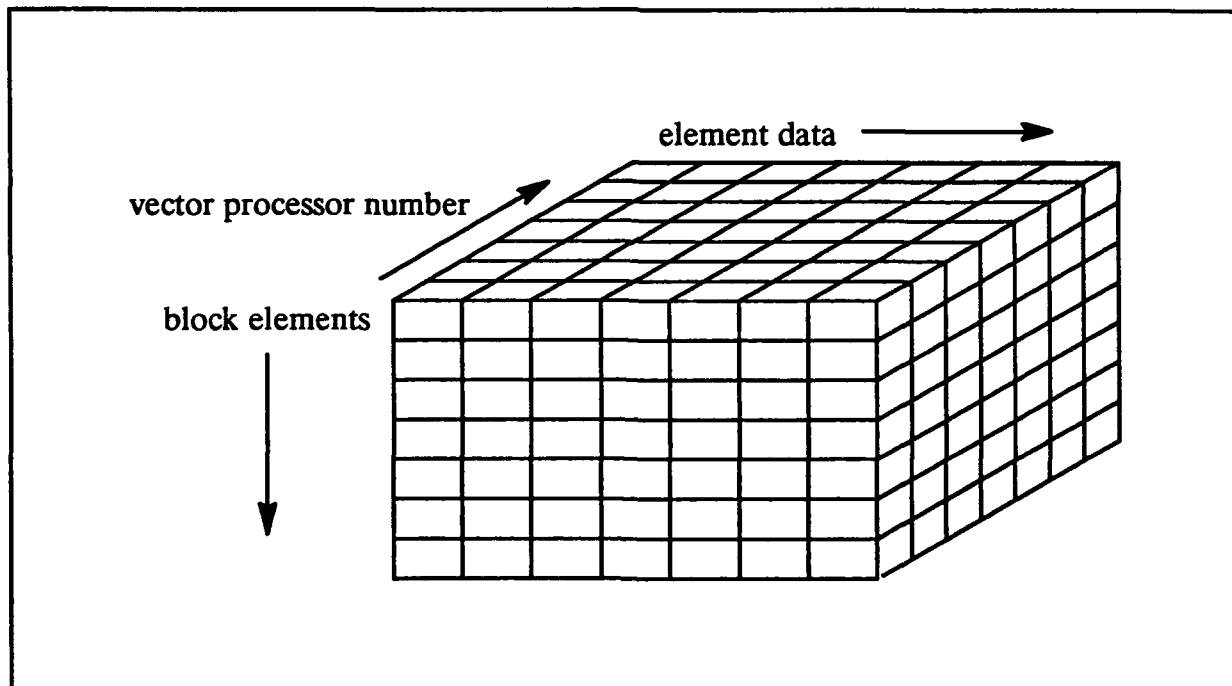


Fig. (2.3-11), element block data structures for CBEV

The element blocking algorithm for CBEV is more involved than for BEC and BEV, requiring an extra step. First, the same algorithm as for BEC and BEV is applied with the maximum allowable block size equal to the size of the vector registers multiplied by the number of available vector processors (8x32 for the Alliant FX/8, 4x128 for the Convex C240). These large element blocks must then be subdivided into smaller blocks for parallel processing. Two approaches can be adopted, one favoring vectorization, the other concurrency. To favor vectorization, the large element block is divided by the size of the vector registers with the remainder placed in the final smaller element block. If the final block is too small, the denominator can be reduced from the initial size of the vector registers so that the smaller element block sizes are more balanced. This approach can often leave processors unused. To favor concurrency, the large element block is divided by the number of available vector processors. The remainder is distributed among the smaller element blocks. With this scheme, all the processors are used, which is ostensibly more efficient than leaving any processors idle. Both of these approaches are implemented to determine if one is significantly more efficient than the other. Of course, if the structure is large enough so that the great majority of the large element blocks are full, then there is scant if any difference between the two blocking schemes.

2.3.3.4 BEC WITH MATERIAL POINT VECTORIZATION

Block element concurrency with material point vectorization (BEC/MPV) is identical to BEC except that incidental vectorization is replaced with vectorization over the material points of an element. For computations which encompass only the element level of the computational hierarchy BEC/MPV reduces to BEC. The local block data structures are similar to those of CBEV with the element block span being replaced by the span of material points. The idea behind BEC/MPV is to formalize the vectorization by providing a consistent designed vector length. The use of the material points to furnish this vector length is a natural choice compatible with the element computational hierarchy. For problems utilizing twenty node isoparametric elements with full integration, the vector length is an acceptable value of twenty-seven. However, if reduced integration or eight node isoparametric elements are used the vector length can be eight or less, which is clearly unacceptable and a severe drawback.

The element blocking algorithm for BEC/MPV is identical to that for BEC.

2.3.3.5 BEV WITH MATERIAL POINT CONCURRENCY

Block element vectorization with material point concurrency (BEV/MPC) differs from BEV in that incidental concurrency is supplanted by the parallel execution of the vectorized block element calculations specific to the material points, one material point per

vector processor. Again, computations occurring only at the element level of the computational hierarchy devolve to BEV. The local block data structures are again similar to CBEV, this time with the vector processor number being replaced by the material point number. The concept of BEV/MPC is to exploit the material point level of the computational hierarchy with designed concurrency. BEV/MPC does not suffer the same deficiency as BEC/MPV since even if the number of material points is eight or slightly less a satisfactory parallelizable granularity can be obtained. However, the amount of data assigned to each processor is much larger than for BEV and bus traffic and cache misses increase. Also, the number of material points will not necessarily be an even multiple of the number of vector processors, possibly increasing overhead. Lastly, material point quantities cannot be gathered or scattered in parallel due to data conflicts, necessitating a nonoptimized outer material point loop in the element block scatter that would otherwise be buried in the vectorized element block calculations.

The element blocking algorithm for BEV/MPC is identical to that for BEV.

2.3.3.6 BLOCK MATERIAL POINT CONCURRENCY

Block material point concurrency (BMPC) performs the computations for a block of similar, nonconflicting material points in parallel, one material point per vector processor. The local block data structures follow the concurrent form of Fig. 2.3-9 and are very similar to those for BEC except that storage required for the material points of an element in BEC is not needed in BMPC. BMPC grew out of the observation that for a machine like the Alliant FX/8 favoring concurrency over vectorization as in BEC leads to greater efficiency. BMPC has a higher degree of parallelism than BEC, with a smaller parallelizable granularity and an equivalent vectorizable granularity. BMPC also has a smaller local block data structure than BEC, which causes fewer cache misses. The drawbacks of BMPC are threefold. First, the greater degree of parallelism increases the associated overhead. Second, for a block of material points to be nonconflicting, they must refer to separate, nonconflicting elements. If the element quantity being computed has to be manipulated in its complete form, such as with the tangent mass, then a hybrid approach combining BMPC and BEC must be employed. Lastly, BMPC is not applicable to purely element level calculations and BEC must be used instead, further obscuring the difference between BMPC and BEC.

BMPC uses the same element blocking algorithm as for BEC, with material point blocks being derived from element blocks by first considering material point one for all block elements, then material point two, and so forth until all the material points are processed.

2.3.3.7 BLOCK MATERIAL POINT VECTORIZATION

Block material point vectorization (BMPV) vectorizes element computations over blocks of similar, nonconflicting material points. It is mentioned as the complement of BMPC for approaches favoring vectorization. In practise, it is virtually identical to BEV and is not considered.

2.3.4 IMPLEMENTATION

BEC, BEV, and CBEV are fully implemented where practical and in the most efficient form on the Alliant FX/8 and the Convex C240. The material point algorithms – BEC/MPV, BEV/MPC, and BMPC – are not fully implemented for the following reasons. First, it is a basic tenet of finite element analysis to minimize the number of material points used in order to reduce the overall computational effort. Although a one point integration scheme for eight node bricks and the corresponding scheme for twenty node bricks is not adopted in NLD FEP, it certainly could be, and such a scheme would demolish the efficiency of the material point algorithms. As it is, even an eight point integration sharply restricts BEC/MPV. Second, all the material point algorithms reduce to BEC or BEV for element level computations and are not applicable in many situations. Finally, there was not enough time to institute all of the above algorithms, and it was decided to install only the basic algorithms plus a sequential algorithm for timing comparisons.

2.4 MATERIAL MODEL

Material nonlinearities in NLD FEP exist in both a small strain form, where the effects of geometric nonlinearities are ignored, and a finite strain form, where those effects are included. The small strain material model is based on the rate independent incremental (flow) theory of plasticity with isotropic and kinematic hardening and an isotropic Von Mises yield surface. The uniaxial response of the material is assumed to be bilinear. The finite strain model is an extension of the small strain model so that finite strains and rotations are included. This model is intended for the analysis of ductile metals which experience large plastic strains and comparatively small elastic strains, so that the unloaded state is virtually identical to the undeformed. This condition allows the use of an additive rather than a multiplicative decomposition of the elastic and plastic components of the strain increments referred to the deformed configuration. In the finite strain models, a hypoelastic approach is adopted where an objective measure of the rate of Cauchy stress is coupled with a constitutive law defined in terms of the Cauchy stress and the rate of deformation tensor. The objective

rates of Cauchy stress employed in NLD FEP are the Truesdell and Green–Naghdi rates. The Green–Naghdi rate is implemented through an unrotated formulation that requires a polar decomposition of the deformation gradient. Both the unrotated and Truesdell formulations represent a unique approach to finite strain plasticity in implicit computational mechanics.

In the following discussion, the theory underlying the small strain and finite strain plasticity models is presented and the application of these models is developed and outlined.

2.4.1 DEFINITIONS OF STRESS AND STRAIN FOR FINITE DEFORMATION

Let \mathbf{X} identify a material point in the reference (undeformed) configuration \mathbf{B}_0 and \mathbf{x} denote the position vector of the same material point in the deformed (current) configuration \mathbf{B} . The deformation gradient \mathbf{F} of the motion from \mathbf{B}_0 to \mathbf{B} is given as

$$\mathbf{F} = \partial \mathbf{x} / \partial \mathbf{X}, \quad \det(\mathbf{F}) = |\mathbf{F}| > 0 \quad (2.4-1)$$

The polar decomposition of \mathbf{F} generates

$$\mathbf{F} = \mathbf{V}\mathbf{R} = \mathbf{R}\mathbf{U} \quad (2.4-2)$$

where \mathbf{V} and \mathbf{U} are the symmetric, positive definite left and right stretch tensors, respectively, and \mathbf{R} is the proper orthogonal rotation tensor. The principal values of \mathbf{V} and \mathbf{U} are the stretch ratios of the deformation; their natural logarithms are sometimes referred to as the natural strains. The two forms of equation (2) that comprise alternate methods of separating the motion of a material point are demonstrated in Fig. 2.4–1. In all configurations, spatial orthogonal axes which refer to fixed, global Cartesian coordinates are defined. Attached to each material point exist material or convected axes, etched in the body and originally aligned with the global spatial axes in \mathbf{B}_0 , which follow both the deformation and rotation of the material. In addition, an orthogonal reference frame fastened to individual material points such that throughout the loading history the motion relative to these axes is only deformation may be identified. Given the $\mathbf{R}\mathbf{U}$ decomposition, for instance, these axes are spatial during the motion from \mathbf{B}_0 to \mathbf{B}_u ; they are not affected by the deformation of the body. From \mathbf{B}_u to \mathbf{B} they are material and rotate with the body. These axes are termed the unrotated axes; stress and strain tensors and their rates associated with this frame are said to be defined in the unrotated configuration.

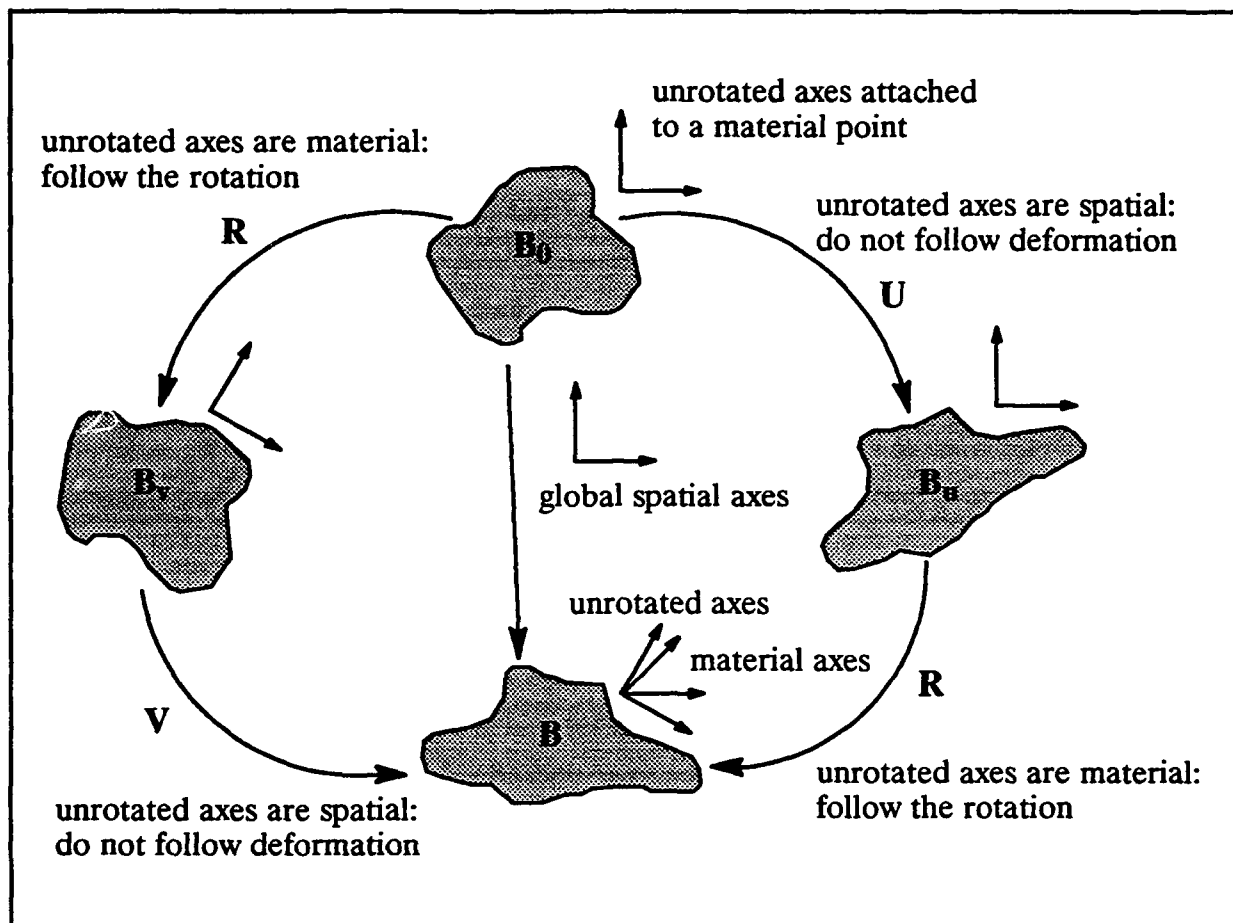


Fig. (2.4-1), configurations characteristic of two methods of decomposition

The symmetric, positive definite right Cauchy-Green or metric tensor is defined as

$$\mathbf{C} = \mathbf{F}^T \mathbf{F} = \mathbf{U}^2 \quad (2.4-3)$$

with which the Lagrange or Green strain tensor can be calculated as

$$\mathbf{E} = \frac{1}{2}(\mathbf{C} - \mathbf{I}) \quad (2.4-4)$$

where \mathbf{I} is the identity tensor. The material velocity vector is given as $\mathbf{v} = \dot{\mathbf{x}}$; the velocity gradient is then given as

$$\mathbf{L} = \frac{\partial \mathbf{v}}{\partial \mathbf{x}} = \frac{\partial \mathbf{v}}{\partial \mathbf{X}} \frac{\partial \mathbf{X}}{\partial \mathbf{x}} = \dot{\mathbf{F}} \mathbf{F}^{-1} \quad (2.4-5)$$

The symmetric part of \mathbf{L} is called the rate of deformation tensor \mathbf{D} ; the antisymmetric part is the rate of spin tensor \mathbf{W} . Therefore,

$$\mathbf{L} = \mathbf{D} + \mathbf{W} \quad (2.4-6)$$

where

$$\mathbf{D} = \frac{1}{2}(\mathbf{L} + \mathbf{L}^T); \quad \mathbf{W} = \frac{1}{2}(\mathbf{L} - \mathbf{L}^T) \quad (2.4-7)$$

\mathbf{W} represents the angular velocity of the principal axes of the rate of deformation tensor \mathbf{D} . When integrated over the loading history, the principal values of \mathbf{D} are recognized as the logarithmic (true) strains of infinitesimal fibers oriented in the principal directions if the principal directions do not rotate. \mathbf{D} and \mathbf{W} are instantaneous rates that refer to the global spatial axes at time t ; in general they do not pertain to a particular point during the motion of the body and have no awareness of the history of the deformation.

The \mathbf{RU} decomposition of \mathbf{F} can be employed to rewrite (6) yielding

$$\mathbf{L} = \dot{\mathbf{R}}\mathbf{R}^T + \mathbf{R}\dot{\mathbf{U}}\mathbf{U}^{-1}\mathbf{R}^T \quad (2.4-8)$$

The first term of the right-hand side of (8) is the antisymmetric tensor $\mathbf{\Omega}$:

$$\mathbf{\Omega} = \dot{\mathbf{R}}\mathbf{R}^T \quad (2.4-9)$$

$\mathbf{\Omega}$ represents the rate of rigid body rotation at a material point, or the rate of rotation of the unrotated axes. The symmetric part of the quantity $\dot{\mathbf{U}}\mathbf{U}^{-1}$ is the unrotated rate of deformation tensor \mathbf{d} :

$$\mathbf{d} = \frac{1}{2}(\dot{\mathbf{U}}\mathbf{U}^{-1} + \mathbf{U}^{-1}\dot{\mathbf{U}}) = \mathbf{R}^T\mathbf{D}\mathbf{R} \quad (2.4-10)$$

\mathbf{d} measures a material strain rate relative to the unrotated axes. The rate of Green strain can now be written in the interchangeable forms

$$\dot{\mathbf{E}} = \mathbf{F}^T\mathbf{D}\mathbf{F} \quad (2.4-11)$$

and

$$\dot{\mathbf{E}} = \mathbf{U}\mathbf{d}\mathbf{U} \quad (2.4-12)$$

The Cauchy stress tensor \mathbf{T} may be expressed as either

$$\mathbf{T} = \frac{1}{|\mathbf{F}|}\mathbf{F}\mathbf{S}\mathbf{F}^T \quad (2.4-13)$$

or

$$\mathbf{T} = \mathbf{R}\mathbf{t}\mathbf{R}^T \quad (2.4-14)$$

where \mathbf{S} is the symmetric Piola-Kirchoff stress tensor and \mathbf{t} is the unrotated Cauchy stress tensor. \mathbf{t} is the Cauchy stress referenced to the unrotated axes; it is the Cauchy stress with the effects of material rotation eliminated.

2.4.2 STRESS RATES AND CONSTITUTIVE RELATIONSHIPS

The hypoelastic constitutive law in the finite strain plasticity models is based on the Cauchy stress and the rate of deformation tensor. The Cauchy stress is chosen because it is a tangible and physically meaningful measure of stress referred to the deformed configuration. The use of the rate of deformation tensor follows because it is the measure of strain rate that is work conjugate to the Cauchy stress in the sense that the work per unit volume in the deformed configuration is given by $T_{ij}D_{ij}$. This leads to a constitutive framework of the form

$$\hat{\mathbf{T}} = f(\mathbf{T}, \mathbf{D}) \quad (2.4-15)$$

where $\hat{\mathbf{T}}$ is an objective rate of Cauchy stress. By objective it is meant that a given second rank tensor is either unchanged or unchanged apart from orientation under a superposed rigid body motion [44]. This prevents a simple rotation from altering the principal values and invariants of the tensor. Consider the motion

$$\mathbf{x}^*(t) = \mathbf{a}(t) + \mathbf{Q}(t)\mathbf{x}(t) \quad (2.4-16)$$

where \mathbf{x} is the given motion of the body, \mathbf{a} expresses a rigid body translation, and \mathbf{Q} is a proper orthogonal tensor depicting a rigid body rotation of the deformed state. From (16) it is a simple matter to show that

$$\mathbf{F}^* = \mathbf{Q}\mathbf{F} \quad (2.4-17)$$

$$\mathbf{R}^* = \mathbf{Q}\mathbf{R} \quad (2.4-18)$$

$$\mathbf{L}^* = \dot{\mathbf{Q}}\mathbf{Q}^T + \mathbf{Q}\mathbf{L}\mathbf{Q}^T \quad (2.4-19)$$

$$\mathbf{D}^* = \mathbf{Q}\mathbf{D}\mathbf{Q}^T \quad (2.4-20)$$

$$\dot{\mathbf{W}} = \dot{\mathbf{Q}}\mathbf{Q}^T + \mathbf{Q}\dot{\mathbf{W}}\mathbf{Q}^T \quad (2.4-21)$$

$$\dot{\mathbf{d}} = \dot{\mathbf{d}} \quad (2.4-22)$$

$$\dot{\mathbf{T}} = \dot{\mathbf{Q}}\mathbf{T}\mathbf{Q}^T \quad (2.4-23)$$

$$\dot{\mathbf{S}} = \dot{\mathbf{S}} \quad (2.4-24)$$

$$\dot{\mathbf{t}} = \dot{\mathbf{t}} \quad (2.4-25)$$

$$\dot{\mathbf{T}} = \dot{\mathbf{Q}}\mathbf{T}\mathbf{Q}^T + \mathbf{Q}\dot{\mathbf{T}}\mathbf{Q}^T + \mathbf{Q}\mathbf{T}\dot{\mathbf{Q}}^T \quad (2.4-26)$$

$$\dot{\mathbf{S}} = \dot{\mathbf{S}} \quad (2.4-27)$$

$$\dot{\mathbf{t}} = \dot{\mathbf{t}} \quad (2.4-28)$$

It is seen that the rate of deformation tensor undergoes a similarity transformation so that it retains its eigenvalues but not its eigenvectors. It is thus unchanged apart from orientation and objective. Unfortunately, the rate of Cauchy stress is not objective and alternate forms of the rate of Cauchy stress must be considered.

When choosing an objective rate of Cauchy stress to satisfy (15), two principal factors should be considered. First, the use of the stress rate should lead to a convenient formulation. Second, the results obtained should be physically plausible. Accordingly, the first stress rate implemented in NLD FEP is the Truesdell stress rate. Differentiating equation (13) with respect to time one obtains

$$\hat{\mathbf{T}}_T = \dot{\mathbf{T}} - \mathbf{L}\mathbf{T} - \mathbf{T}\mathbf{L}^T + tr(\mathbf{D})\mathbf{T} \quad (2.4-29)$$

or

$$\hat{\mathbf{T}}_T = \frac{1}{|F|} \dot{\mathbf{F}}\mathbf{S}\mathbf{F}^T \quad (2.4-30)$$

where $\hat{\mathbf{T}}_T$ is the Truesdell stress rate. Observing (30), the Truesdell stress rate can be described as the rate of change of the symmetric Piola-Kirchoff stress expressed in the deformed configuration. That is, the change in symmetric Piola-Kirchoff stress with time is measured

with respect to the deformed configuration acting as the reference configuration. By combining equations (17), (27), and (29) one can write

$$\hat{\mathbf{T}}_T^* = \mathbf{Q}\hat{\mathbf{T}}_T\mathbf{Q}^T \quad (2.4-31)$$

Thus, the Truesdell stress rate is objective.

The principal advantage afforded by the use of the Truesdell stress rate is the straightforward transferral of the constitutive relationship to the reference configuration, which is necessary in a Total Lagrangian formulation. The principal disadvantage is that the invariants of \mathbf{T} are not necessarily stationary for vanishing $\hat{\mathbf{T}}_T$ [44]. This property is particularly disturbing in light of the fact that a zero governing stress rate implies that there is no movement along the yield surface when the material is undergoing plastic flow. However, if the invariants of \mathbf{T} vary for a zero value of $\hat{\mathbf{T}}_T$, the yield surface might actually move away from the stress point, initiating spurious plastic flow. A second disadvantage associated with the Truesdell stress rate is that it is a relatively complicated process to incorporate small strain plasticity into the finite strain procedure, and the increments of stress involved in the recovery process are not physically meaningful.

The second stress rate installed in NLD FEP is the Green-Naghdi stress rate. This time differentiating equation (14) with time, applying equation (9), and rearranging gives

$$\hat{\mathbf{T}}_G = \dot{\mathbf{T}} - \boldsymbol{\Omega}\mathbf{T} + \mathbf{T}\boldsymbol{\Omega} \quad (2.4-32)$$

or

$$\hat{\mathbf{T}}_G = \mathbf{R}\dot{\mathbf{t}}\mathbf{R}^T \quad (2.4-33)$$

here $\hat{\mathbf{T}}_G$ is the Green-Naghdi stress rate. From equation (33) one can state that the Green-Naghdi stress rate is the rate of change of the unrotated stress referred to the deformed configuration. That is, the Green-Naghdi stress rate measures the rate of Cauchy stress with the change in stress due to the material rotation removed. Applying equations (18), (28), and (32), one arrives at

$$\hat{\mathbf{T}}_G^* = \mathbf{Q}\hat{\mathbf{T}}_G\mathbf{Q}^T \quad (2.4-34)$$

proving that the Green-Naghdi stress rate is objective.

Dienes [20] shows that by application of the Cayley-Hamilton theorem one can express the constitutive relation

$$\hat{\mathbf{T}}_G = f(\mathbf{T}, \mathbf{D}) \quad (2.4-35)$$

in the equivalent form of

$$\dot{\mathbf{t}} = f(\mathbf{t}, \mathbf{d}) \quad (2.4-36)$$

equations (22) and (28) show that \mathbf{d} and $\dot{\mathbf{t}}$ are objective rates. The formulation of equation (36) is desirable because it leads to a simple and elegant fusion of small and finite strain plasticity. By performing the stress recovery in the unrotated configuration, the small and finite strain plasticity models are essentially divorced; they can function independently. Any existing small strain plasticity model could become the kernel on which the finite strain model operates. For a Von Mises approach, stress and strain rate at the beginning of the recovery process are transformed into the unrotated configuration, the small strain plasticity model is applied, and the results are transformed back to the reference configuration. Tensorial state variables such as the back stress necessary for the evolution of the material response need only be defined in the unrotated configuration and are not transformed. Equation (36) represents the description of the Green-Naghdi stress rate instituted in NLDFEP; henceforth it will be identified as the unrotated Cauchy stress rate.

Use of the unrotated Cauchy stress rate has other advantages as well. First, the invariants of the Cauchy stress remain constant when $\dot{\mathbf{t}}$ disappears [44]. Second, the unrotated Cauchy stress is physically meaningful; during the stress recovery relevant stress increments are manipulated. The drawback connected with the unrotated Cauchy stress rate is the necessity of a polar decomposition of the deformation gradient. The next subsection will show that this is not a significant problem for implicit applications.

A third stress rate was considered for application in NLDFEP. This is the Jaumann stress rate and is characterized by the equation

$$\hat{\mathbf{T}}_J = \dot{\mathbf{T}} - \mathbf{W}\mathbf{T} + \mathbf{T}\mathbf{W} \quad (2.4-37)$$

The Jaumann stress rate can be derived from the Truesdell stress rate by setting \mathbf{D} in equation (29) equal to zero. Equations (21), (23), (26), and (37) merge to yield

$$\hat{\mathbf{T}}_J^* = \mathbf{Q}\hat{\mathbf{T}}_J\mathbf{Q}^T \quad (2.4-38)$$

illustrating the objectivity of the Jaumann stress rate.

Equation (37) is similar to equation (32), with Ω being replaced with W . Consequently, the Jaumann stress rate is the rate of Cauchy stress with the change in stress related to the rotation of the principal axes of D removed. Defining the antisymmetric tensor Z as

$$Z = DV - VD \quad (2.4-39)$$

and the angular velocity vectors ω , w , and z as

$$\Omega_{ik} = \epsilon_{ijk}\omega_j \quad (2.4-40)$$

$$W_{ik} = \epsilon_{ijk}w_j \quad (2.4-41)$$

$$Z_{ik} = \epsilon_{ijk}z_j \quad (2.4-42)$$

where ϵ_{ijk} is the permutation tensor, Dienes [20] derives the following relationship between Ω and W through the equation

$$\omega = w + [I \operatorname{tr}(V) - V]^{-1} z \quad (2.4-43)$$

Except for special cases, such as rigid body rotation where $D = 0$ and pure dilation where $V = \alpha I$, Ω and W differ for general motions.

The Jaumann stress rate has been widely applied in finite element programs due to its accessibility as a consequence of calculating D . Unfortunately, for some deformations the Jaumann stress rate projects a physically implausible result for the material at truly finite strains, as described in Section 1.2 for the finite simple shear problem. This case with a linear elastic, isotropic constitutive matrix relating \hat{T}_J and D is examined in Section 3.5. Given this, and the fact that the unrotated stress rate accounts for the actual material rotation while the Jaumann stress rate only approximates it, it was decided to implement only the unrotated stress rate.

2.4.3 PERFORMING THE POLAR DECOMPOSITION

As stated in the subsection above, a polar decomposition of the deformation gradient is required for implementation of the unrotated stress rate. The decomposition can be accomplished exactly, involving the solution of a 3x3 eigenvalue problem for each material point, or it can be approximated employing an algorithm based on equation (43). In this subsection, both of these approaches will be presented and numerical results compared to determine which is the most efficient.

2.4.3.1 THE EXACT APPROACH

The algorithm presented here is based directly on work done by Hoger and Carlson [37]. The algorithm consists of the following steps at each material point:

- 1) Compute the components of the metric tensor C given by $C_{ij} = F_{ki}F_{kj}$ in accordance with equation (3).
- 2) Compute the components of C^2 given by $C_{ij}^2 = C_{ik}C_{kj}$.
- 3) Compute the principal values of C by solving a 3x3 eigenvalue problem.
- 4) Compute λ_i , the principal values of U , as the square roots of the principal values of C .
- 5) Compute the invariants of U from the principal values:

$$I_u = \lambda_1 + \lambda_2 + \lambda_3$$

$$II_u = \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_1\lambda_3 \quad (2.4-44)$$

$$III_u = \lambda_1\lambda_2\lambda_3$$

- 6) Compute the components of U based on the following:

$$a = 1 / (I_u II_u - III_u)$$

$$b = I_u III_u \quad (2.4-45)$$

$$c = I_u^2 - II_u$$

$$U_{11} = a (b + cC_{11} - C_{11}^2)$$

$$U_{12} = U_{21} = a (cC_{12} - C_{12}^2)$$

$$U_{22} = a (b + cC_{22} - C_{22}^2)$$

$$U_{13} = U_{31} = a (cC_{13} - C_{13}^2)$$

$$U_{23} = U_{32} = a (cC_{23} - C_{23}^2)$$

$$U_{33} = a (b + cC_{33} - C_{33}^2) \quad (2.4-46)$$

7) Compute the components of U^{-1} based on the following:

$$\begin{aligned}
 a &= 1 / [III_u (I_u II_u - III_u)] \\
 b &= I_u II_u^2 - III_u (I_u^2 + II_u) \\
 c &= -III_u - I_u (I_u^2 - 2II_u) \\
 d &= I_u
 \end{aligned}
 \tag{2.4-47}$$

$$\begin{aligned}
 U_{11}^{-1} &= a (b + cC_{11} + dC_{11}^2) \\
 U_{12}^{-1} &= U_{21}^{-1} = a (cC_{12} + dC_{12}^2) \\
 U_{22}^{-1} &= a (b + cC_{22} + dC_{22}^2) \\
 U_{13}^{-1} &= U_{31}^{-1} = a (cC_{13} + dC_{13}^2) \\
 U_{23}^{-1} &= U_{32}^{-1} = a (cC_{23} + dC_{23}^2) \\
 U_{33}^{-1} &= a (b + cC_{33} + dC_{33}^2)
 \end{aligned}
 \tag{2.4-48}$$

8) Compute the components of the rotation tensor R given by $R_{ij} = F_{ik} U_{kj}^{-1}$ in accordance with equation (2).

In practice, the tensors U , U^{-1} , C , and C^2 are stored and manipulated in upper triangular vector form consisting of 6 entries in the order of $\{ 11, 12, 22, 13, 23, 33 \}$. The major sticking point in the algorithm is step 3. Hoger and Carlson [37] proposed a closed form solution to the quartic equation relating the invariants of U to the invariants of C given by

$$I_u^4 - 2I_c I_u^2 - 8\sqrt{III_c} I_u + (I_c^2 - 4II_c) = 0
 \tag{2.4-49}$$

but later retracted it based on work by Sawyers [74]. Although it may yet be possible to achieve a closed form solution to equation (49), that solution will involve evaluating numerous square root floating point operations. By contrast, step 3 is accomplished by a very efficient eigenvalue routine specifically tailored to the 3×3 size and employing Jacobi transformations. All "do loops" are unrolled with specific rather than symbolic references to subscripted arrays. Generally, less than five iterations are necessary to reduce C to diagonal form. It is likely that solving the eigenvalue problem would be faster than a closed form solution if one existed.

Timing studies of the above algorithm with only scalar optimization were made on an Alliant FX/8, a Convex 240, and an Apollo DN10000 computer. No more than 0.0004 CPU seconds were required to perform all the steps for a material point on any of the computers. Given a structure with 100,000 material points, it would take about half a minute to all the polar decompositions. This should be trivial compared to the time required to compute and factorize the structural stiffness matrix necessary for implicit applications. On the Alliant and the Convex vector and parallel optimizations within the context of the element computation algorithms of Section 2.3 should further reduce this time significantly.

2.4.3.2 THE APPROXIMATE APPROACH

The algorithm to approximate the polar decomposition was developed by Flanagan and Taylor [26]. It is based on application of equations (39)–(43) and computes the left rather than the right stretch tensor as well as the rotation tensor. The forward integration dictated by equation (9) to obtain $\mathbf{R}_{t+\Delta t}$ is performed in a manner designed to maintain the orthogonality of \mathbf{R} following the work of Hughes and Winget [40]. The algorithm proceeds as follows for each material point:

- 1) Compute \mathbf{D} and \mathbf{W} based on equations (5)–(7).
- 2) Compute \mathbf{z} , ω , and Ω based on equations (39)–(43.)
- 3) Solve for $\mathbf{R}_{t+\Delta t}$ from

$$\left(\mathbf{I} - \frac{1}{2} \Delta t \Omega \right) \mathbf{R}_{t+\Delta t} = \left(\mathbf{I} + \frac{1}{2} \Delta t \Omega \right) \mathbf{R}_t \quad (2.4-50)$$

- 4) Calculate

$$\dot{\mathbf{V}} = (\mathbf{D} + \mathbf{W})\mathbf{V} - \mathbf{V}\Omega \quad (2.4-51)$$

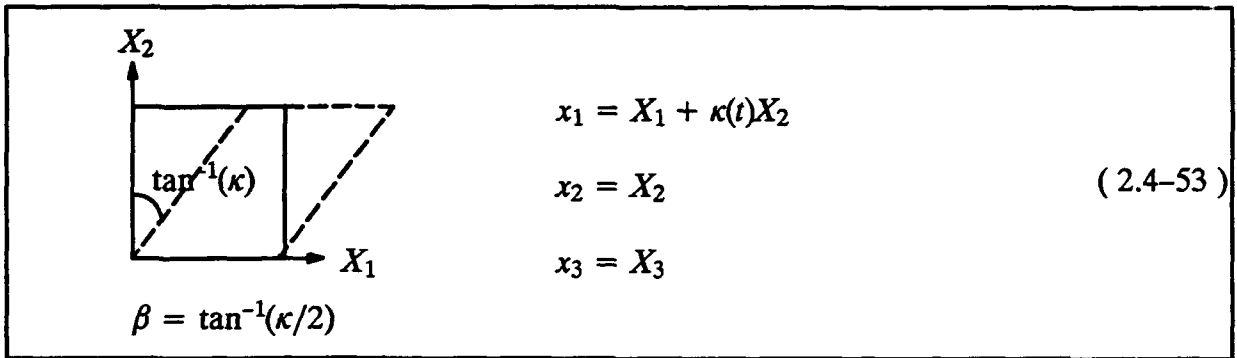
- 5) Calculate

$$\mathbf{V}_{t+\Delta t} = \mathbf{V}_t + \Delta t \dot{\mathbf{V}}_{\Delta t} \quad (2.4-52)$$

In general, instead of computing rate tensors such as \mathbf{D} in their rate form one calculates $\Delta \mathbf{D} = \Delta t \mathbf{D}_{n+1/2}$. This results from midpoint evaluation of \mathbf{L} using equation (5), which will be discussed in further detail later on.

2.4.3.3 NUMERICAL COMPARISON

Consider the finite deformation simple shear problem characterized by the equations



The rotation tensor is given as

$$[R] = \begin{bmatrix} \cos(\beta) & \sin(\beta) & 0 \\ -\sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.4-54)$$

A computer program was written to compute the rotation tensor by both of the algorithms outlined above and to compare the results to equation (54). Increments of κ were taken to be 0.0001 for 30 time steps and 0.01 for 970 time steps to mirror the analysis of the finite deformation simple shear problem presented later on. The shearing of the material was performed at a constant rate. The outcome of the study is shown in Fig. 2.4-2. The norm of the relative error of the exact computation increases slightly during the study due to accumulated round off error, but never exceeds 10^{-12} . The relative error of the approximate computation, however, follows a far different path. At low levels of deformation, the relative error is acceptable, but it rapidly deteriorates when the larger increments of κ commence. As κ reaches 1.0, the relative error nears the clearly unacceptable level of 0.1 before leveling off at around 2.8. This illustrates the pitfalls of the approximate method. Very small time steps must be administered if the accuracy of the approximation is to be maintained. Flanagan and Taylor [26] applied the approximate method to explicit dynamics where the criterion for stability of the numerical integration demands sufficiently small time steps to preserve the accuracy of the approximation. For implicit dynamics, the constraint of realistic time steps excludes the approximate polar decomposition method from consideration.

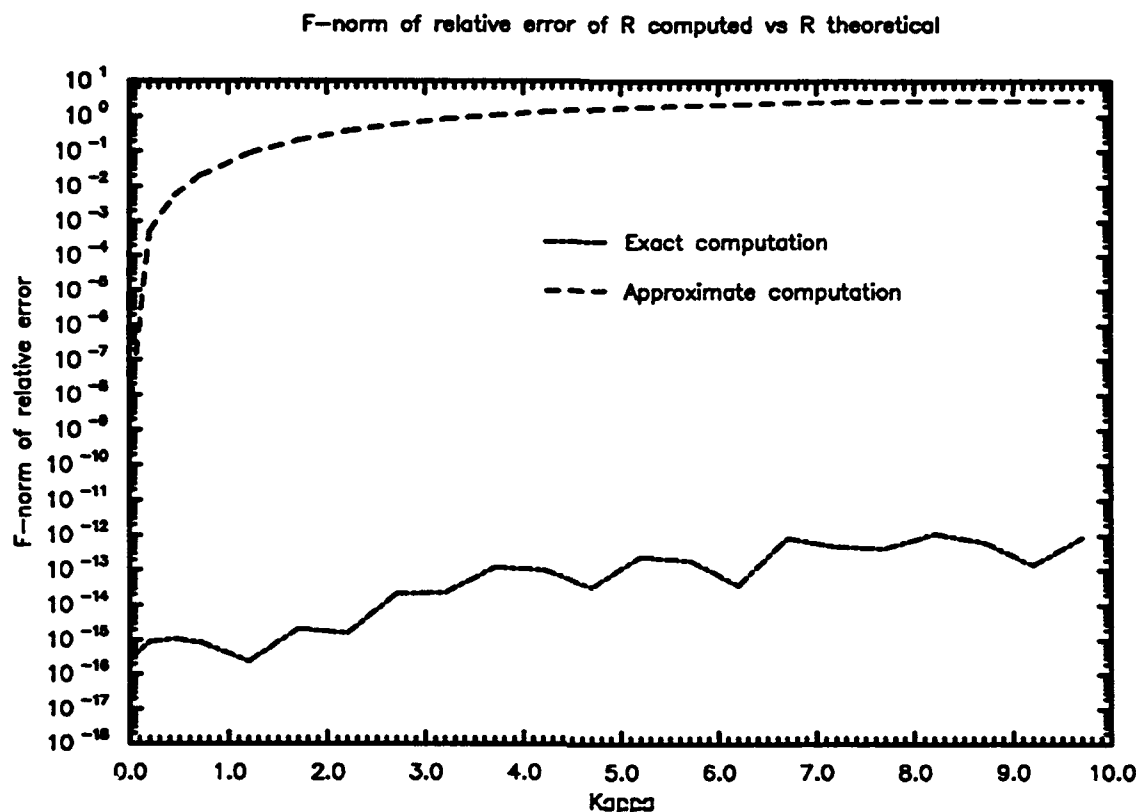


Fig. (2.4-2), comparison of exact and approximate methods of polar decomposition

2.4.4 SMALL STRAIN PLASTICITY ALGORITHM

During a time step from state n to state $n + 1$, global equilibrium iterations, designated by i , are performed at a constant external load level to reduce the residual sufficiently close to zero. Each iteration allows a new estimate of the strain rate to be determined at the state $n + 1$ which is associated with the iteration. With this estimate, the stress at the i th manifestation of state $n + 1$ is updated. This process is termed the stress recovery and is the principal focus of a material model. If the stress is recovered using the strain rate estimate from n to $n + 1$ at i and the stress at n , then the recovery is called path independent. If the stress is recovered using the strain rate estimate from $n + 1$ at $i - 1$ to $n + 1$ at i and the stress for $n + 1$ at $i - 1$, then the recovery is said to be path dependent. In NLDFEP the path independent strategy is employed with no subincrementation, or subdivision of increments over the time step.

Also necessary at each global iteration is a constitutive tangent operator that relates stress rate to strain rate, or changes in stress to changes in strain, so that increments of displacement from $n + 1$ at $i - 1$ to $n + 1$ at i may be computed and strain rates estimated. This task is also the responsibility of the material model.

The small strain plasticity model in NLD FEP is based on rate independent isotropic J_2 flow theory considering both isotropic and kinematic hardening and utilizing a bilinear uniaxial material response.. The stress recovery during plastic flow is performed using an elastic predictor – radial return numerical integration scheme. A consistent rather than a continuum tangent operator is computed for use in the calculation of the element tangent stiffness matrix in order to maintain quadratic convergence in the global nonlinear solution. The complete algorithm for the stress recovery and the evaluation of the consistent tangent operator at a given material point is developed and outlined in the following discussion.

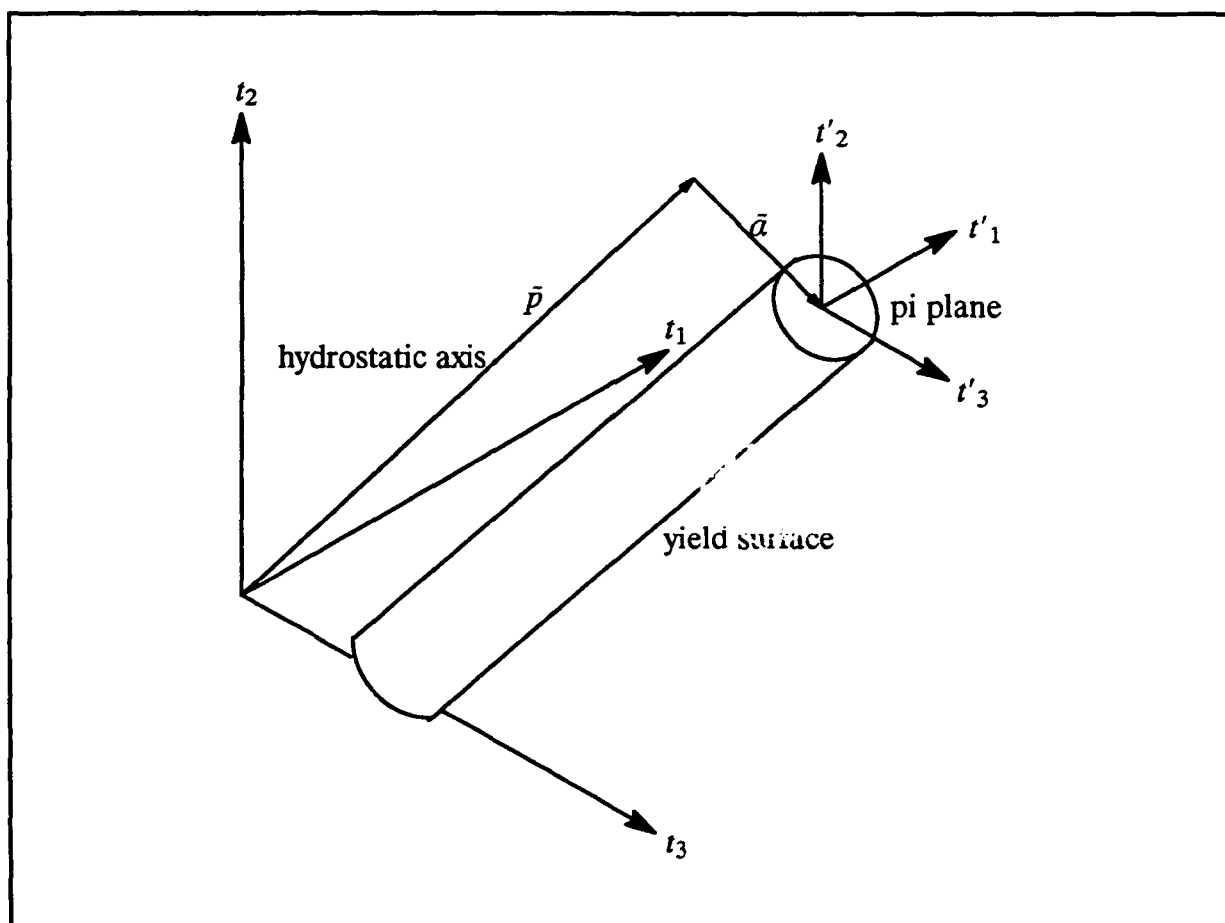


Fig. (2.4-3), Von Mises yield surface in principal stress space.

2.4.4.1 STRESS RECOVERY

Let t_{ij} , d_{ij} , and α_{ij} be the stress, strain rate, and back stress respectively. Deviator values and norms associated with these tensors are defined by

$$(\)'_{ij} = (\)_{ij} - \frac{(\)_{kk}}{3} \delta_{ij} ; \quad \| (\)_{ij} \| = \sqrt{(\)_{ij} (\)_{ij}} \quad (2.4-55)$$

Because the vector corresponding to α_{ij} in principal stress space lies in the π plane of the yield surface, α_{kk} is zero and the deviator of the back stress is the back stress. Accordingly, the deviator relative stress is given as

$$\xi'_{ij} = t'_{ij} - \alpha_{ij} \quad (2.4-56)$$

allowing the Von Mises yield surface (Fig. 2.4-3) to be described by the equation

$$\frac{\xi'_{ij} \xi'_{ij}}{2} - k^2 = 0 \quad (2.4-57)$$

where k is proportional to the radius of the yield surface in the π plane.

The strain rate is decomposed into elastic and plastic components by the equation

$$d_{ij} = d^e_{ij} + d^p_{ij} \quad (2.4-58)$$

The unit normal tensor is defined as

$$n_{ij} = \frac{\xi'_{ij}}{\| \xi'_{ij} \|} \quad (2.4-59)$$

so that plastic strain rate can be described by the equation

$$d^p_{ij} = \lambda n_{ij} \quad (2.4-60)$$

Increments of the plastic strain will thus be normal to the yield surface in stress space. As a consequence of J_2 flow theory, d^p_{kk} , the change in plastic volume with time, is zero; the deviator plastic strain rate is therefore equal to the plastic strain rate.

The effective plastic strain rate and the effective stress are defined as

$$\dot{\epsilon}^p = \sqrt{(2/3) d^p_{ij} d^p_{ij}} \quad (2.4-61)$$

and

$$q = \sqrt{3J_2'} ; \quad J_2' = \frac{1}{2} t'_{ij} t'_{ij} \quad (2.4-62)$$

The derivative of the effective stress with respect to the effective strain is the plastic modulus h . For a Von Mises yield surface with a bilinear uniaxial stress-strain diagram, h is given as

$$h = \frac{EE_t}{E - E_t} \quad (2.4-63)$$

where E and E_t are Young's modulus and the tangent modulus, respectively. Note that for a bilinear material, E_t and h are constants.

Along with equation (60), the evolution equations for the material are given by

$$\dot{\alpha}_{ij} = \frac{2}{3}(1 - \beta)h d p_{ij} = \frac{2}{3}(1 - \beta)h \lambda n_{ij} \quad (2.4-64)$$

$$\dot{k} = \frac{\sqrt{2}}{3}\beta h \| d p_{ij} \| = \frac{\sqrt{2}}{3}\beta h \lambda \quad (2.4-65)$$

$$\dot{t}'_{ij} = 2G(d'_{ij} - d p'_{ij}) \quad (2.4-66)$$

$$\dot{p} = \frac{\dot{t}_{kk}}{3} = K \dot{d}_{kk} \quad (2.4-67)$$

The parameter β controls the type of hardening used in the analysis. It measures the proportion of the hardening which is isotropic, ranging in value between zero and one. Values of $\beta = 0.0$, 1.0 , and 0.25 indicate pure kinematic hardening, pure isotropic hardening, and 25% isotropic hardening - 75% kinematic hardening. The parameters K and G are the bulk and shear moduli of the material.

The material point is assumed to be strained at a constant rate during the time step. Rate tensors are evaluated at state $n + 1/2$ when integrated to produce an increment over the step. Consequently, the hydrostatic stress p of equation (67) and the elastic predictor trial deviator stress at state $n + 1$ are computed as

$$^{n+1}p = ^n p + K \Delta t ^{n+1/2} d'_{kk} \quad (2.4-68)$$

$$^{n+1}t'_{ij} = ^n t'_{ij} + 2G \Delta t ^{n+1/2} d'_{ij} \quad (2.4-69)$$

The trial deviator relative stress is defined in terms of the trial deviator stress and the back stress at state n :

$$^{n+1}\xi'_{ij} = ^{n+1}t'_{ij} - ^n \alpha_{ij} \quad (2.4-70)$$

At this stage, if the material point is elastic, the stress recovery is essentially complete. It would only remain to reintegrate the hydrostatic stress and the trial deviator stress. If the

material point is in the state of plastic flow, using equation (66) the trial deviator stress is modified by a stress increment corresponding to a radial return to the yield surface in order to calculate the updated deviator stress at state $n+1$:

$${}^{n+1}t'_{ij} = {}^{n+1}t'^t_{ij} - 2G\Delta t {}^{n+1/2}dP'_{ij} = {}^{n+1}t'^t_{ij} - 2g\lambda\Delta t n_{ij} \quad (2.4-71)$$

For simplicity of notation, in equation (71) and beyond λ is taken as evaluated at state $n+1/2$ and n_{ij} at state $n+1$. The updated back stress at state $n+1$ follows from equations (64):

$${}^{n+1}a_{ij} = {}^na_{ij} + \frac{2}{3}(1-\beta)h\lambda\Delta t n_{ij} \quad (2.4-72)$$

Combining equations (71) and (72), the deviator relative stress at state $n+1$ is expressed as

$${}^{n+1}\xi'_{ij} = {}^{n+1}\xi'^t_{ij} - \lambda\Delta t \left[2G + \frac{2}{3}(1-\beta)h \right] n_{ij} \quad (2.4-73)$$

Specifying the unit normal tensor at state $n+1$ to be

$$n_{ij} = \frac{{}^{n+1}\xi'^t_{ij}}{\| {}^{n+1}\xi'^t_{ij} \|} \quad (2.4-74)$$

and substituting into equation (73) leads to the relationship

$$\| {}^{n+1}\xi'_{ij} \| = \| {}^{n+1}\xi'^t_{ij} \| - \lambda\Delta t \left[2G + \frac{2}{3}(1-\beta)h \right] \quad (2.4-75)$$

Recasting equation (57) as

$$\| {}^{n+1}\xi'_{ij} \| - \sqrt{2} {}^{n+1}k = 0 \quad (2.4-76)$$

and noting from equation (65) that

$${}^{n+1}k = {}^nk + \frac{\sqrt{2}}{3}\beta h\lambda\Delta t \quad (2.4-77)$$

allows equation (75) to be manipulated, yielding

$$\lambda\Delta t = \frac{\| {}^{n+1}\xi'^t_{ij} \| - \sqrt{2} {}^nk}{2G(1 + \frac{h}{3G})} \quad (2.4-78)$$

$\lambda\Delta t$ is back substituted into the preceding equations to resolve all stresses and state variables. It is possible to directly compute $\lambda\Delta t$ because h is a constant signifying that the effective stress is a linear function of the effective plastic strain. If this function were not linear, then it would be necessary to iterate to determine $\lambda\Delta t$.

A flow chart illustrating the steps required for the recovery of stresses is displayed in Fig. 2.4-4. The algorithm above is implemented in a vector form. The six distinct components of stress tensors are arrayed in the order $\{ 11 \ 22 \ 33 \ 12 \ 23 \ 13 \}$. Strain tensors correspond to vectors with identical ordering but with diagonal terms doubled to form engineering strains.

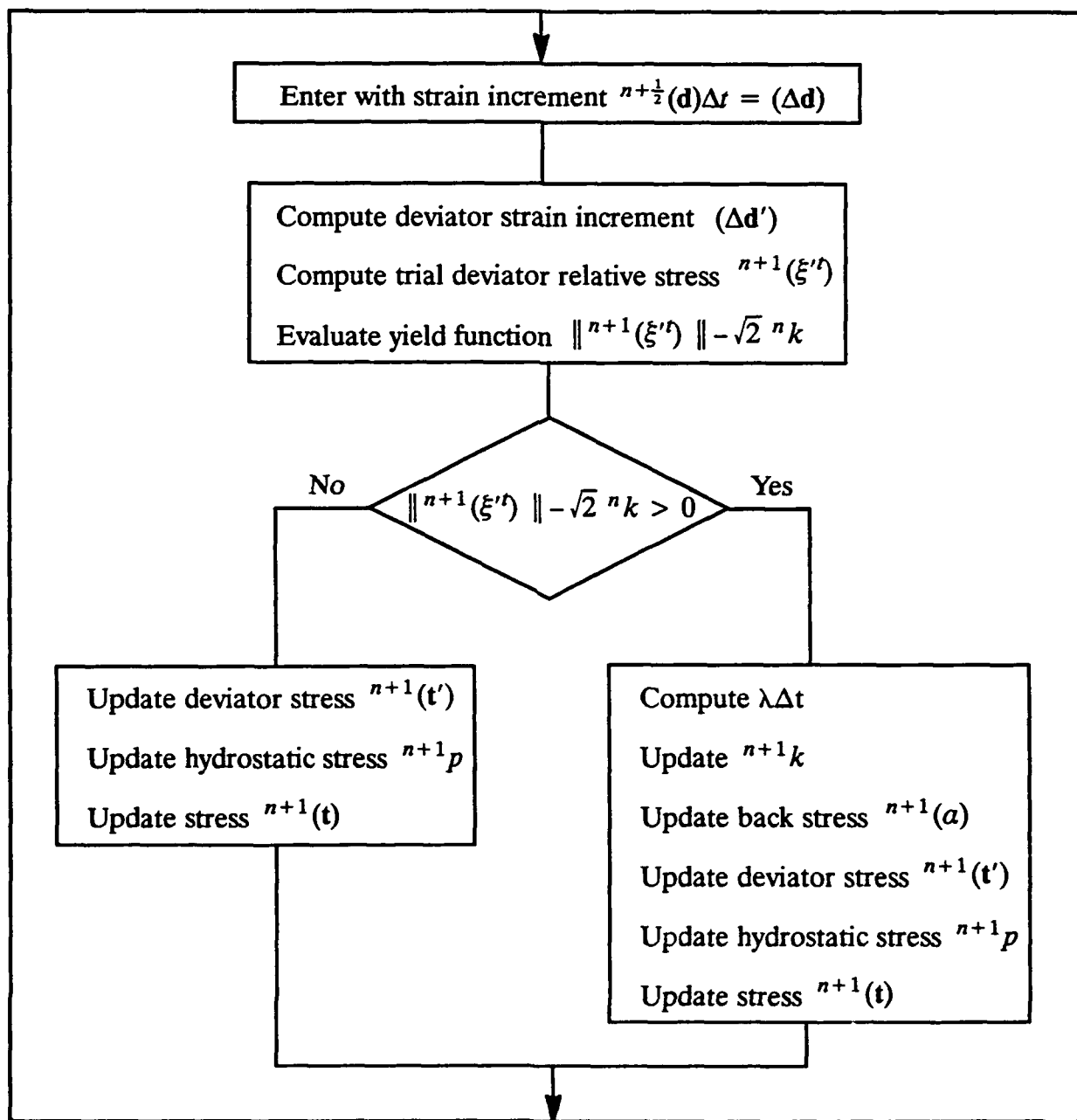


Fig. (2.4-4), Stress recovery flow chart

2.4.4.2 TANGENT OPERATOR

The tangent operator necessary in implicit computational mechanics for the calculation of element tangent stiffness matrices satisfies the following relationship between stress rate and the total strain rate:

$$\dot{\epsilon}_{ij} = C_{ijkl} \dot{\epsilon}_{kl} \quad (2.4-79)$$

For a material point in the elastic state, the isotropic tangent operator is given by

$$C_{ijkl}^E = K\delta_{ij}\delta_{kl} + 2G\left[\frac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) - \frac{1}{3}\delta_{ij}\delta_{kl}\right] \quad (2.4-80)$$

Once the material point experiences plastic flow, the theoretical tangent operator is characterized by

$$C_{ijkl}^{EP} = K\delta_{ij}\delta_{kl} + 2G\left[\frac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) - \frac{1}{3}\delta_{ij}\delta_{kl}\right] - 2G\gamma n_{ij}n_{kl} \quad (2.4-81)$$

$$\gamma = \frac{1}{\left[1 + \left(\frac{h}{3G}\right)\right]} \quad (2.4-82)$$

The operator of equation (81) is termed the continuum tangent operator. Its use is compatible with an exact integration of the evolution equations, which are continuum in nature. However, the preceding elastic predictor – radial return development does not represent an exact integration; it is in essence a secant approach. Not surprisingly, use of the continuum tangent operator leads to a degradation in the asymptotic quadratic convergence characteristic of the Newton–Raphson global nonlinear iterative solution method [78]. Simo and Taylor [78] established a tangent operator which is compatible with the elastic predictor–radial return algorithm and preserves the quadratic convergence. It is called the consistent tangent operator and is the tangent operator employed in NLD FEP. The consistent tangent operator is delineated by the following equations:

$$C_{ijkl}^{EP} = K\delta_{ij}\delta_{kl} + 2GB\left[\frac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) - \frac{1}{3}\delta_{ij}\delta_{kl}\right] - 2G\bar{\gamma}n_{ij}n_{kl} \quad (2.4-83)$$

$$B = \frac{[\sqrt{2}^{n+1}k + \frac{2}{3}(1-\beta)h\lambda\Delta t]}{\|^{n+1}\xi'_{ij}\|} ; \quad \bar{\gamma} = \frac{1}{\left[1 + \left(\frac{h}{3G}\right)\right]} - (1-B) \quad (2.4-84)$$

In practice the tangent operator is applied in a 6x6 matrix form that relates a stress vector to an engineering strain vector.

2.4.5 FINITE STRAIN PLASTICITY ALGORITHMS

In order to complement a geometrically nonlinear analysis, the small strain material model must be augmented to include finite strain effects. To accomplish this goal two principal issues must be addressed. The first is how to decompose the measure of strain into elastic and plastic components. In small strain plasticity, this decomposition is additive; certainly it would be most convenient from the standpoint of compatibility to pursue the same course for finite strain plasticity. However, an additive decomposition in the context of geometric nonlinearity is theoretically incorrect. Consider a motion consisting of first an irrecoverable plastic deformation, then a recoverable elastic deformation, then a material rotation. From the definition in equation (1), the deformation gradient for the entire motion can be written as

$$\mathbf{F} = \mathbf{R}\mathbf{U}_e\mathbf{U}_p \quad (2.4-85)$$

Equation (85) implies a multiplicative decomposition of the right stretch tensor given by

$$\mathbf{U} = \mathbf{U}_e\mathbf{U}_p \quad (2.4-86)$$

as opposed to an additive decomposition such as

$$\mathbf{U} = \mathbf{U}_e + \mathbf{U}_p \quad (2.4-87)$$

Considering only the elastic deformation and combining equations (3) and (4), the Green strain from the deformed plastic configuration to the total deformed configuration is

$$\mathbf{E} = \frac{1}{2}(\mathbf{U}_e^2 - \mathbf{I}) \quad (2.4-88)$$

If \mathbf{U}_e is written as

$$\mathbf{U}_e = \mathbf{I} + \boldsymbol{\delta} \quad (2.4-89)$$

then after some manipulation equation (88) becomes

$$\mathbf{E} = \boldsymbol{\delta} + \frac{\boldsymbol{\delta}^2}{2} \quad (2.4-90)$$

Assuming that the elastic deformation is small enough so that $\boldsymbol{\delta}^2$ is negligible compared to $\boldsymbol{\delta}$, then $\boldsymbol{\delta}$ is seen to be the linear strain tensor.

Choosing the additive decomposition of \mathbf{U} and applying equation (10), the elastic and plastic unrotated rate of deformation tensor can be defined as

$$\mathbf{d}_e = \frac{\mathbf{U}_e^{-T} \dot{\mathbf{U}}_e^T + \dot{\mathbf{U}}_e \mathbf{U}_e^{-1}}{2} \quad (2.4-91)$$

and

$$\mathbf{d}_p = \frac{\mathbf{U}_p^{-T} \dot{\mathbf{U}}_p^T + \dot{\mathbf{U}}_p \mathbf{U}_p^{-1}}{2} \quad (2.4-92)$$

where

$$\mathbf{d} = \mathbf{d}_e + \mathbf{d}_p \quad (2.4-93)$$

Using the multiplicative decomposition of (86), \mathbf{d} is given by

$$\mathbf{d} = \mathbf{d}_e + \frac{\mathbf{U}_e^{-T} \mathbf{U}_p^{-T} \dot{\mathbf{U}}_p^T \mathbf{U}_e^T + \mathbf{U}_e \dot{\mathbf{U}}_p \mathbf{U}_p^{-1} \mathbf{U}_e^{-1}}{2} \quad (2.4-94)$$

If the linear strains of the presumed small elastic deformation are insignificant compared to unity when evaluating the rightmost term of the rhs of equation (94), then equation (94) reduces to equation (93) and the additive decomposition is valid. The supposition of small elastic deformation is reasonable considering that it is limited by the yield level and the generally low level of hardening characteristic of ductile metals. Consequently, equation (93) or an equivalent expression concerning the rate of deformation tensor \mathbf{D} is used to decompose strain in the finite strain plasticity models.

The second question to be settled is that of which type of constitutive relationship the material is assumed to follow. The choices are a hyperelastic material, where total stress is related to total strain, or a hypoelastic material, where stress rate is related to strain rate. For a hyperelastic material, a strain energy functional is differentiated with respect to the nonlinear strains to produce the required secant constitutive law. Actually, a hyperelastic material is a subset of hypoelasticity, as the secant equation can then be differentiated in time to yield one in terms of rates. A hyperelastic approach is most appropriate for path independent materially linear problems that are governed by some global energy functional. For incremental plasticity problems, however, no such energy functional exists and the analysis is decidedly path dependent. Therefore, in order to model such problems, a hypoelastic approach of degree zero is adopted. This is the simplest formulation and is most directly an extension of finite strain plasticity. It should be noted that a hypoelastic formulation is path dependent for elastic problems as well due to its incremental nature.

Fortunately, elastic strains prior to yield are small enough so that this fact is not a significant problem.

The balance of this subsection is devoted to the development and presentation of the finite strain plasticity models associated with both the unrotated stress rate and the Truesdell stress rate. Note that a Total Lagrangian approach is adopted for the global solution so that stress and the constitutive law must be transformed to the reference configuration.

2.4.5.1 THE UNROTATED STRESS RATE

The following series of computations are executed at a given material point for the stress recovery using the unrotated stress rate. Both the tensor form and equivalent vector form of the associated equations are given. All stress tensors are manipulated in stress vector form and all strain tensors are manipulated in engineering strain vector form except the right stretch tensor, for which the upper triangular vector form is used. Note that rate tensors are assumed constant over a time step and that tensor increments correspond to rate tensors evaluated at $n + 1/2$ multiplied by the time increment Δt . In the following development, the left side of the arrow represents the continuum equations, and the right side represents the equivalent matrix and vector forms.

- 1) Using the converged displacements at n , \mathbf{u}_n , the i th estimate of the displacements at $n + 1$, \mathbf{u}_{n+1} , and the mid-point evaluation, $\mathbf{u}_{n+1/2}$, compute required deformation gradients:

$$\mathbf{F}_{n+1} = \frac{\partial(\mathbf{X} + \mathbf{u}_{n+1})}{\partial \mathbf{X}} \longrightarrow [\mathbf{F}]_{n+1} \quad (2.4-95)$$

$$\mathbf{F}_{n+1/2} = \frac{\partial(\mathbf{X} + \mathbf{u}_{n+1/2})}{\partial \mathbf{X}} \longrightarrow [\mathbf{F}]_{n+1/2} \quad (2.4-96)$$

$$\Delta \mathbf{F} = \frac{\partial(\mathbf{u}_{n+1} - \mathbf{u}_n)}{\partial \mathbf{X}} \longrightarrow [\Delta \mathbf{F}] \quad (2.4-97)$$

- 2) Compute the displacement increment gradient associated with the integration of the velocity gradient :

$$\Delta \mathbf{L} = \Delta \mathbf{F} \mathbf{F}_{n+\frac{1}{2}}^{-1} \longrightarrow [\Delta \mathbf{L}] = [\Delta \mathbf{F}] [\mathbf{F}]_{n+\frac{1}{2}}^{-1} \quad (2.4-98)$$

- 3) Compute the deformation increment associated with the integration of the rate of deformation tensor \mathbf{D} as the symmetric part of $\Delta \mathbf{L}$.

$$\Delta \mathbf{D} = \frac{1}{2}(\Delta \mathbf{L} + \Delta \mathbf{L}^T) \longrightarrow (\Delta \mathbf{D}) \quad (2.4-99)$$

The converged increments of $(\Delta \mathbf{D})$ are accumulated during the solution to provide a measure of logarithmic strain for output.

- 4) Transform the second Piola-Kirchoff stress at n to the unrotated Cauchy stress at n :

$$\mathbf{t}_n = \frac{1}{|\mathbf{F}_n|} \mathbf{U}_n \mathbf{S}_n \mathbf{U}_n \longrightarrow (\mathbf{t})_n = [\mathbf{T}_U]_n (\mathbf{S})_n \quad (2.4-100)$$

The structure of the transformation matrix $[\mathbf{T}_U]$ is given in Appendix A. The right stretch tensor is previously computed and retrieved from storage.

- 5) Decompose the mid-point deformation gradient to obtain the mid-point rotation:

$$\mathbf{F}_{n+\frac{1}{2}} = \mathbf{R}_{n+\frac{1}{2}} \mathbf{U}_{n+\frac{1}{2}} \longrightarrow [\mathbf{R}]_{n+\frac{1}{2}} \quad (2.4-101)$$

- 6) Rotate the increment of deformation $(\Delta \mathbf{D})$ into the unrotated increment of deformation $(\Delta \mathbf{d})$:

$$\Delta \mathbf{d} = \mathbf{R}_{n+\frac{1}{2}}^T \Delta \mathbf{D} \mathbf{R}_{n+\frac{1}{2}} \longrightarrow (\Delta \mathbf{d}) = [\mathbf{Q}_U]_{n+\frac{1}{2}} (\Delta \mathbf{D}) \quad (2.4-102)$$

The structure of matrix $[\mathbf{Q}_U]$ is given in Appendix A.

- 7) Using $(\Delta \mathbf{d})$ as the strain increment and $(\mathbf{t})_n$ as the stress, call the small strain plasticity model to compute the updated unrotated Cauchy stress $(\mathbf{t})_{n+1}$. Material state variables such as the back stress and the effective plastic strain reside in the

unrotated configuration.

- 8) Transform the unrotated Cauchy stress at $n+1$ to the second Piola-Kirchoff stress at $n+1$:

$$\mathbf{S}_{n+1} = |F_{n+1}| \mathbf{U}_{n+1}^{-1} \mathbf{t}_{n+1} \mathbf{U}_{n+1}^{-1} \longrightarrow (\mathbf{S})_{n+1} = [\bar{\mathbf{T}}_U]_{n+1} (\mathbf{t})_{n+1} \quad (2.4-103)$$

The matrix $[\bar{\mathbf{T}}_U]$ is given in Appendix A. A second polar decomposition is required to obtain the right stretch tensor, which is stored for future use upon global convergence of the time step. During the solution of the next time step this quantity is used in step 4.

The current implementation of this algorithm in NLDFEP does not globally store the right stretch tensor at state n as implied in steps 4 and 8. This necessitates a third polar decomposition to obtain this tensor for use in step 4. Future implementations of NLDFEP will apply the algorithm as described above and eliminate the need for the third polar decomposition.

The small strain consistent tangent operator can be invoked to relate the rate of unrotated Cauchy stress to the unrotated rate of deformation tensor. This relationship must be transferred to the reference configuration in accordance with the Total Lagrangian approach. To do this equations coupling the rate of second Piola-Kirchoff stress to the rate of unrotated Cauchy stress and the rate of Green strain to the unrotated rate of deformation tensor must be found. The former is represented by equations (100) and (103). The latter is given by equation (12) and in the reverse form by

$$\mathbf{d} = \mathbf{U}^{-1} \dot{\mathbf{E}} \mathbf{U}^{-1} \quad (2.4-104)$$

By differentiating the tensor form of equation (100) and dropping the subscript n , the rate of second Piola-Kirchoff stress is found to be

$$\begin{aligned} \dot{\mathbf{S}} &= |F| [\mathbf{U}^{-1} \dot{\mathbf{t}} \mathbf{U}^{-1} + tr(\mathbf{D}) \mathbf{S} - [\mathbf{U}^{-1} \dot{\mathbf{U}} \mathbf{S} + (\mathbf{U}^{-1} \dot{\mathbf{U}} \mathbf{S})^T] \\ &= |F| [\mathbf{U}^{-1} [\dot{\mathbf{t}} + tr(\mathbf{D}) \mathbf{t} - \dot{\mathbf{U}} \mathbf{U}^{-1} \mathbf{t} - (\dot{\mathbf{U}} \mathbf{U}^{-1} \mathbf{t})^T] \mathbf{U}^{-1} \end{aligned} \quad (2.4-105)$$

In vector form, the rate of unrotated Cauchy stress and the unrotated rate of deformation tensor are linked by the consistent tangent operator:

$$\dot{\mathbf{t}} = [\mathbf{C}_{EP}](\dot{\mathbf{d}}) \quad (2.4-106)$$

Expressing equations (104) and the second of (105) in vector form and combining them with equation (106) gives

$$\dot{\mathbf{S}} = [\mathbf{C}_{EP}^*](\dot{\mathbf{E}}) \quad (2.4-107)$$

where

$$[\mathbf{C}_{EP}^*] = |F|[\bar{\mathbf{T}}_U][\mathbf{C}_{EP}] - [\mathbf{Q}][\bar{\mathbf{T}}_U]^T \quad (2.4-108)$$

Unfortunately, the matrix \mathbf{Q} is not symmetric. Attempts to implement a symmetric form of \mathbf{Q} have not met with success. Although work is continuing in this area, a much simpler form of equations (105) is obtained if one assumes that

- a) the material is incompressible, forcing $\text{tr}(\mathbf{D})$ to zero ;
- b) the rate of the right stretch tensor is negligible compared to \mathbf{U} and \mathbf{S} or \mathbf{t} .

During plastic flow the first of these assumptions is reasonable due to the incompressibility requirement of J_2 flow theory and the predominance of the plastic deformation over the elastic. The second assumption is valid if gross yielding of the material does not occur resulting in large, rapid changes in deformation. In the elastic range, the deformation is small enough so that no serious error is engendered by the application of the assumptions. The new form of equation (105) is

$$\dot{\mathbf{S}} = |F|\mathbf{U}^{-1}\dot{\mathbf{t}}\mathbf{U}^{-1} \quad (2.4-109)$$

so that equation (108) becomes

$$[\mathbf{C}_{EP}^*] = |F|[\bar{\mathbf{T}}_U][\mathbf{C}_{EP}][\bar{\mathbf{T}}_U]^T \quad (2.4-110)$$

Numerical tests have shown that the simplified equations (109) and (110) do not seriously deteriorate the convergence of the global equilibrium iterations in the absence of gross homogeneous deformation. Even so, difficulties arise only in the case of load control, and

then a simple acceleration technique or a line search greatly enhances convergence, as is discussed in Section 3.5. Until further research provides for successful use of equation (108), equation (110) is applied.

2.4.5.2 THE TRUESDELL STRESS RATE

The required calculations for the stress recovery at a given material point using the Truesdell stress rate are more involved than those for the unrotated stress rate. This fact arises from the nature of the Truesdell stress rate, given by equation (29). The Truesdell stress rate does not correspond to a simple rotation of the rate of Cauchy stress like the unrotated stress rate, but to the rate of second Piola–Kirchhoff stress referred to the current configuration. As such, the integration of the evolution equations cannot be performed in a convenient intermediate configuration, but in the current configuration itself. Thus, the Cauchy stress at n must be updated by an integration of the velocity gradient terms in equation (29) along with the increment of stress furnished by the evolution equations. In addition, the rate of back stress must be objective; for reasons of consistency the Truesdell rate of back stress is used. It too must be updated by the integration of similar velocity gradient terms. Disturbingly, this transformation does not preserve the deviator nature of the back stress. This dilemma can be dealt with in one of two ways. First, the hydrostatic back stress can be permanently purged from the solution as soon as it appears. This would compromise the objectivity of the stress rate. Or, the hydrostatic back stress can be removed before use of the back stress in the stress recovery and replaced afterward. This preserves objectivity and does not corrupt the deviator calculations in the stress recovery. This is the path chosen in NLDFEP. Either way, kinematic hardening problems are almost certainly rendered problematical by the use of the Truesdell stress rate.

The following sequence of operations are performed at a given material point for the stress recovery using the Truesdell stress rate. Again, both the tensor form and equivalent vector form of the associated equations are given, with all stress tensors employed in stress vector form and all strain tensors employed in engineering strain vector form. Rate tensors are once more assumed constant over a time step and that tensor increments correspond to rate tensors evaluated at $n + 1/2$ multiplied by the time increment Δt . As before, the left side of the arrow represents the continuum equations, and the right side represents the equivalent matrix and vector forms.

1–3) The first three steps are identical to those pertaining to the unrotated stress rate.

Picking τ the algorithm in the fourth step, then:

4) Transform the second Piola–Kirchhoff stress at n to the Cauchy stress at n :

$$\mathbf{T}_n = \frac{1}{|\mathbf{F}_n|} \mathbf{F}_n \mathbf{S}_n \mathbf{F}_n^T \longrightarrow (\mathbf{T})_n = [\mathbf{T}_T]_n (\mathbf{S})_n \quad (2.4-111)$$

The structure of the transformation matrix $[\mathbf{T}_T]$ is given in Appendix A.

- 5) Transform the Cauchy stress at n by integration of the velocity gradient terms in the Truesdell stress rate:

$$\bar{\mathbf{T}}_n = \mathbf{T}_n + \Delta \mathbf{V} \mathbf{T}_n + \mathbf{T}_n \Delta \mathbf{V}^T - tr(\Delta \mathbf{D}) \mathbf{T}_n \longrightarrow (\bar{\mathbf{T}})_n = [\mathbf{Q}_T]_{n+\frac{1}{2}} (\mathbf{T})_n \quad (2.4-112)$$

The structure of matrix $[\mathbf{Q}_T]$ is given in Appendix A. Since the Cauchy stress at $n + 1/2$ is not available, the Cauchy stress at n is used in the integration. This choice does lead to some error, as will be discussed presently.

- 6) Transform the back stress at n by integration of the velocity gradient terms in the Truesdell stress rate:

$$\bar{\mathbf{a}}_n = \mathbf{a}_n + \Delta \mathbf{V} \mathbf{a}_n + \mathbf{a}_n \Delta \mathbf{V}^T - tr(\Delta \mathbf{D}) \mathbf{a}_n \longrightarrow (\bar{\mathbf{a}})_n = [\mathbf{Q}_T]_{n+\frac{1}{2}} (\mathbf{a})_n \quad (2.4-113)$$

As above, the back stress at $n + 1/2$ is not available. The back stress at n is thus used in the integration.

- 7) Compute the deviator values of the back stress at n :

$$\bar{\mathbf{a}}'_n = \bar{\mathbf{a}}_n - \frac{1}{3} tr(\bar{\mathbf{a}}_n) \mathbf{I} \longrightarrow (\bar{\mathbf{a}}')_n \quad (2.4-114)$$

- 8) Using $(\Delta \mathbf{D})$ as the strain increment and $(\bar{\mathbf{T}})_n$ as the stress, call the small strain plasticity model to compute the updated Cauchy stress $(\mathbf{T})_{n+1}$. Material state variables other than the back stress such as the effective plastic strain reside in the current configuration.

- 9) Replace the hydrostatic back stress extracted in step 7:

$$a_{n+1} = a'_{n+1} + \frac{1}{3} \text{tr}(\bar{a}_n) \mathbf{I} \longrightarrow (a)_{n+1} \quad (2.4-115)$$

10) Transform the Cauchy stress at $n+1$ to the second Piola-Kirchoff stress at $n+1$:

$$\mathbf{S}_{n+1} = |F_{n+1}| \mathbf{F}_{n+1}^{-1} \mathbf{T}_{n+1} \mathbf{F}_{n+1}^{-T} \longrightarrow (\mathbf{S})_{n+1} = [\bar{\mathbf{T}}_T]_{n+1} (\mathbf{T})_{n+1} \quad (2.4-116)$$

The matrix $[\bar{\mathbf{T}}_T]$ is given in Appendix A.

For the Truesdell stress rate, the small strain consistent tangent operator relates the Truesdell stress rate to the rate of deformation tensor. Again, this relationship must be expressed in the reference configuration. The Truesdell stress rate is associated with the rate of second Piola-Kirchoff stress by equation (30) and the rate of Green strain with the rate of deformation tensor by equation (11). These equations are given in the reverse form by

$$\dot{\mathbf{S}} = |F| \mathbf{F}^{-1} \hat{\mathbf{T}}_T \mathbf{F}^{-T} \quad (2.4-117)$$

and

$$\mathbf{D} = \mathbf{F}^T \dot{\mathbf{E}} \mathbf{F}^{-1} \quad (2.4-118)$$

In vector form, the constitutive law between the Truesdell stress rate and the rate of deformation tensor is

$$(\hat{\mathbf{T}}_T) = [\mathbf{C}_{EP}](\mathbf{D}) \quad (2.4-119)$$

Expressing equations (117) and (118) in vector form and combining them with equation (119) gives

$$(\dot{\mathbf{S}}) = [\mathbf{C}_{EP}^*](\dot{\mathbf{E}}) \quad (2.4-120)$$

where

$$[\mathbf{C}_{EP}^*] = |F| [\bar{\mathbf{T}}_T] [\mathbf{C}_{EP}] [\bar{\mathbf{T}}_T]^T \quad (2.4-121)$$

Equation (121) exhibits the convenient form of equation (110) without assumption or restriction. This is a benefit of using the Truesdell stress rate.

2.4.6 NUMERICAL EXAMPLES

In this section, the theory and application of the finite strain plasticity models based on the Truesdell and unrotated stress rates were presented, from the hypoelastic relationship between objective stress and strain rates to the polar decomposition algorithm to the small strain plasticity algorithm in its role as a kernel for the finite strain models. It remains to determine which, if either, of the finite strain plasticity models is preferable to the other. Numerical examples are presented in Section 3.5 to assess the performance of these models in a series of representative problems. Based on the results of these problems, recommendations concerning the usage of the finite strain plasticity models are made.

3 NUMERICAL RESULTS: FINITE ELEMENT ANALYSES

A number of example problems have been solved with NLD FEP in order to examine the performance of the various element computation algorithms and nonlinear solution algorithms previously described. In this chapter the results of the finite element analyses of these example problems are presented to demonstrate the accuracy and viability of finite element simulations performed with NLD FEP. While these analyses are not the main focus of this research, it is important to show that NLD FEP can accomplish the tasks for which it was designed.

3.1 BAR EXTENSION PROBLEM

The bar extension problem is a simple one consisting of a one dimensional array of eight noded three dimensional isoparametric elements. The details of the mesh are illustrated in Fig. 3.1-1. This problem was chosen in order to examine the characteristics of a one dimensional mesh. It also provides a solution that is readily verifiable by a straightforward theoretical analysis.

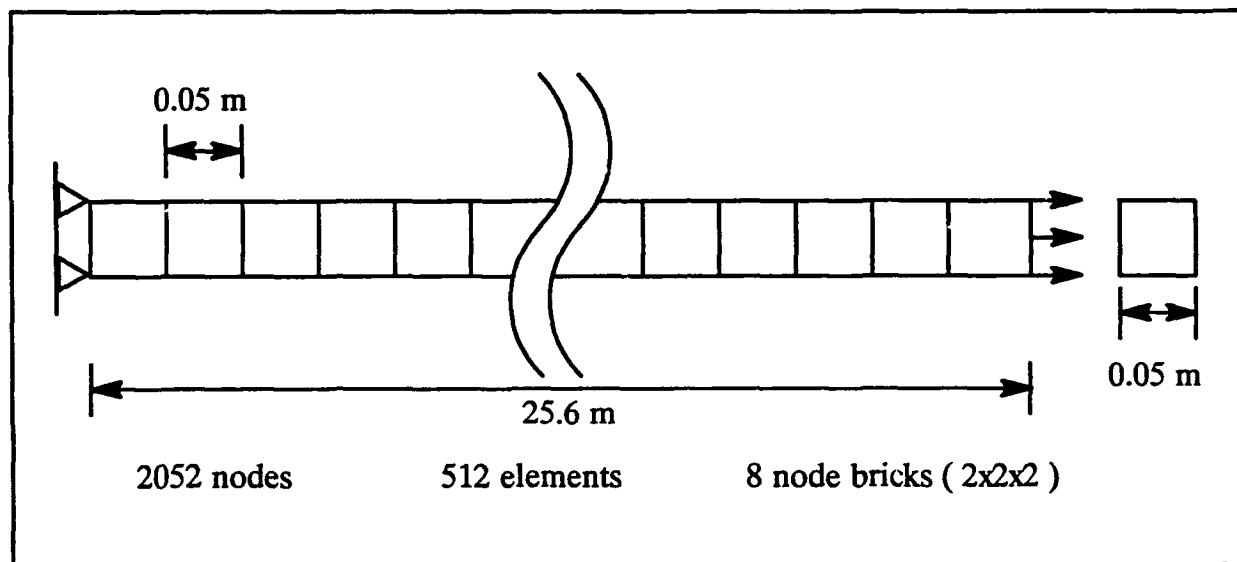


Fig. (3.1-1), finite element mesh for bar extension problem

A description of the loading history and the material properties used in the finite element analysis is depicted in Fig. 3.1-2. The bar is extended by a tensile force applied at the end of the bar that is increased linearly with time and reaches a maximum value corresponding to an axial stress of approximately 300 Mpa in 10 milliseconds and then is held constant.

The stress is approximately 300 Mpa due to the fluctuation in the cross-sectional area with time that affects the magnitude of the axial stress. This value of axial stress is significantly larger than the yield stress of the material, which is 250 Mpa. The analysis was performed in 80 equal time steps of 1.0 milliseconds, using a value of β equal to $1/4$. The strain hardening is assumed isotropic and is represented by a bilinear tangent modulus of 10000.0 Mpa. This value was selected to provide a plastic wave velocity that is readily observable. A Poisson's ratio of zero was chosen to furnish an elastic response that is most like the one dimensional idealization of a bar. Of course, during plastic flow an effective Poisson's ratio of nearly 0.5 occurs due to the restriction on plastic volume change. Fixed end boundary conditions were applied to maximize the number of active dof (6144) for timing purposes. A lumped mass matrix was employed.

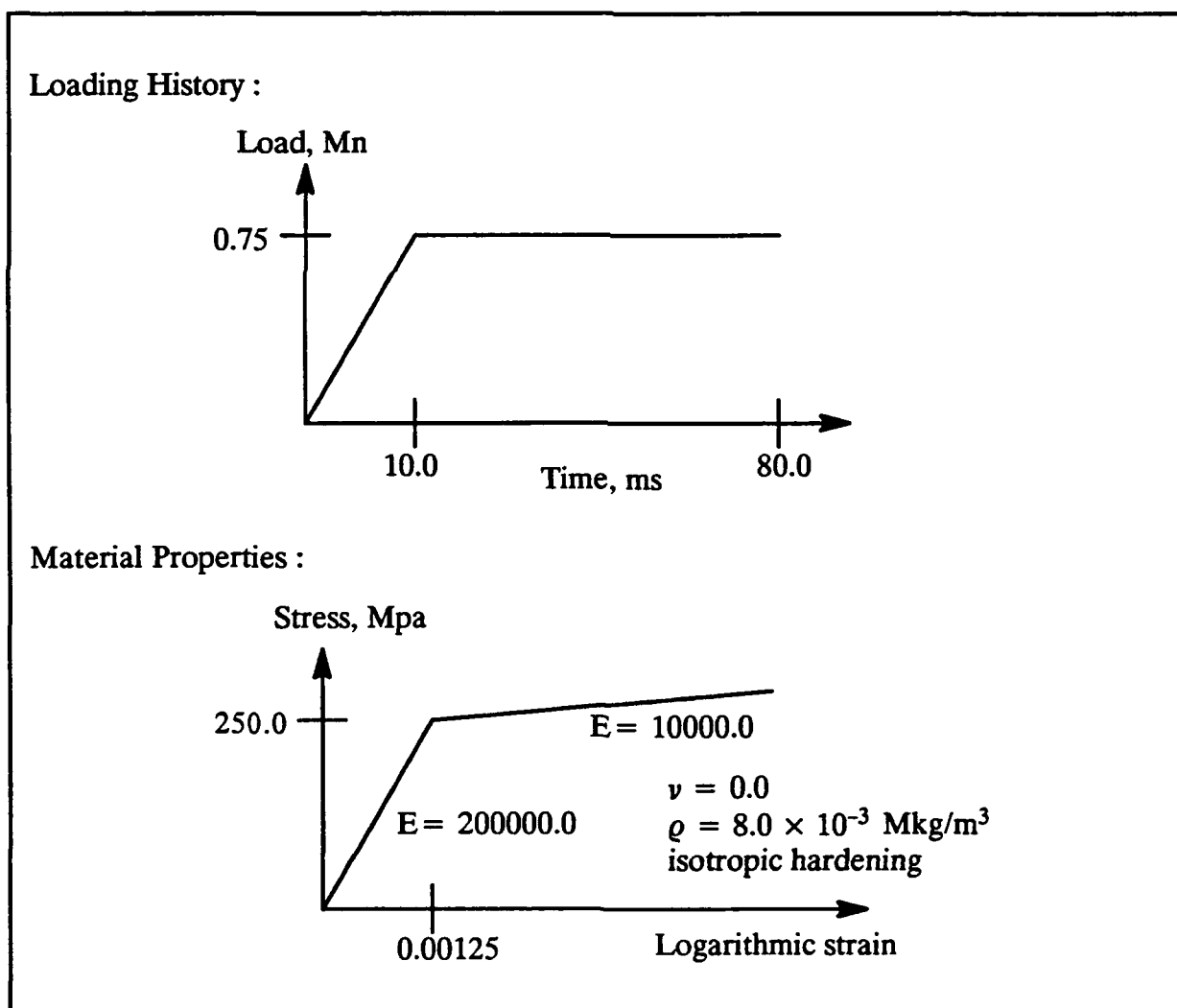


Fig. (3.1-2), problem definition for bar extension problem

For a linear elastic material and excluding the effects of geometric nonlinearity, the fundamental natural frequency of vibration of a bar fixed at one end is specified as

$$\omega_1 = \frac{\pi}{2} \sqrt{\frac{EA}{ml^2}} \quad (3.1-1)$$

where m is the mass per unit length of the bar and l is the total length [55]. The equation of motion of an idealized bar where the stress acts uniformly over each cross-section and the cross-section remains plane [50] is given by

$$\rho \frac{\partial^2 u}{\partial t^2} = E \frac{\partial^2 u}{\partial x^2} \quad (3.1-2)$$

Equation (2) is the well known one dimensional wave equation. The velocity of propagation of longitudinal elastic waves along the length of the bar is

$$c_0 = \sqrt{\frac{E}{\rho}} \quad (3.1-3)$$

From the solution of the wave equation, for an elastic wave pulse traveling in the direction of decreasing x , the following relationship exists between the axial stress and the particle velocity:

$$\sigma_{xx} = \rho c_0 \frac{\partial u}{\partial t} \quad (3.1-4)$$

After the onset of plastic flow, the velocity of propagation of longitudinal plastic waves along the length of the bar becomes

$$c_p = \sqrt{\frac{E_t}{\rho}} \quad (3.1-5)$$

While the above equations are for geometrically linear problems, the geometric nonlinearity in the bar extension problem is insignificant in the range of elastic response and is not pronounced during plastic flow due to the high tangent modulus employed. Thus the above equations can be used to evaluate the accuracy of the finite element results.

Plots for displacement, velocity, acceleration, and stress versus time for selected points along the length of the bar are given in Figs. 3.1-4 to 3.1-7. Considering the plot for displacement, one can observe that somewhere in the vicinity of 48 milliseconds the bar experiences elastic unloading into a mode of free vibration. By measuring the time between peaks of the displacement plots the period of this vibration is found to be about 0.02 seconds. From equation (1), the fundamental frequency is calculated as 306.79 radians/second, which trans-

lates to a period of 0.0205 seconds. Thus, the bar unloads to a free vibration dominated by the fundamental mode.

The table presented in Fig. 3.1-3 tests the agreement with equation (4). Results are given for $x = 20$ meters and for times between 1 and 5 milliseconds. From equation (3), the velocity of an elastic wave is calculated as 5000 meters/second, meaning that the elastic waves resulting from the loading at the right end will hit the supports and begin rebounding after just over 5 milliseconds, or 5 time steps. Selection of $x = 20$ meters avoids any effects of rebounding waves, which negate equation (4), and which are accentuated by the smoothing effect of implicit dynamics. From Fig. 3.1-3 it is seen that the agreement between the computed axial stress and that estimated with equation (4) is excellent.

time (ms)	velocity (m/s)	axial stress (Mpa)	$\rho c_0 \times \text{velocity}$ (Mpa)
1.0	0.0798429	3.19385	3.19371
2.0	0.517357	20.6943	20.6943
3.0	1.39842	55.9334	55.9366
4.0	2.23131	89.2479	89.2523
5.0	2.86029	114.444	114.411

Fig. (3.1-3), comparison of computed versus scaled particle velocity, $x = 20$ m

From the peaks in acceleration readily discernible in Fig. 3.1-6 the progress of the plastic wave front corresponding to the increase in stress from 250 to 300 Mpa during plastic flow is easily tracked. One can also clearly observe the evolution of the plastic wave in the velocity plot, Fig. 3.1-5. By measuring the time differences between peaks and knowing the distance separating the sample points, the plastic wave velocity is determined to be between 1100 and 1120 meters/second. From equation (5), this velocity is calculated as 1118 meters/second.

The plot of axial stress, Fig. 3.1-7, is an interesting one. The plastic wave front can again be distinctly followed, and from the plot for $x = 5$ meters one can observe the reflection of the wave front from the fixed end, causing the plastic change in stress to nearly double. The elastic unloading to free vibration is again conspicuous, with noise from higher modes notice-

able. This noise also exists in the acceleration and velocity plots, but not in the displacement plot. The displacements due to the higher modes are not large enough to observe. The slight effect of geometric nonlinearity can be seen from the way the stress at the loaded end oscillates about approximately 304 Mpa. The average stress is greater than 300 Mpa due to the decrease in cross-sectional area and the oscillation occurs due to vibrations which cause the area to fluctuate.

The bar extension problem is not a particularly complicated one. It was solved because a purely one dimensional mesh was desired for evaluation of the element computation and nonlinear solution algorithms; its simplicity also allows for an elementary test of the finite element analysis. Based on these results, for the bar extension problem NLDFEP is an operable finite element program.

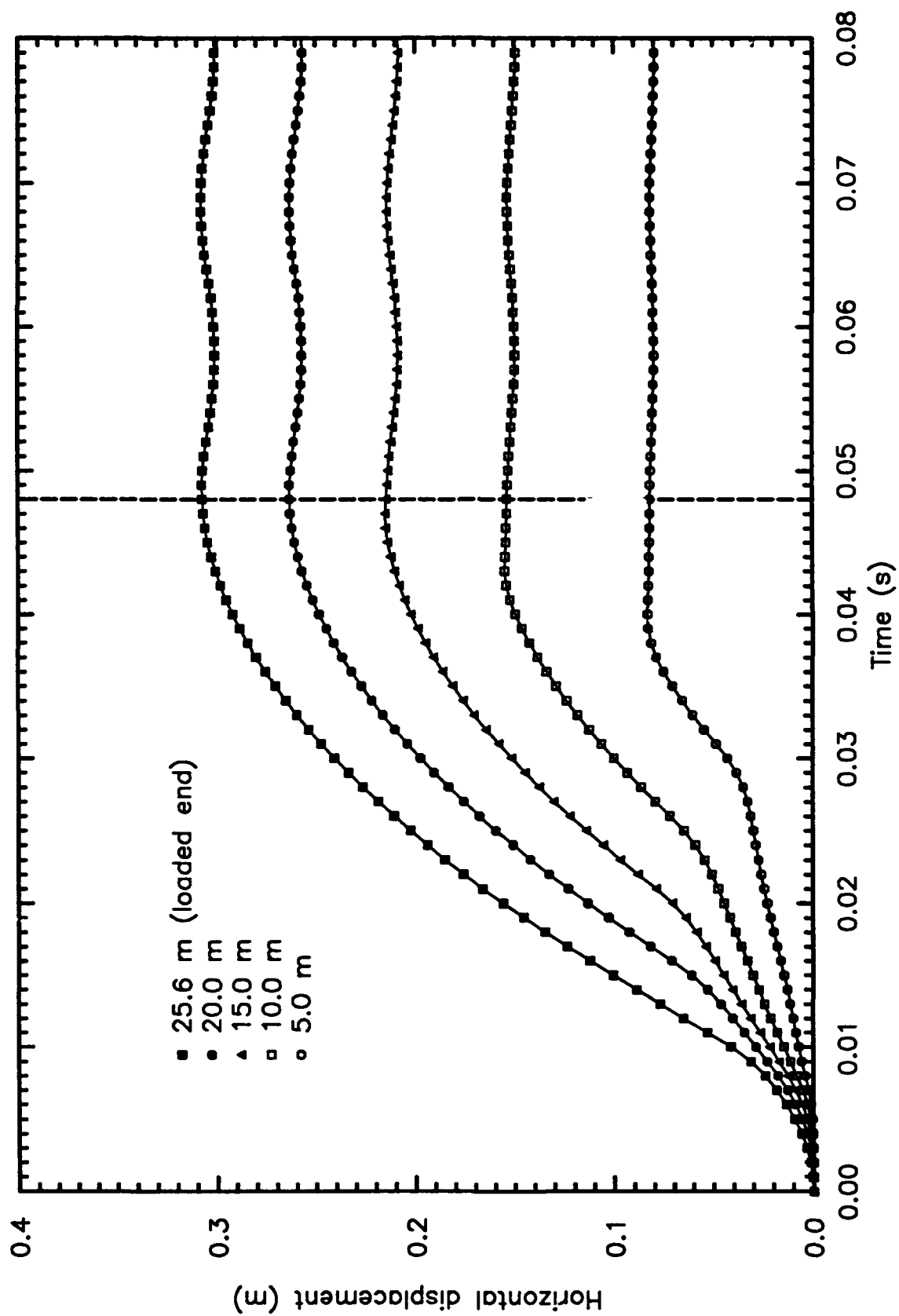


Fig. (3.1-4), horizontal displacement vs. time for selected points along the bar

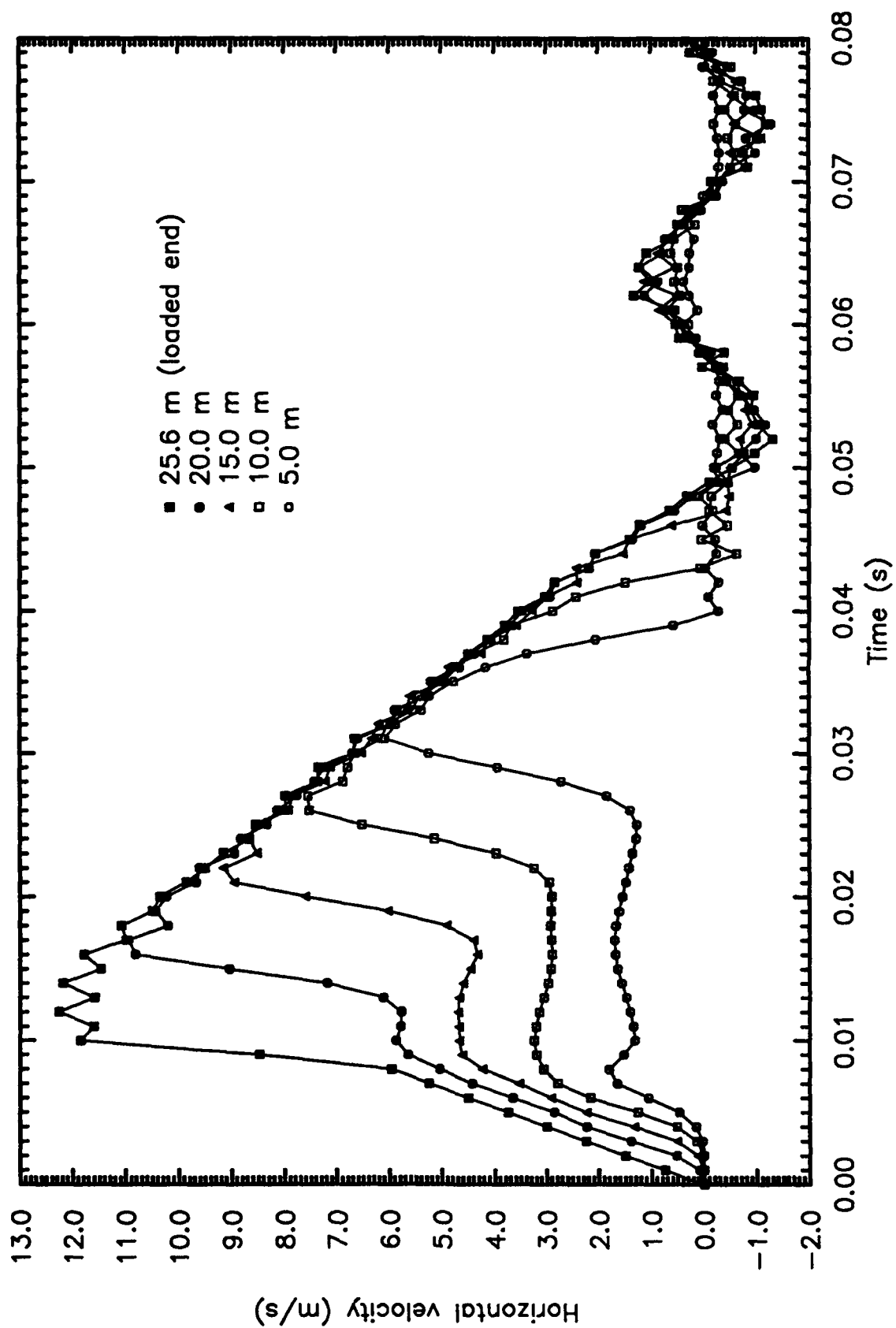


Fig. (3.1-5), horizontal velocity vs. time for selected points along the bar

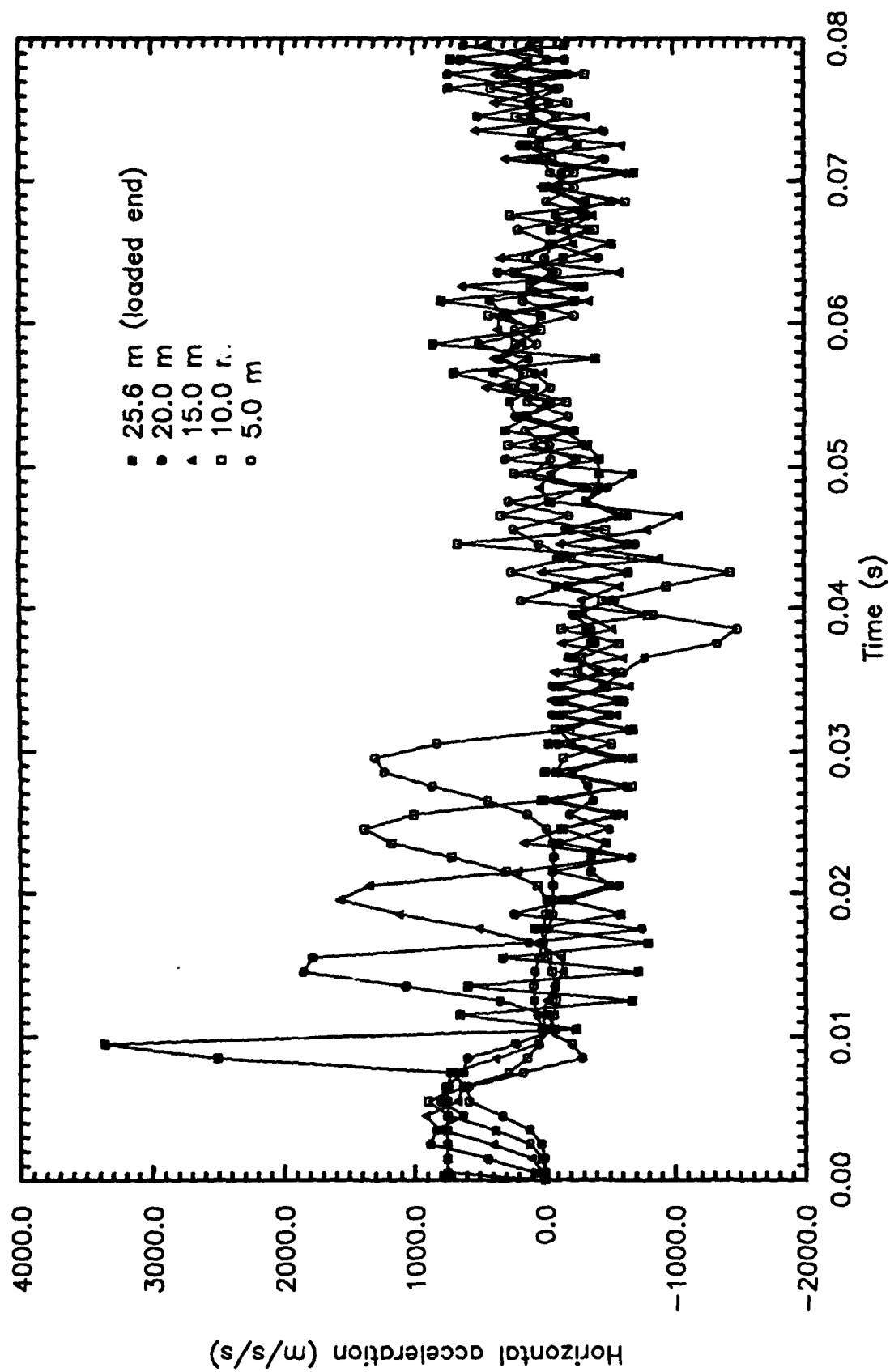


Fig. (3.1-6), horizontal acceleration vs. time for selected points along the bar

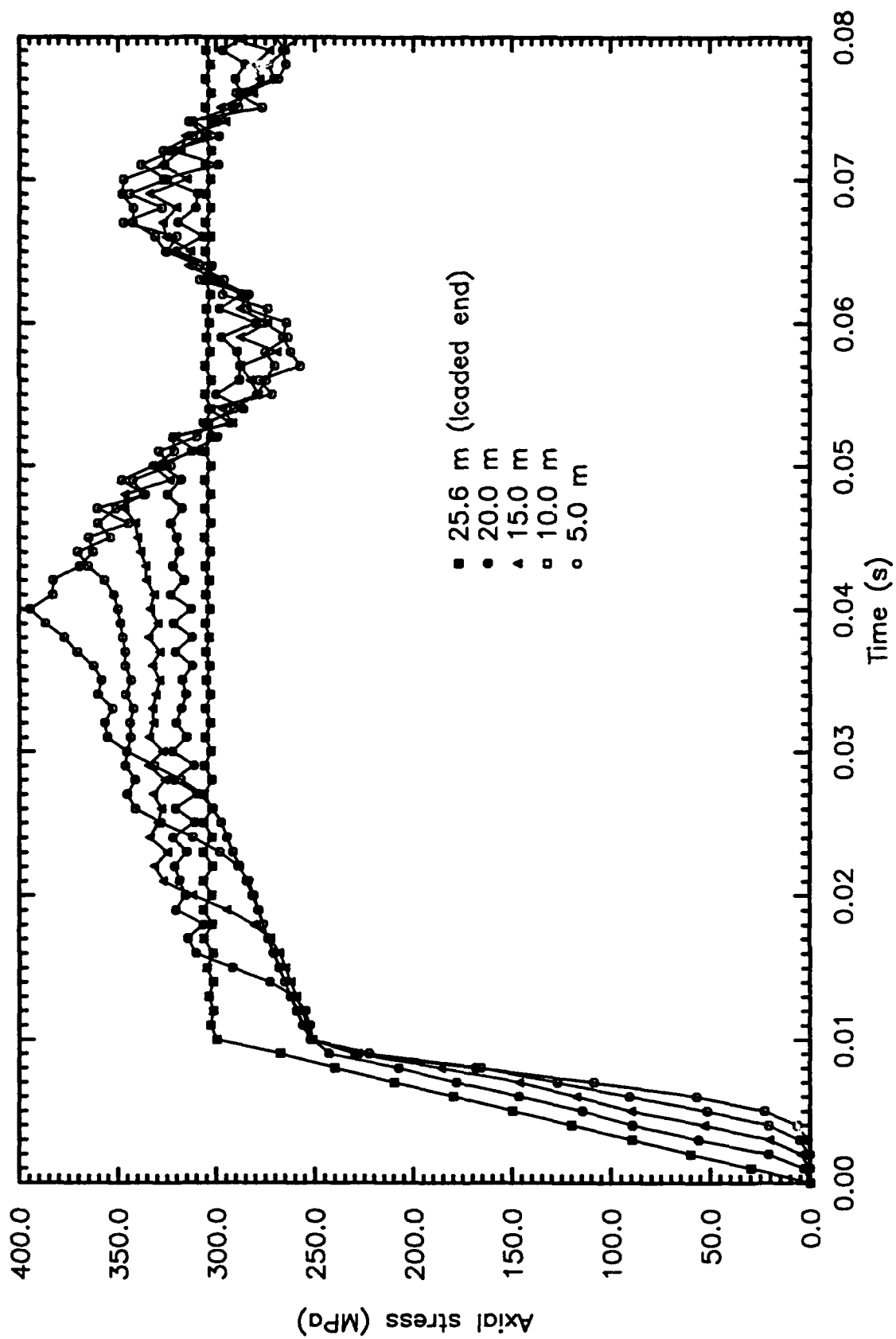


Fig. (3.1-7), axial stress vs. time for selected points along the bar

3.2 SHEAR BAND PROBLEM

The shear band problem provides a test of the element computation and nonlinear solution algorithms for a two dimensional mesh. From an engineering standpoint, it demonstrates the ability of NLD FEP to simulate the development of a shear band. A shear band is a localized zone of high deformation observed in tests. In this case, the band should extend from the zone of maximum deformation in the vicinity of the square hole in the center of the plate to the free edge. The mesh, shown in Fig. 3.2-1, represents one quarter of a rectangular plate with a square hole in the center. Symmetry boundary conditions are imposed on the center boundaries and a tensile load is applied on the end. Plane strain conditions are enforced by constraining all out of plane displacements to be zero, leaving the total number of active degrees of freedom at 7372. All elements are 20 node bricks employing reduced 2 by 2 by 2 integration to avoid locking.

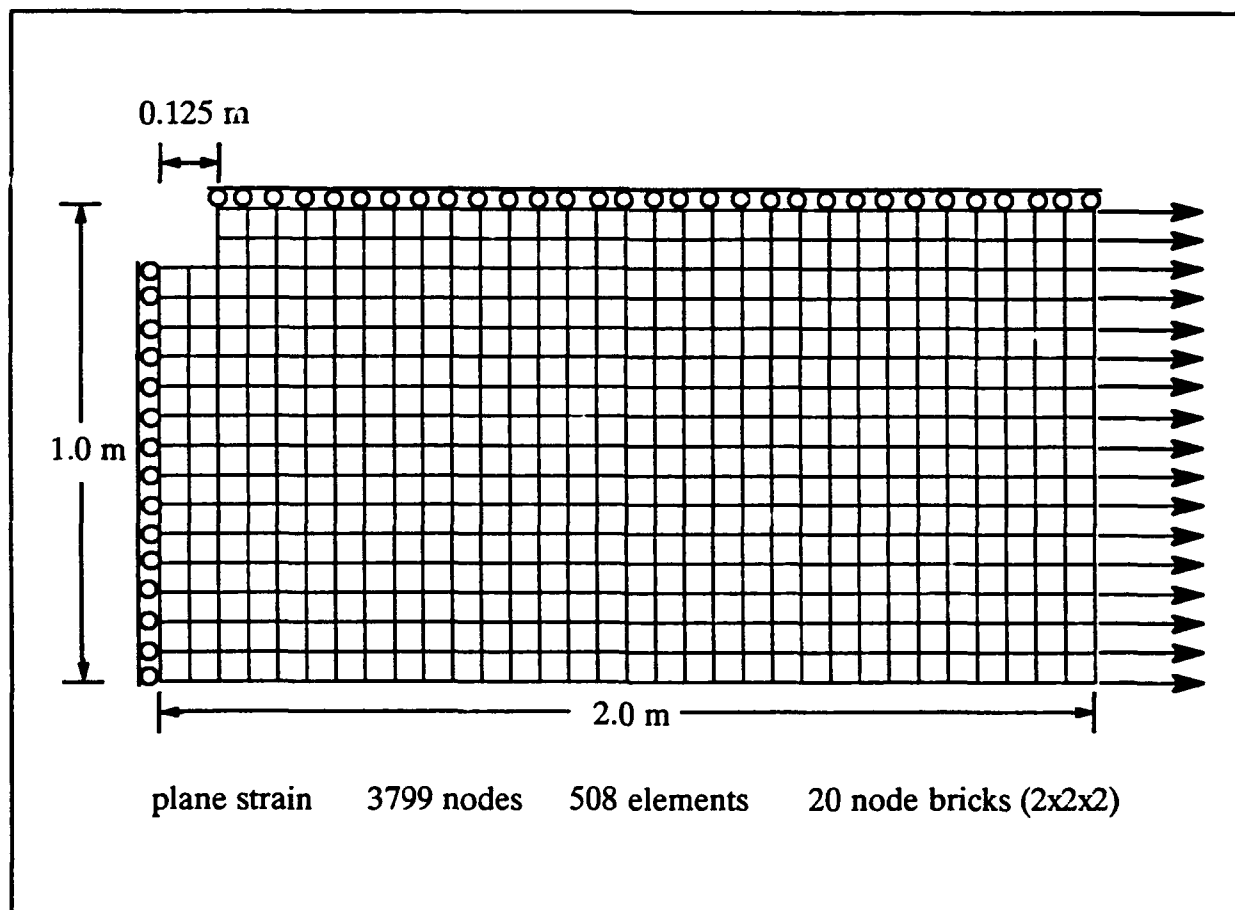


Fig. (3.2-1), mesh for shear band problem

The loading history and material properties are shown in Fig. 3.2-2. The load is increased at a constant rate to a maximum level and then held constant. The maximum load is reached in 10 equal time steps of 0.1 milliseconds, and a total of 100 equal time steps is required to complete the analysis up to a time of 10 milliseconds. Again, β equal to $1/4$ is used.

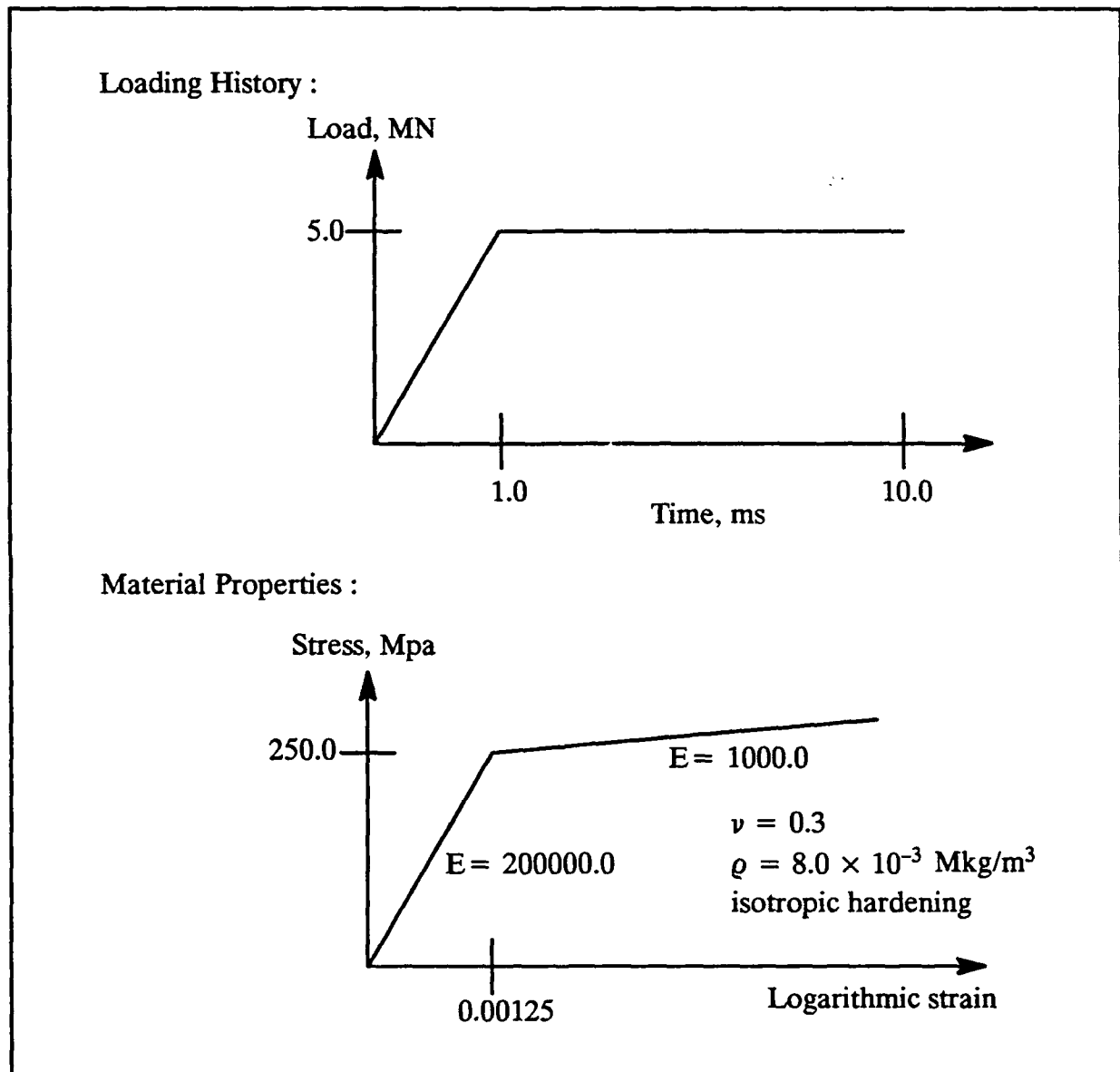


Fig. (3.2-2), problem definition for shear band problem

The material behaviour is bilinear and characterized by isotropic hardening. The maximum load corresponds to a stress of 250 Mpa, the yield stress of the material. The load is not in-

creased beyond this level because gross yielding tends to eradicate the shear band. A lumped mass matrix was employed to model inertial effects.

Plots of the horizontal displacement, velocity, and acceleration at the centerline of the loaded end of the plate and the vertical displacement, velocity, and acceleration at the centerline of the lateral, free edge of the plate are displayed in Figs. 3.2-3 to 3.2-8. Note that a positive vertical displacement at the lateral edge is in toward the center hole. From these plots it can be seen that the motions of the two locations are quite complicated and certainly out of phase with each other. The plate is likely beset by higher mode effects and a twisting motion caused by the formation of the shear band.

The nature of the deformation of the plate is revealed in Fig. 3.2-9. In this figure, fringed contour plots of effective strain are traced over the deformed shape of the plate at four separate times. The deformation is amplified 50 times to clearly delineate the shape. The strains are inelastic and the result of accumulating increments of the rate of deformation tensor as discussed in Section 2.4. Henceforth, reference to effective strain will be understood to allude to these inelastic strains. The four times correspond to the onset of the loading flat top, the initial local maximum of greatest effective strain, and equally spaced times in between. The greatest effective strain occurs near the center of the horizontal edge of the center hole and reaches at various times a maximum of about 0.0315, initially at a time of 0.0025 seconds. From the succession of plots, the formation of a distinct shear band is easily observed, at an angle of about 47 degrees with the vertical centerline of the lateral, free edge.

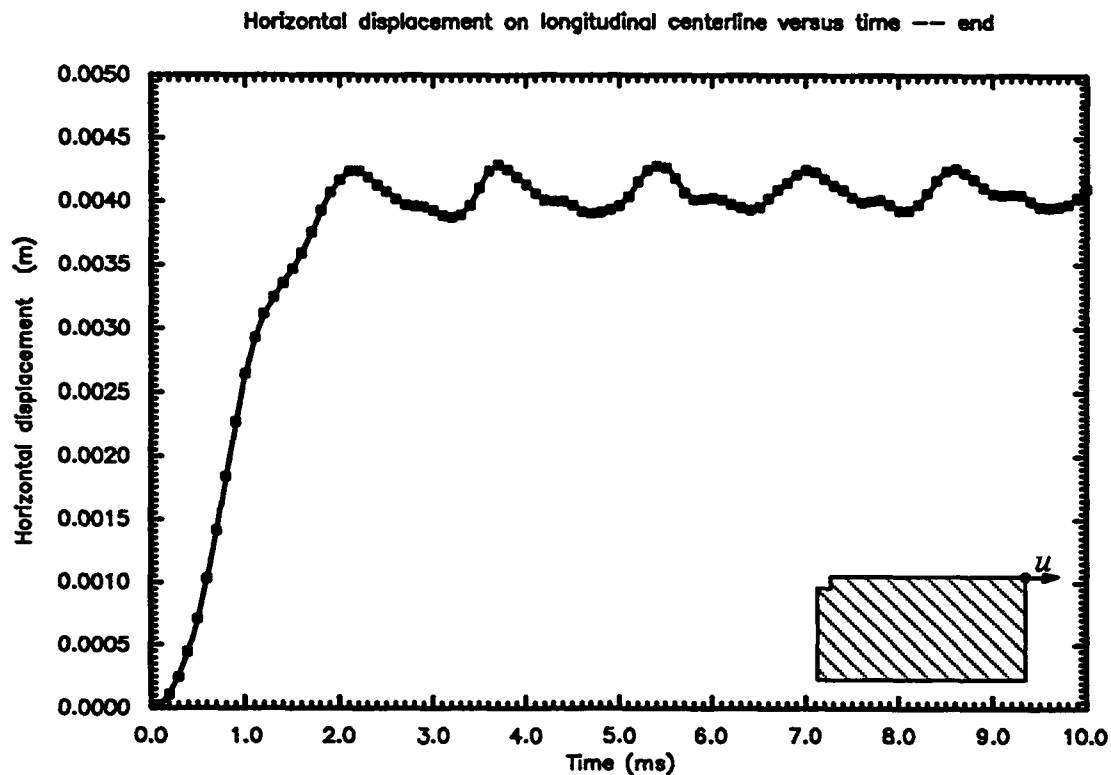


Fig. (3.2-3), horizontal displacement at loaded end, centerline

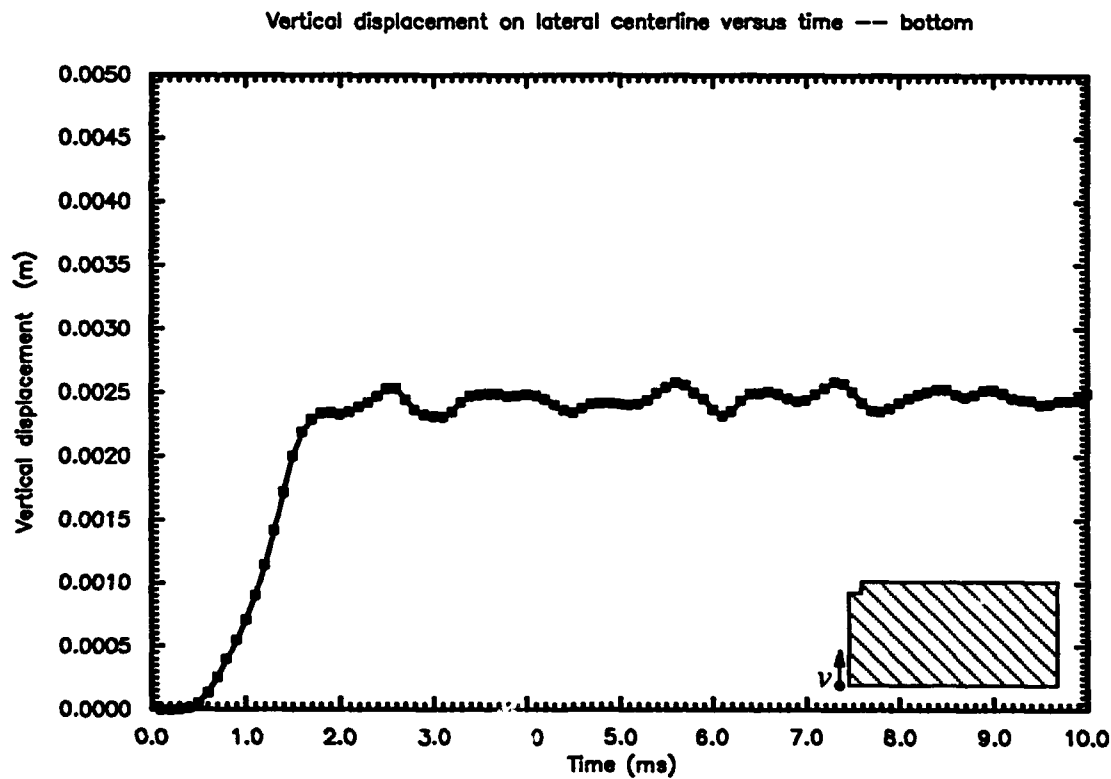


Fig. (3.2-4), vertical displacement on lateral edge, centerline

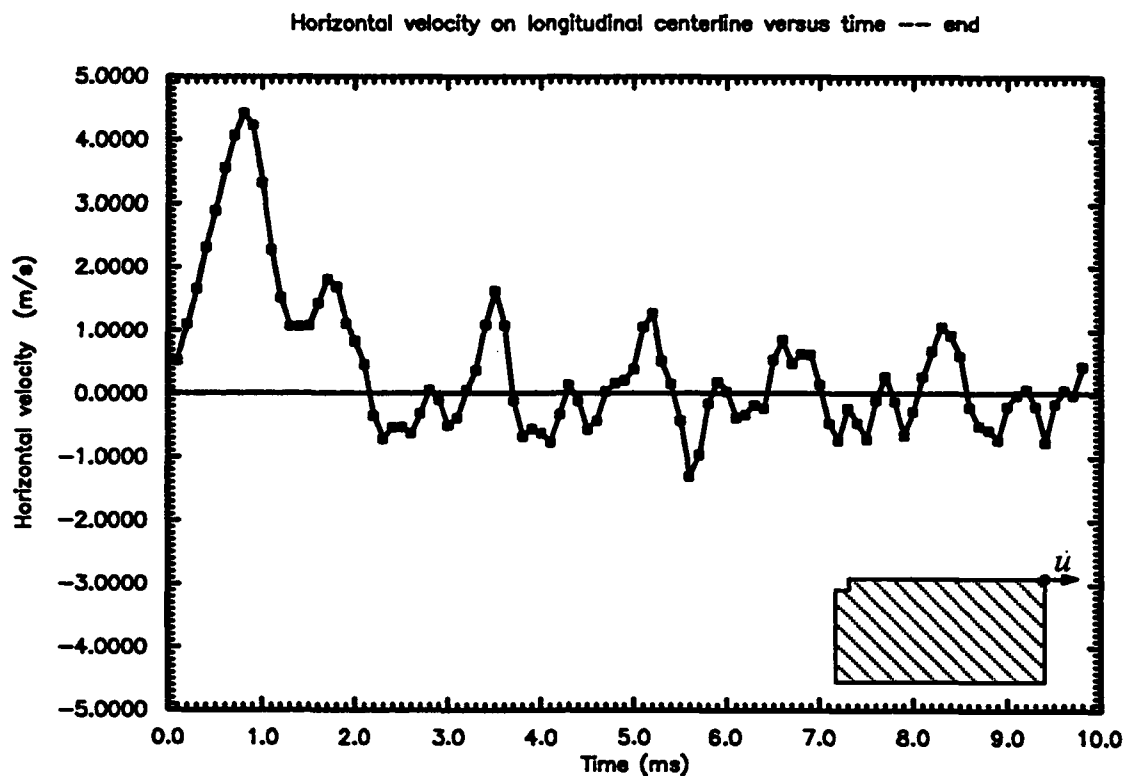


Fig. (3.2-5), horizontal velocity at loaded end, centerline

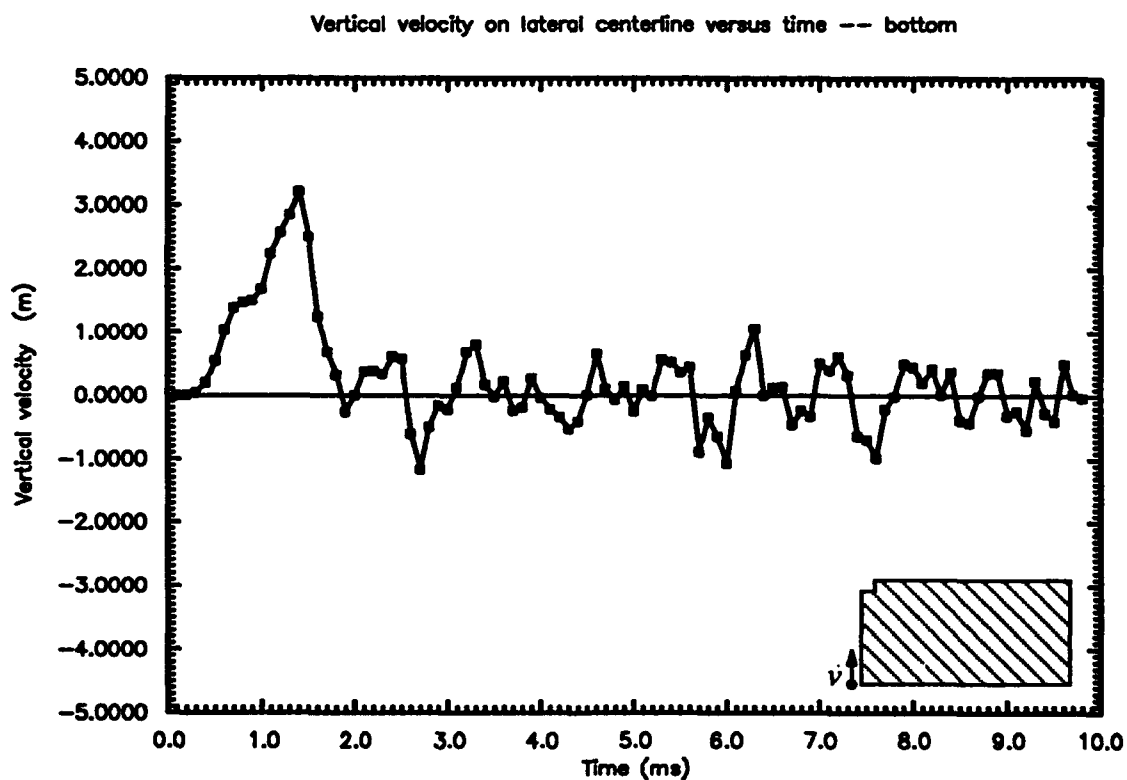


Fig. (3.2-6), vertical velocity on lateral edge, centerline

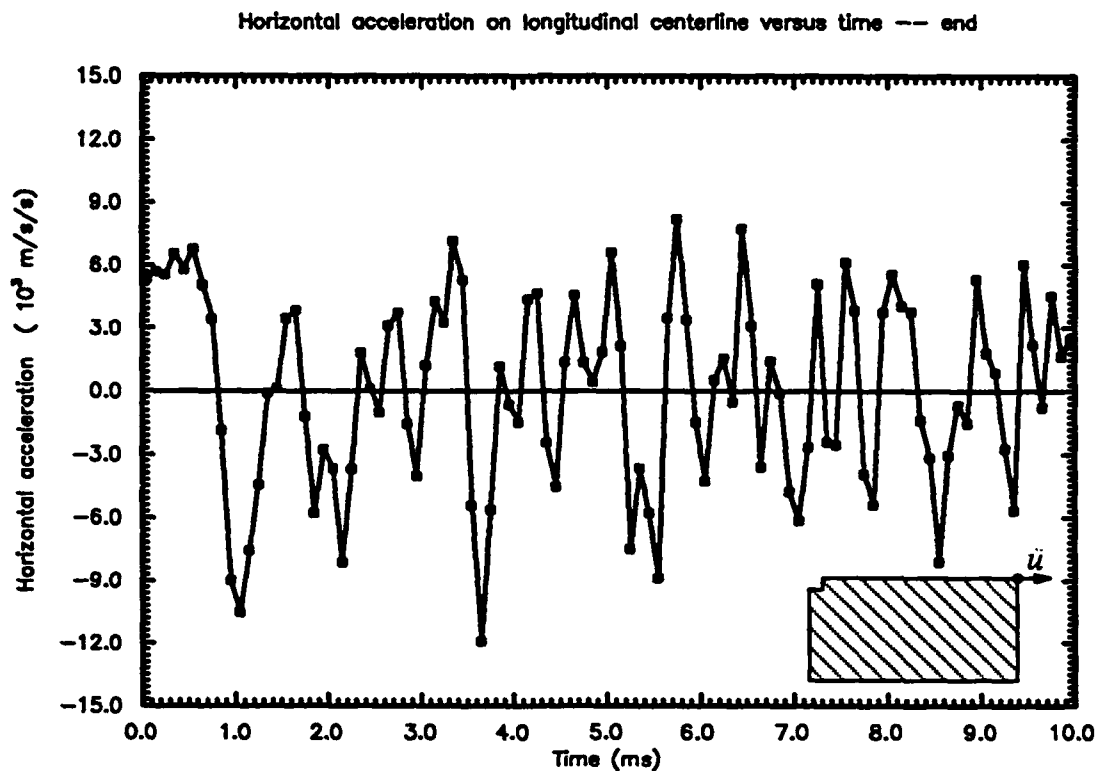


Fig. (3.2-7), horizontal acceleration at loaded end, centerline

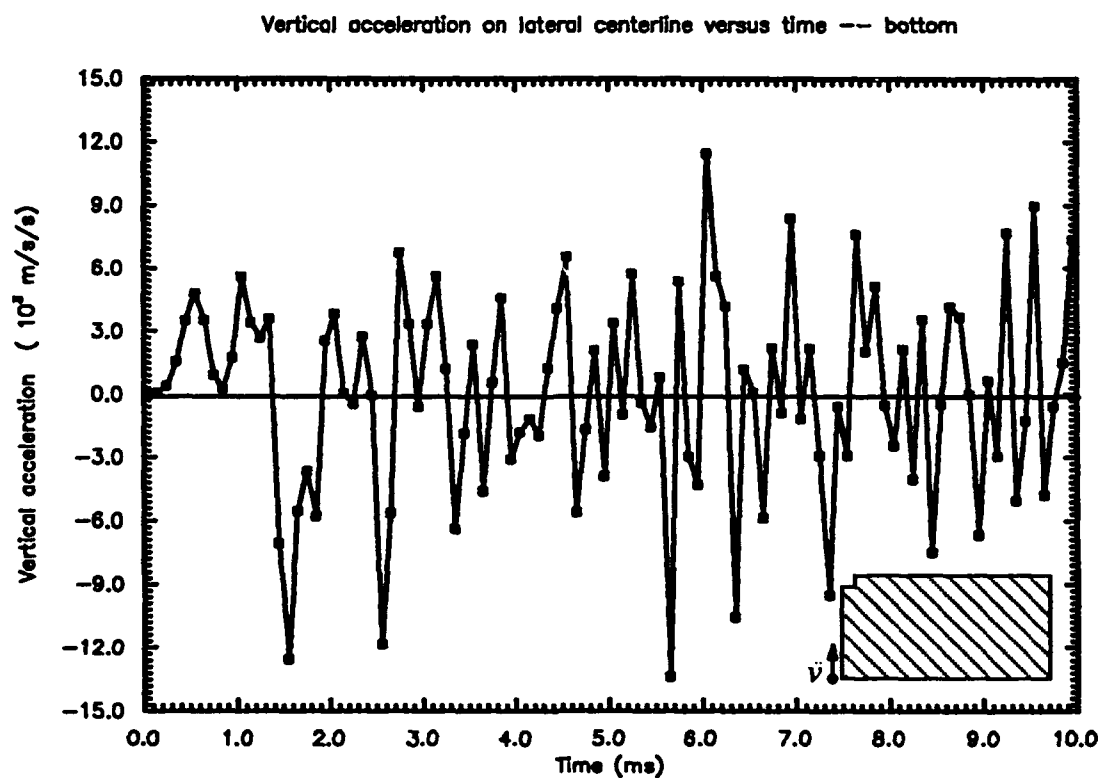


Fig. (3.2-8), vertical acceleration on lateral edge, centerline

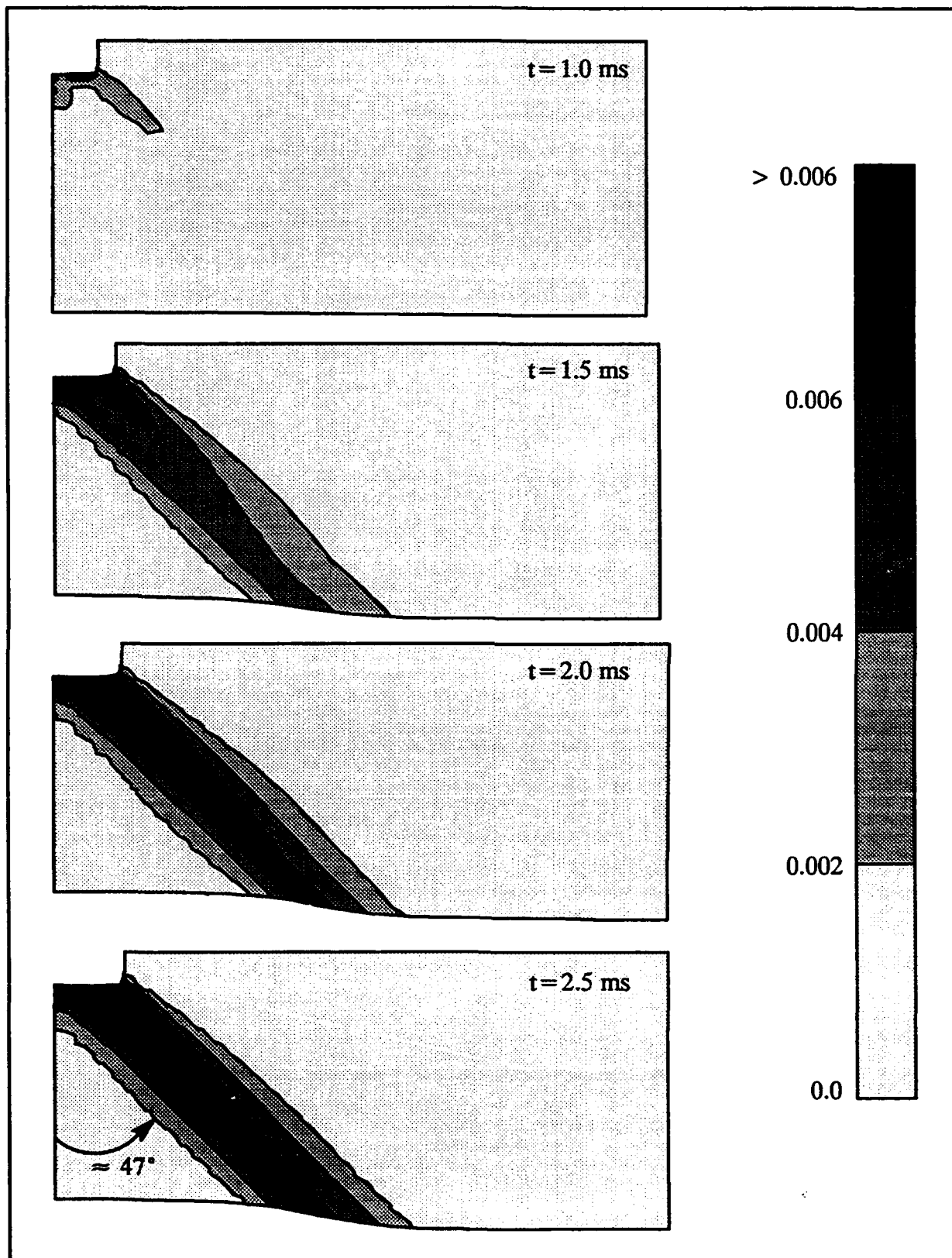


Fig. (3.2-9), effective strain on deformed shape

3.3 3D BEND BAR PROBLEM

The 3D bend bar is a standard three point bend specimen used in experimental fracture mechanics [45]. The mesh, shown in Fig. 3.3-1, is significantly three dimensional. The mesh represents one quarter of a short beam with symmetry boundary conditions on the center boundaries, leaving 9040 active degrees of freedom, and a sawn crack flaw at the center of the longitudinal dimension. The beam rests on simple supports extending the width of the beam near the ends. The load, representing the contact stress produced by a falling weight onto a wedge shaped aluminium absorber [45], acts across the width of the beam on the first two rows of elements, as pictured in Fig. 3.3-1. The crack extends halfway up the height of the beam from the bottom and terminates in a sharp crack tip, which is modeled with collapsed eight noded elements. All elements are integrated with a 2 by 2 by 2 rule.

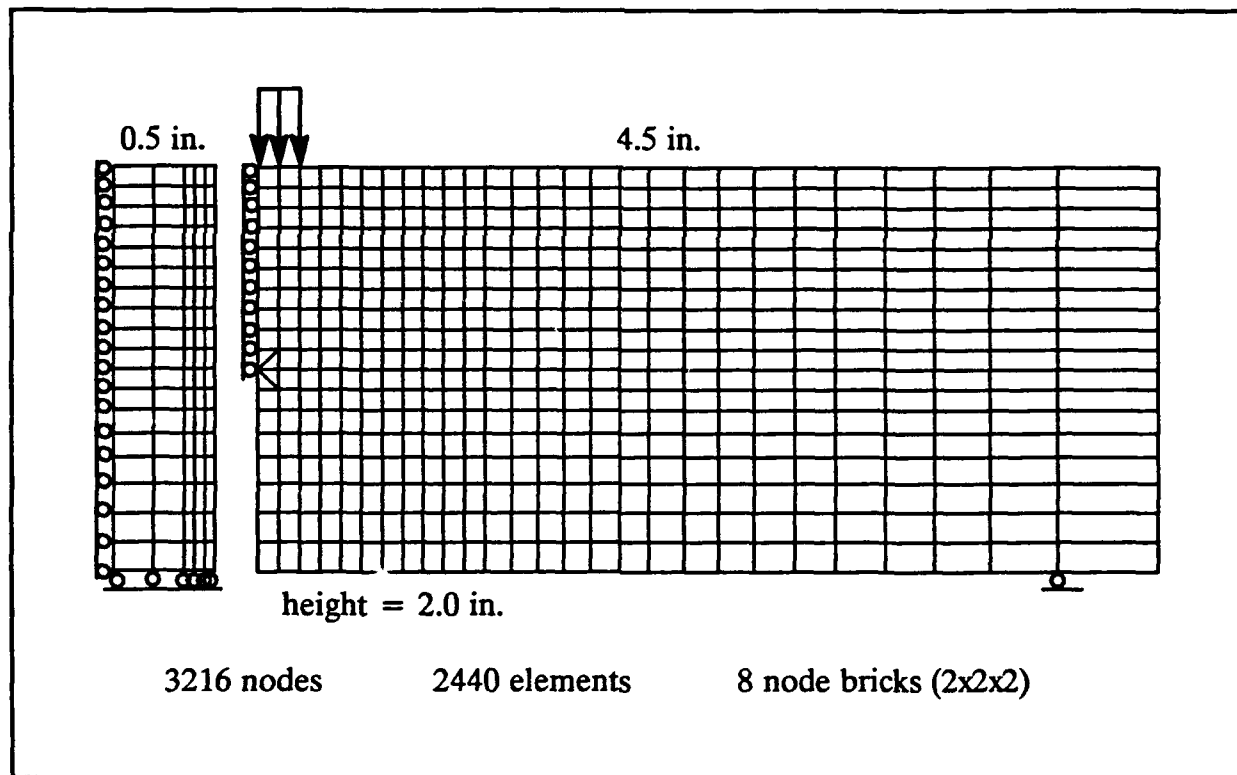


Fig. (3.3-1), mesh for 3D bend bar problem

The loading history and material properties are shown in Fig. 3.3-2. The load is increased linearly to a maximum level and then held constant. The size of the initial time step is 0.0307712 milliseconds and all remaining time steps employ an increment of 0.035 milliseconds, a difference between time increments of approximately 4 microseconds. Noticeable im-

pact-like effects due to the change in time step size after the first step were not observed, although it is difficult to distinguish such effects as the change in time increment occurs at the beginning of the contact loading. The analysis was repeated using equal time increments for all steps, resulting in insignificant differences in displacements, velocities, and accelerations from the original analysis. The maximum load is reached in 10 time steps, and a total of 43 time steps is required to complete the analysis up to a time of 1.5 milliseconds. The material behaviour is bilinear and characterized by kinematic hardening.

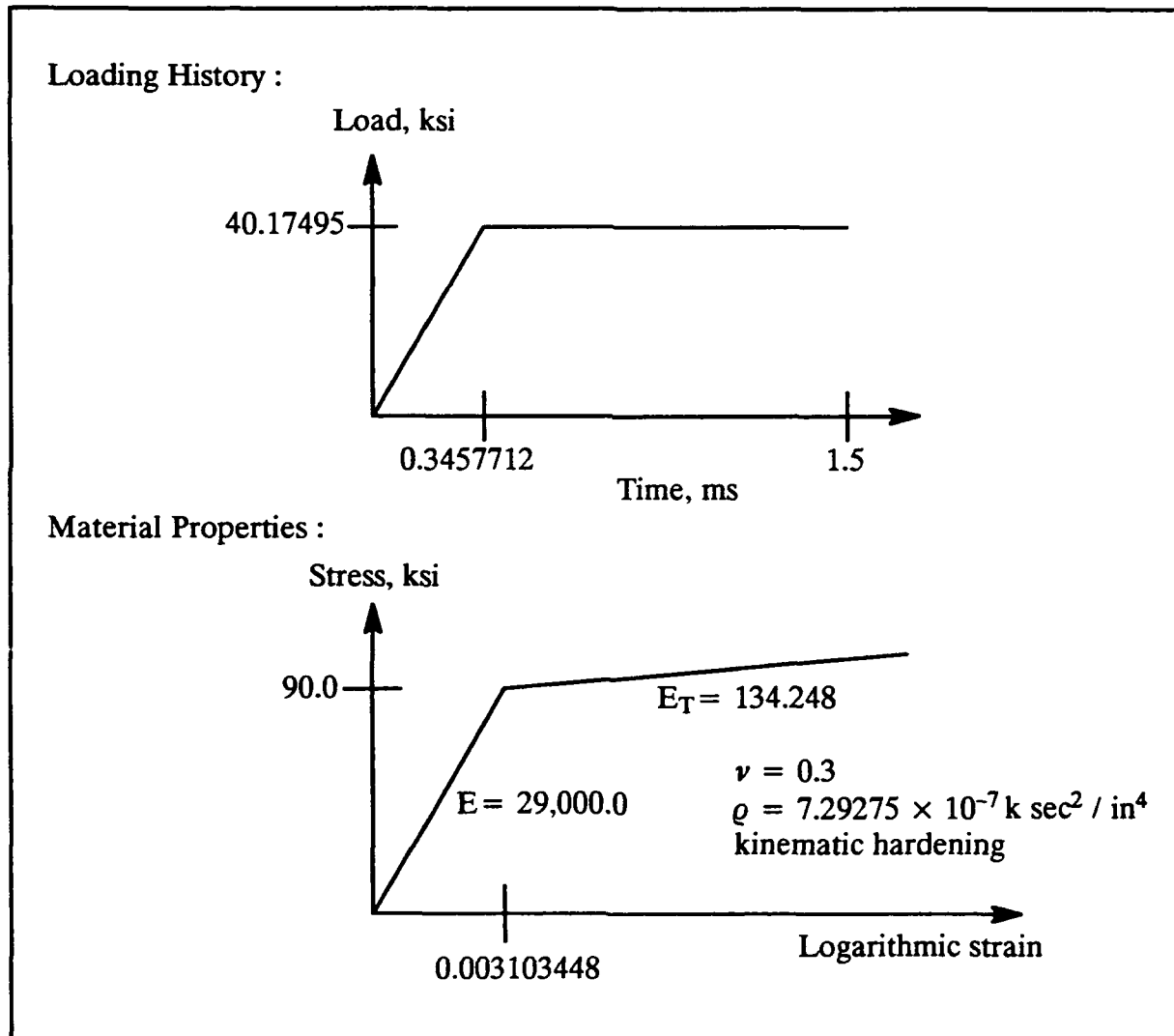


Fig. (3.3-2), problem definition for 3D bend bar problem

A parallel study was performed by Nakamura et al. [57] in which all parameters used were non-dimensionalized and explicit time integration was employed. Unfortunately, insufficient information was provided to infer the appropriate material properties to be used

in this analysis, so that these properties were estimated and the results of Nakamura can be matched in only a qualitative sense. The applied load pattern and time history was developed by Nakamura to accurately represent the nearly constant load application rate and pressure distribution achieved through the proper choice of an impact absorber material [45],[57].

The deformed shape corresponding to the time of the maximum vertical displacement at the longitudinal center of the beam is illustrated in Fig. 3.3-3. A side view close up of the opening of the crack tip region is also presented. The deformed shape is similar to the fundamental mode shape produced by a linear elastic five iteration subspace eigenvalue analysis using POLO-FINITE [70].

The progression of effective strain distribution with time is illustrated in Figs. 3.3-4. The time $t = 0.3457712$ milliseconds corresponds to the distribution at the time when the maximum load is reached, $t = 0.9757712$ milliseconds to the time of maximum vertical displacement at the longitudinal center of the beam, and $t = 0.6607712$ milliseconds to the time midway between the two. The areas of strain concentration are seen to be at the crack tip, towards the free surface, and to a lesser extent beneath the load. A maximum value of effective strain equal to 0.179, attained at the free surface crack tip at $t = 0.9757712$ milliseconds, is approximately 60 times the yield strain. This illustrates the need for a finite strain formulation even though the strains are generally small. The remainder of the beam outside the areas of concentration are virtually strain free, an expected result considering the suddenness of the loading and the simple support. The primary concentration of strain about the free edge crack tip is also an expected, frequently observed phenomenon in this type of problem.

The vertical displacement, velocity, and acceleration at the center of the top surface of the beam are presented in Figs. 3.3-5 to 3.3-7, respectively. In Fig. 3.3-5 the displacement is seen to reach a maximum value considerably larger than the static displacement and then to oscillate after apparently undergoing elastic unload. By measuring the time between peak oscillations, a period of vibration of 0.63 milliseconds is computed. This compares favorably with the fundamental mode period of 0.621034 milliseconds calculated by the eigenvalue analysis, indicating that the oscillations correspond to an elastic free vibration in the fundamental mode. The velocity plot, Fig. 3.3-6, appears to closely correspond to the slope of the displacement plot, as of course it should. The acceleration plot of Fig. 3.3-7 follows the slope of the velocity plot but is quite jerky in comparison with the other plots. This is understandable in light of the fact that the assumption of a constant average acceleration during a time step drives the numerical integration of the equations of motion, and unless there are a

large number of time steps the plot of acceleration is likely to be less smooth than those for displacement or velocity.

The bending moment on the center ligament, or the uncracked area on the center longitudinal cross-section, is depicted in Fig. 3.3-8. Again, an elastic unloading oscillation is observed after the moment has settled into a nearly flat linear increase with time. Up to the point of elastic unload, the bending moment curve resembles that obtained in reference [57]. However, the latter curve does not exhibit elastic unloading until some time after approximately 1.2 milliseconds, whereas Fig. 3.3-8 shows unloading at approximately 0.976 milliseconds. This discrepancy could very well be caused by locking of the elements in the vicinity of the crack tip, as full 2 by 2 by 2 integration was employed in this analysis. The discrepancy may also result from the uncertainty in material properties alluded to earlier.

Although not present in a static analysis of the 3D bend bar problem, a small transient axial force acts on the center ligament and alternates between tension and compression. This force, plotted in Fig. 3.3-9, must be accounted for when computing the ligament bending moment. Specifically, the bending moment is computed as the couple of equal tensile and compression forces on the center ligament with the net axial force neglected. Fig. 3.3-9 is included to demonstrate the existence and relatively small extent of the net axial force.

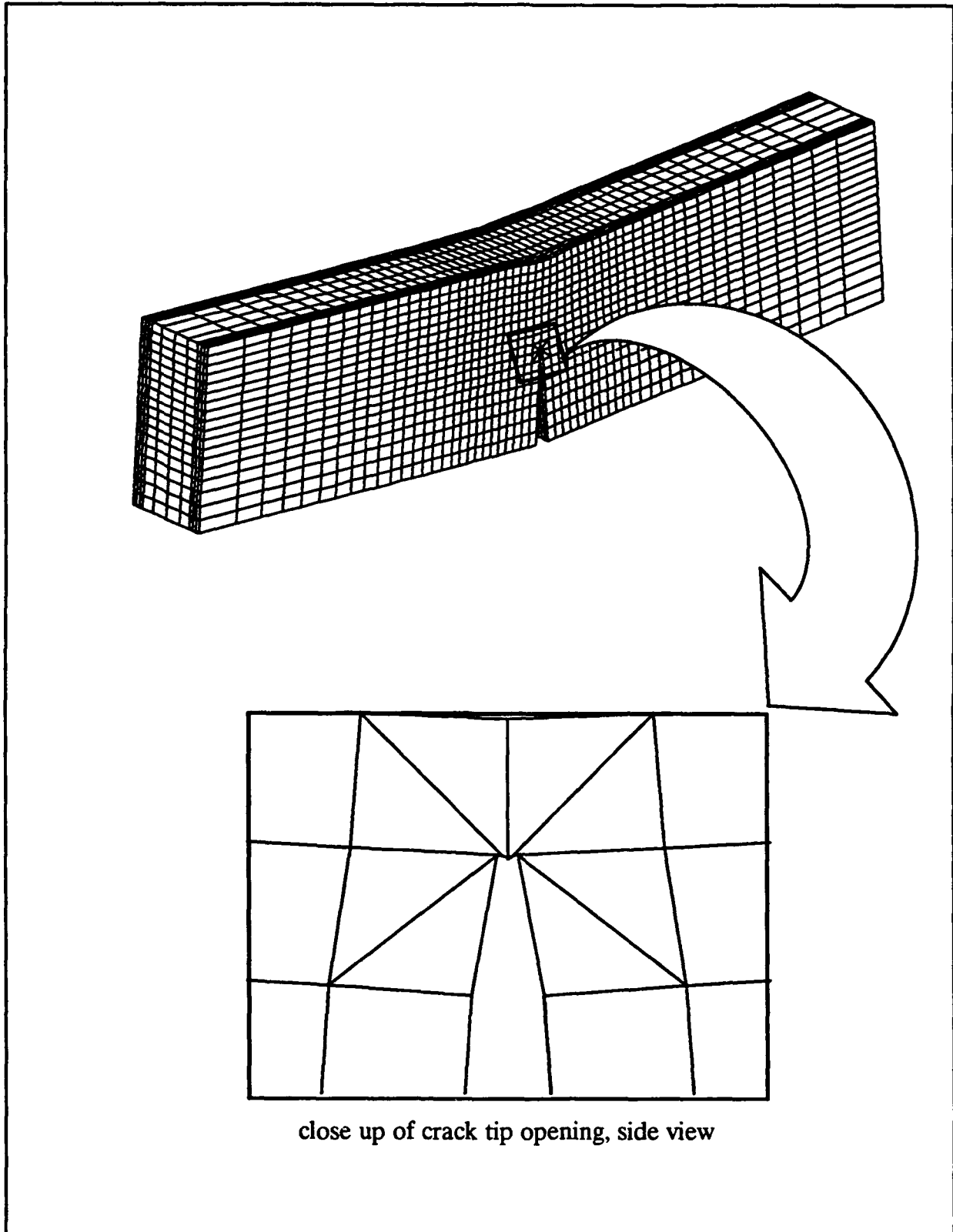


Fig. (3.3-3), deformed shape at maximum vertical deflection

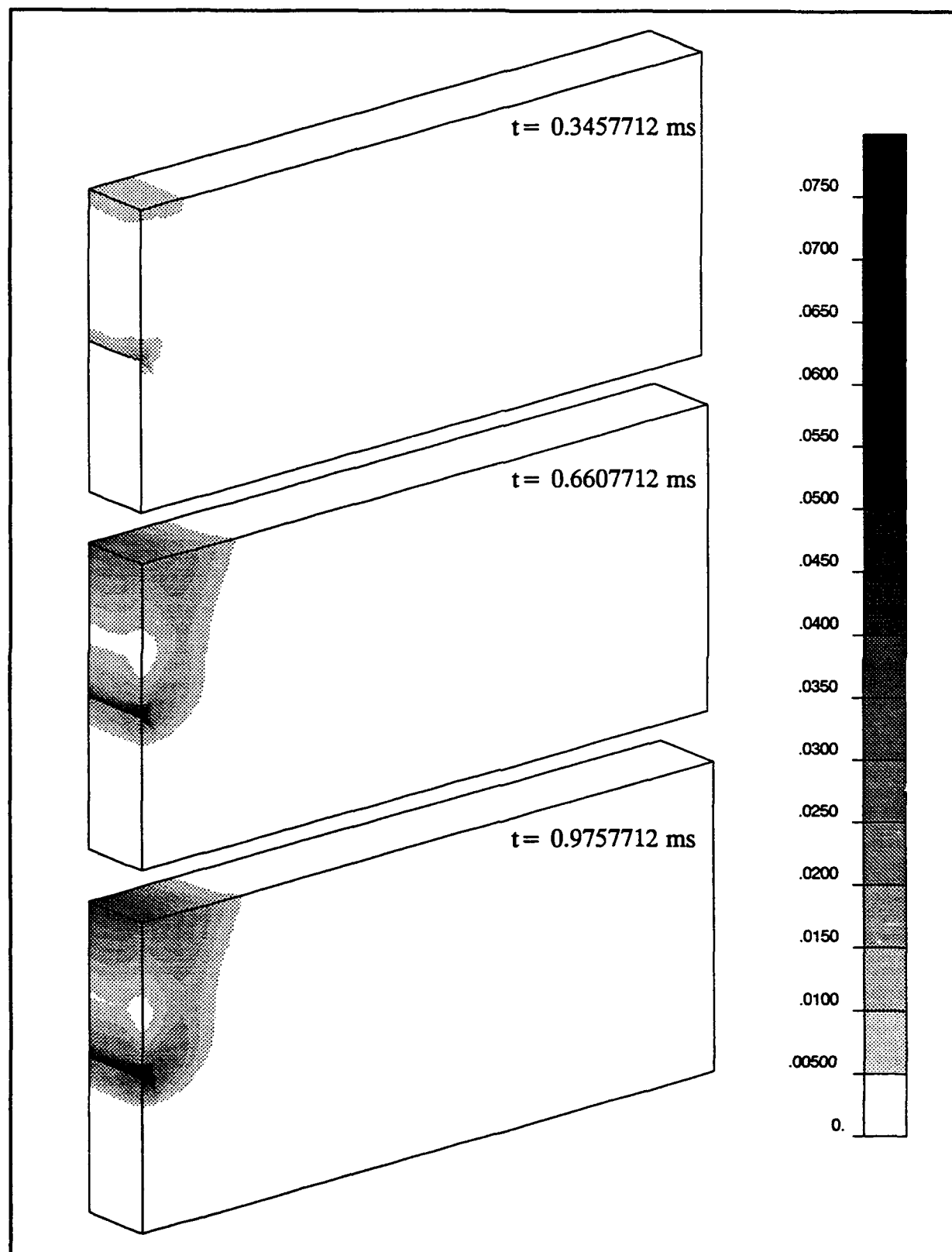


Fig. (3.3-4), progression of effective strain

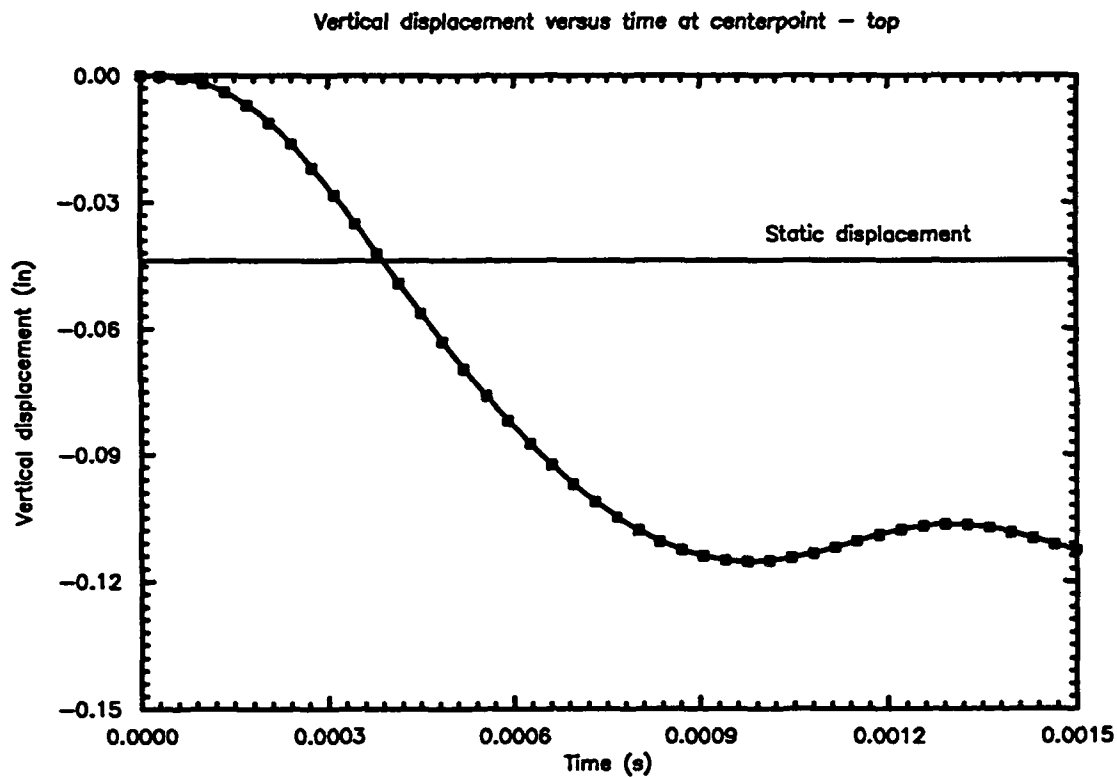


Fig. (3.3-5), vertical displacement at center of bar, under load

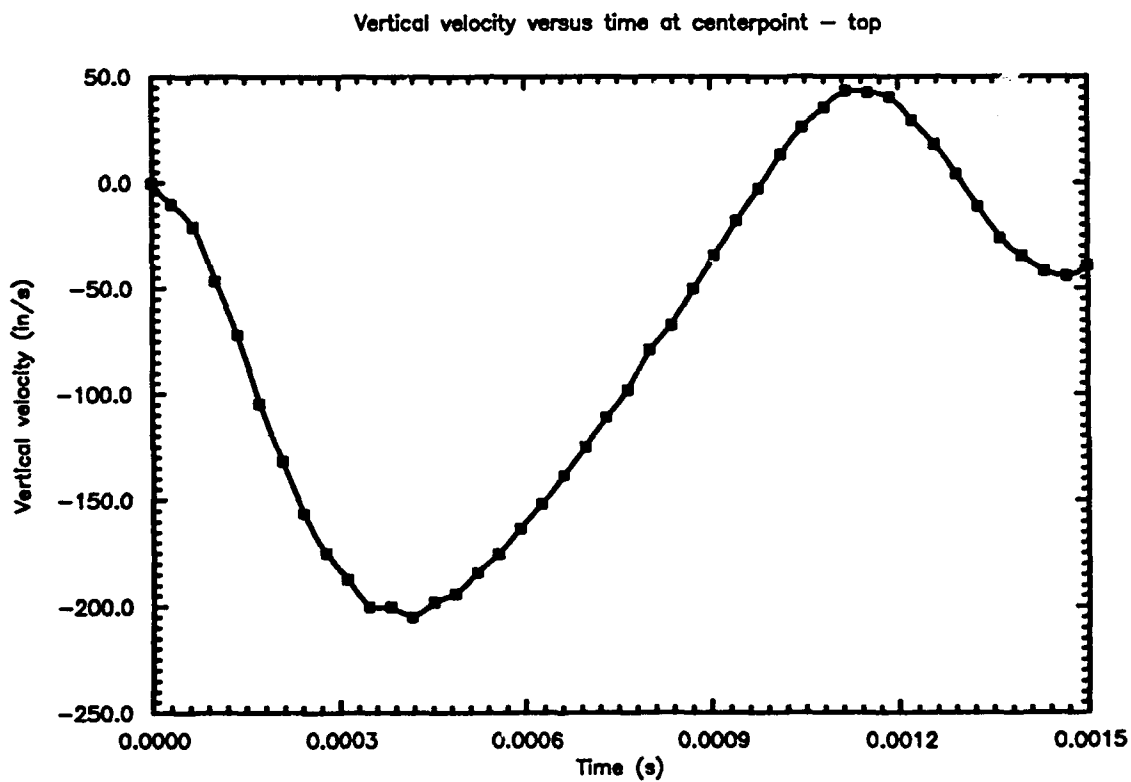


Fig. (3.3-6), vertical velocity at center of bar, under load

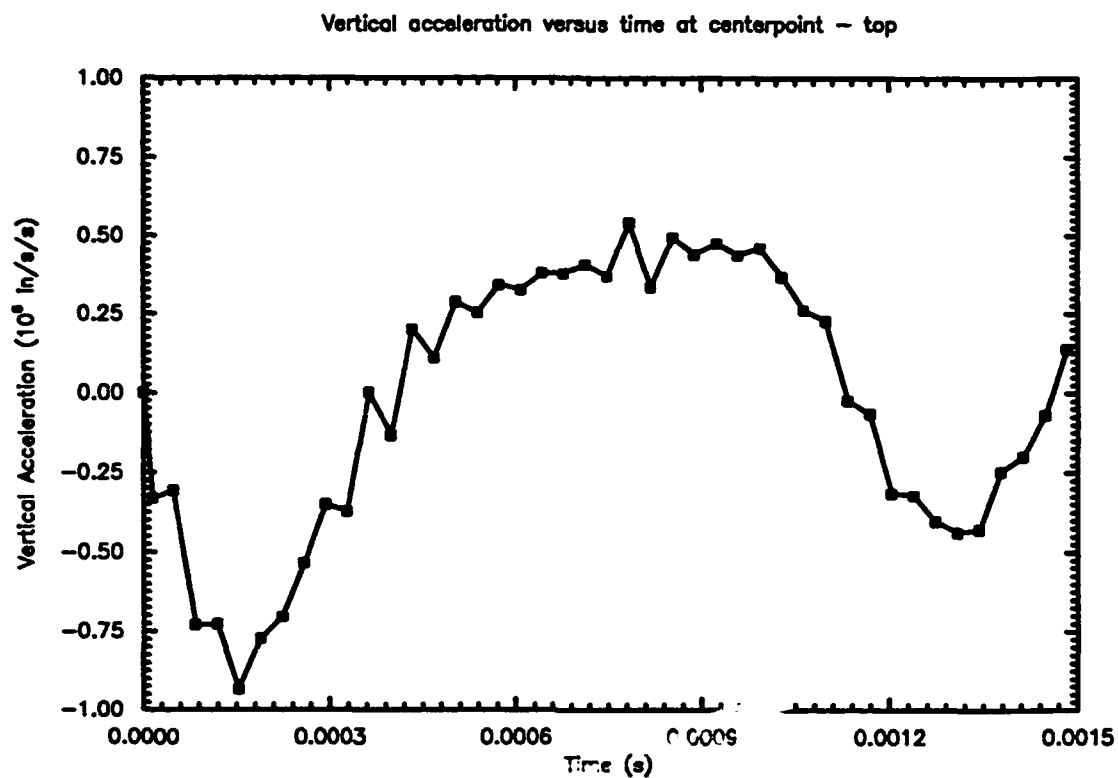


Fig. (3.3-7), vertical acceleration at center of bar, under load

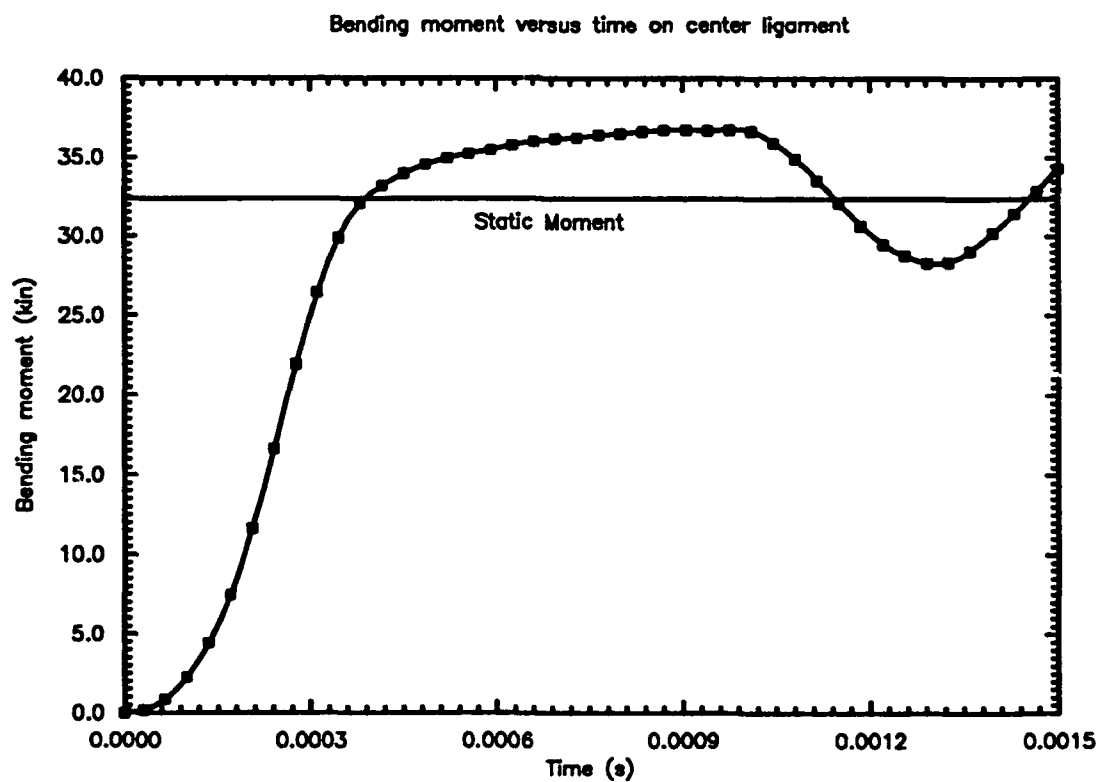


Fig. (3.3-8), bending moment on center ligament

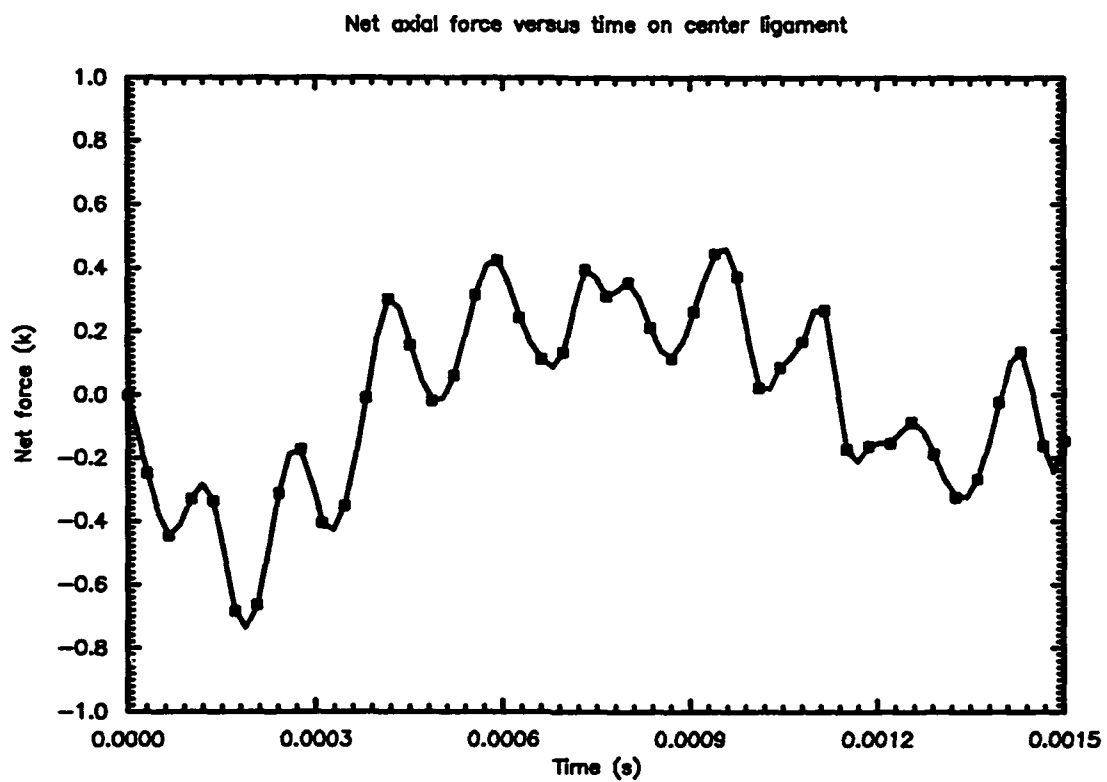


Fig. (3.3-9), net axial force on center ligament

3.4 CYLINDRICAL SHELL PROBLEM

The cylindrical shell problem is described by a mesh incorporating twenty node bricks, a curved geometry, and a significant three dimensional aspect. For these attributes the cylindrical shell problem was solved in order to assess the element computation and linear solution algorithms. The mesh, described in Fig. 3.4-1, consists of one quarter of a moderately thick cylindrical shell fixed at the right end. Symmetry boundary conditions exist along the $z = 0.0$ and $y = 0.0$ faces, generating 11,168 active degrees of freedom. The ratio of thickness

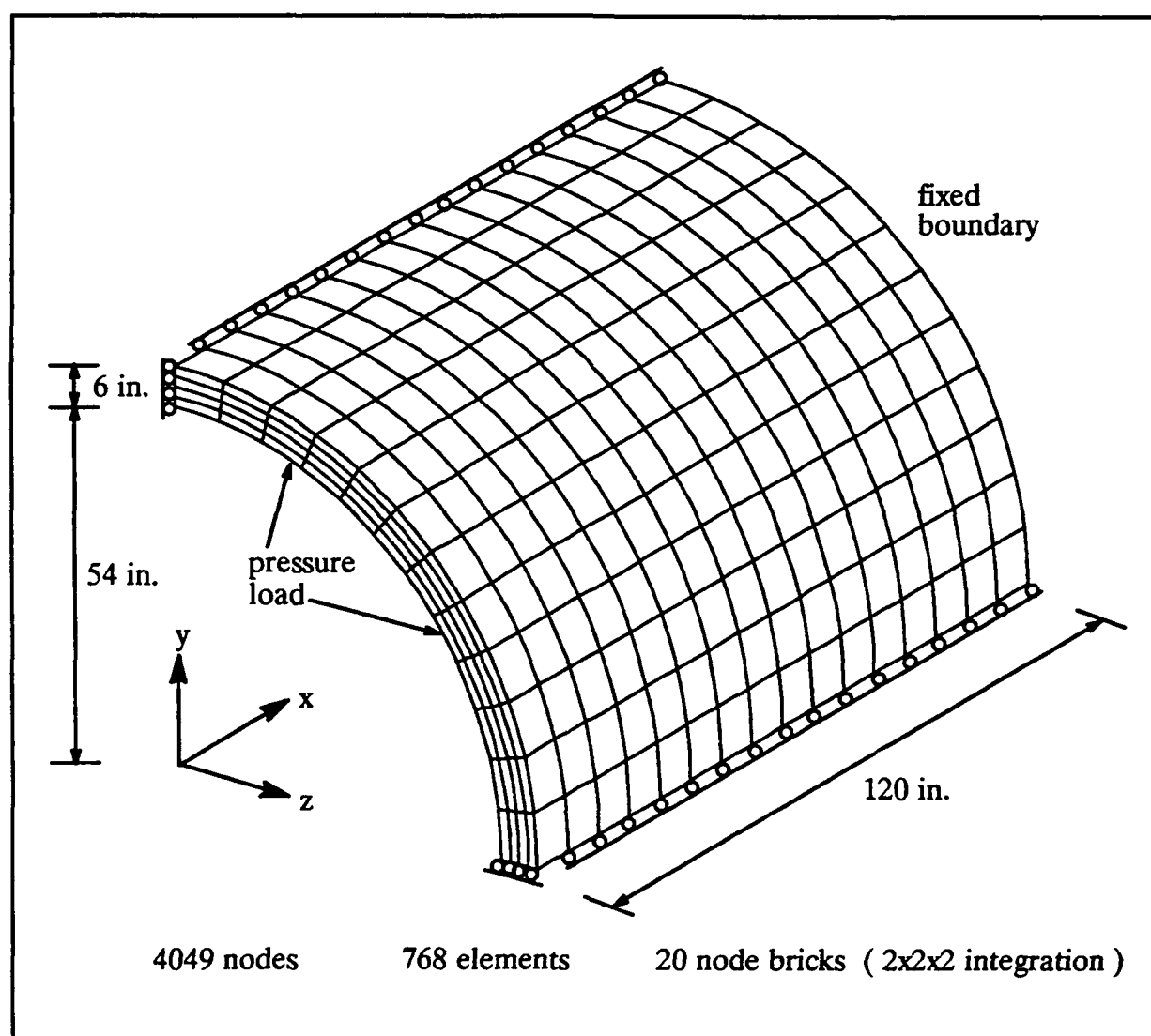


Fig. (3.4-1), mesh for cylindrical shell problem

to outside radius is 0.1. A 15 inch band of internal pressure load, illustrated in Fig. 3.4-2, is applied to the first two rows of elements at the left, free end of the cylinder. All elements employ reduced 2 by 2 by 2 integration.

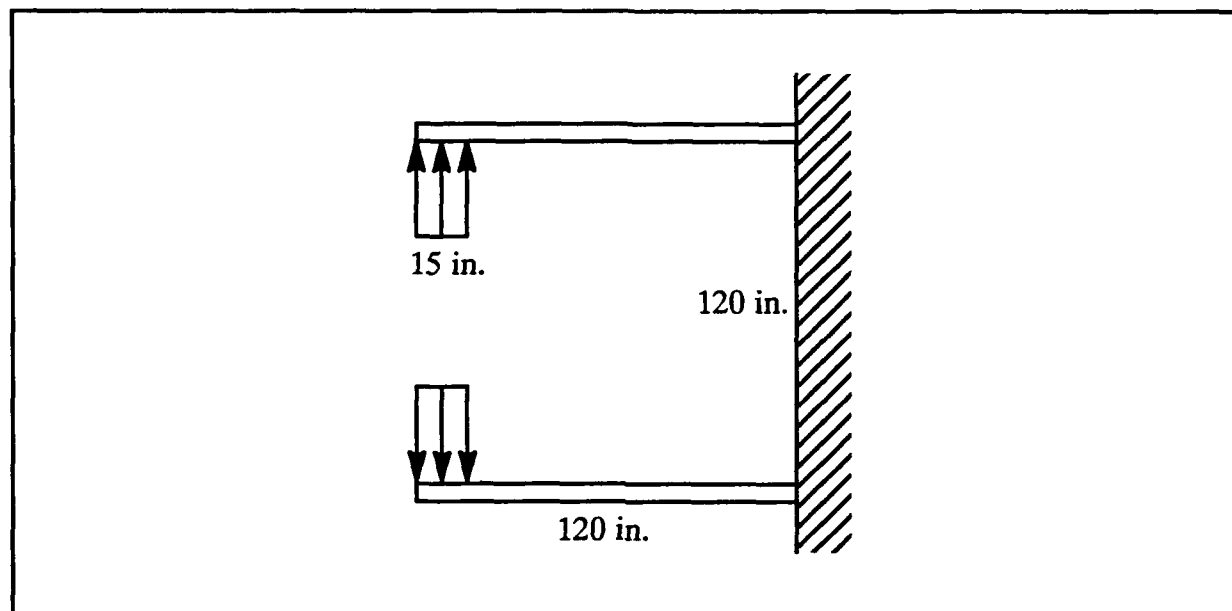


Fig. (3.4-2), pressure load at free end of cylinder

The loading history and material properties are shown in Fig. 3.4-3. The internal pressure is increased linearly to a maximum level and then reduced in similar fashion to zero. This spike loading is intended to represent explosive pressure. The maximum pressure is reached in 10 equal time steps of 0.025 milliseconds, and a total of 100 equal time steps is required to complete the analysis up to a time of 2.5 milliseconds. Throughout the analysis, the material remains in the elastic range, obviating the need for specifying plasticity parameters.

The deformed shapes for $t = 0.675$, 1.55, and 2.45 milliseconds are given in Fig. 3.4-4, along with analogous cantilever beam shapes. These times correspond to the maximum vertical displacements both inward and outward of the free end. For $t = 0.675$ ms, the shape indicates that the deformation is concentrated near the free end and that the end disturbance has not yet affected the rest of the cylinder. For $t = 1.55$ ms, the time pertaining to the maximum inward displacement, the disturbance has propagated over the entire cylinder and the resulting shape resembles the second mode of a cantilever beam. By the time $t = 2.45$ ms is reached, the free end is once again displaced outward, and again the shape is akin to the aforementioned second mode. It seems that the cylinder vibration is dominated by this shape.

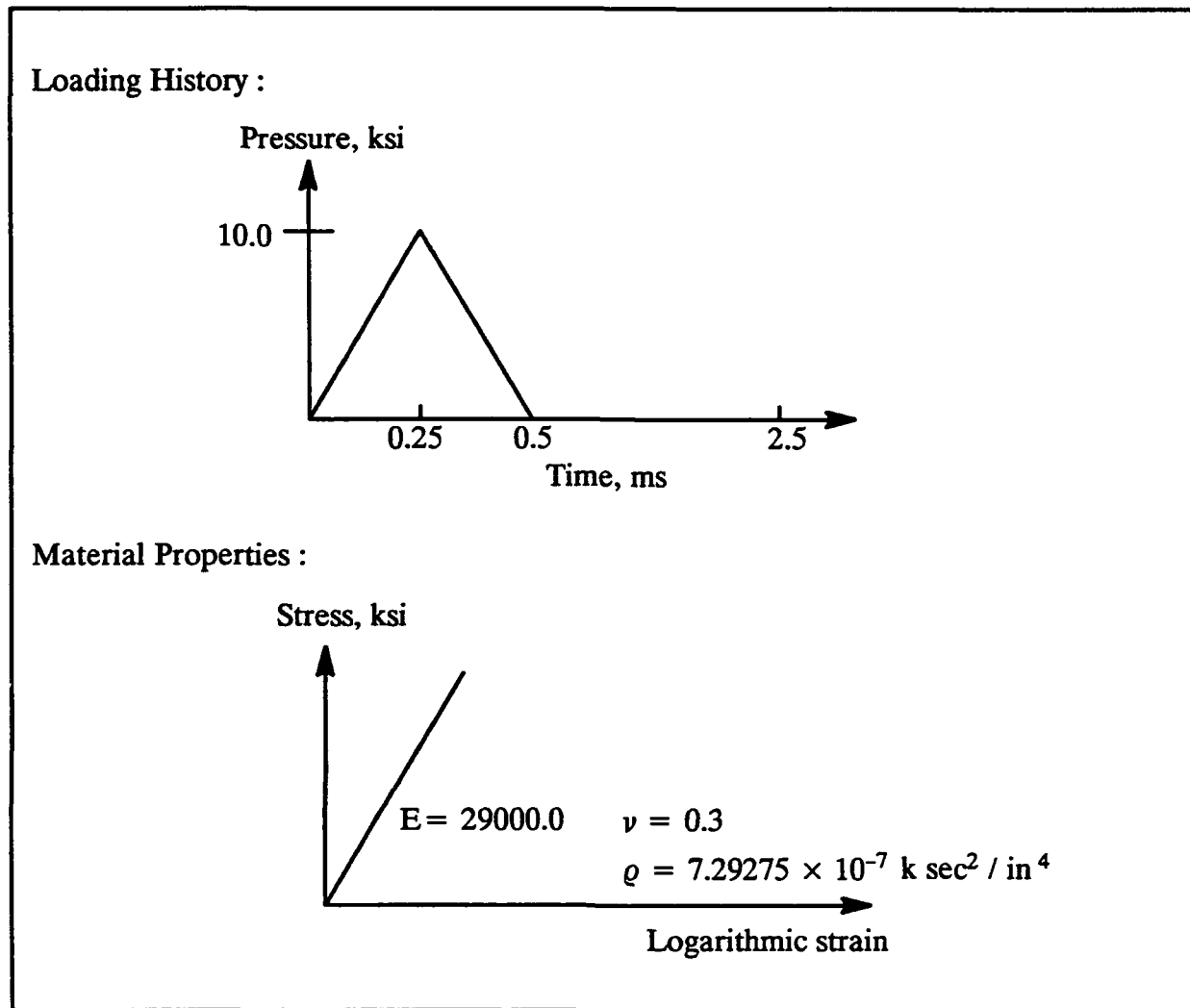


Fig. (3.4-3), problem definition for cylindrical shell problem

In Figs. 3.4-5 to 3.4-7 the displacement, velocity, and acceleration of the midsurface sampled at various points along the $z = 0.0$ face are displayed. The sample points are the quarter points in the x direction, including the free end. Due to symmetry, results along the $z = 0.0$ face should be identical to those of any other longitudinal cross-section. The displacement plot, Fig 3.4-5, seems to verify that the deformed shape is beginning to achieve a synchronous motion in a mode two shape by the cessation of the analysis. From the velocity and acceleration plots, one can clearly see the influence of higher mode shapes, but it seems that the vibration is stabilizing.

Plots of effective Cauchy stress versus time are given in Figs. 3.4-8 to 3.4-12. These plots correspond to the inner, middle, and outer through thickness surfaces at various points

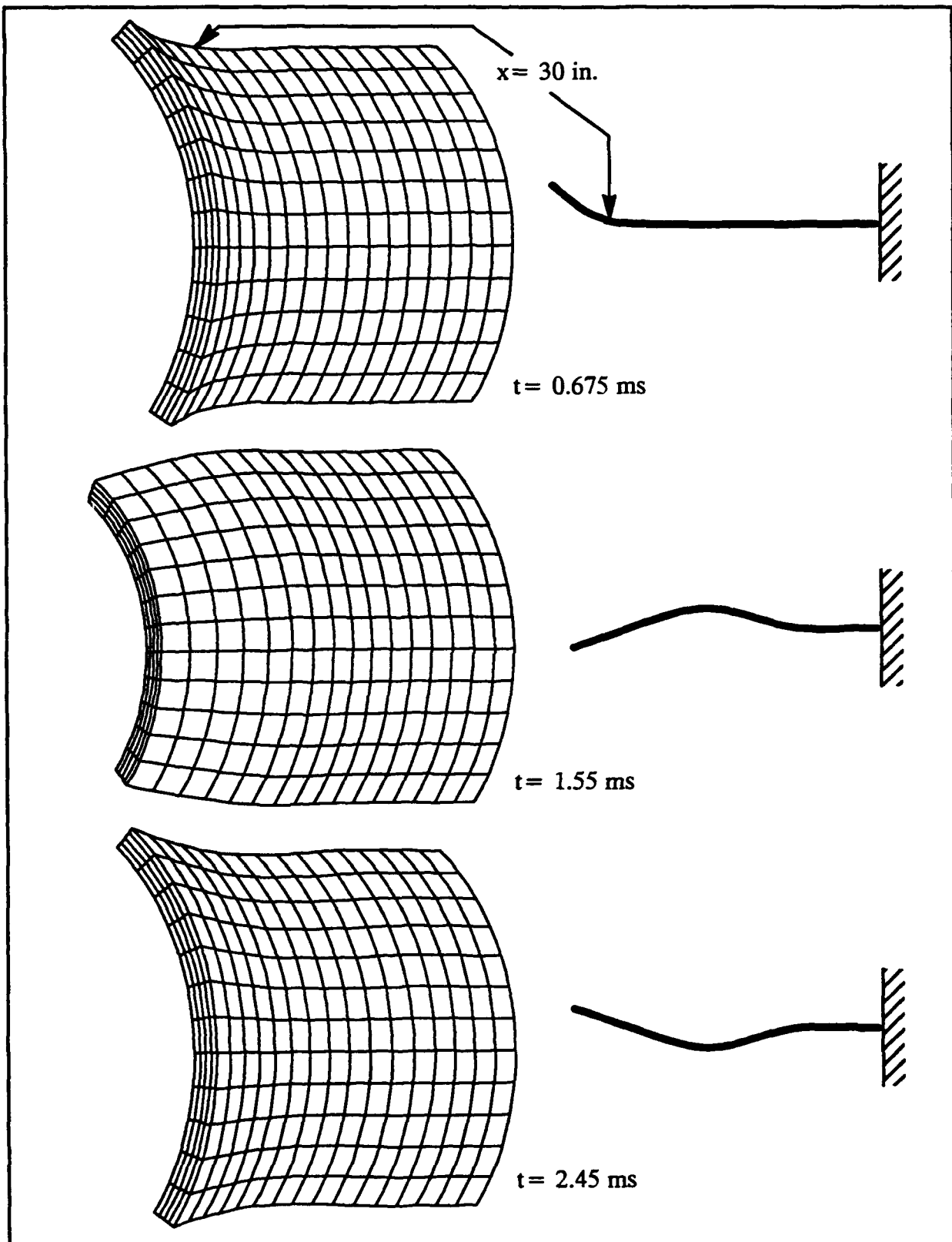


Fig. (3.4-4), deformed shape at various times

along the $z = 0.0$ face. Again, these points are the quarter points in the x direction, including both ends. The first noticeable feature of these plots is the lag time between the initial loading and the first appearance of non-zero effective stress. From the plots, the lag time is approximately 0.15, 0.30, 0.45, and 0.60 milliseconds for $x = 30, 60, 90,$ and 120 inches. These times closely agree with those predicted using the velocity of longitudinal wave propagation for a bar given in Section 3.1. It may be that the initial stress disturbance represents membrane forces and that this membrane disturbance travels with the same velocity as a longitudinal wave in a bar. That the initial disturbance is of a membrane nature is supported by the fact that, initially, the plots for the three surfaces coincide.

The effect of bending in the cylinder is clearly seen in Figs. 3.4-9 to 3.4-12. In Fig. 3.4-8, the plot for the free end at $x = 0.0$ inches, there is little effect of bending due to the free end. The differences between the surfaces at the initial peak seem to consist mostly of the presence of the pressure loading on the inner surface. At later peaks, when there is no applied internal pressure, the plots are more nearly identical. In the other plots, bending is much more significant. The inner and outer surface plots diverge from the middle surface plot when bending at the sample point waxes and rejoin the middle surface plot when the curvature at the sample point wanes. This can be seen by noting the deformed shape and comparing the displacement and effective stress plots at a given time and sample point. For instance, considering Fig. 3.4-9 for sample point $x = 30.0$ inches, between 0.25 and 0.50 milliseconds there is great divergence in the plots for the three surfaces. During this period, Fig. 3.4-5 shows that there is no deflection at the sample point and Fig. 3.4-4 for $t = 0.675$ ms reveals that at $x = 30$ inches the deformed shape implies a significant curvature.

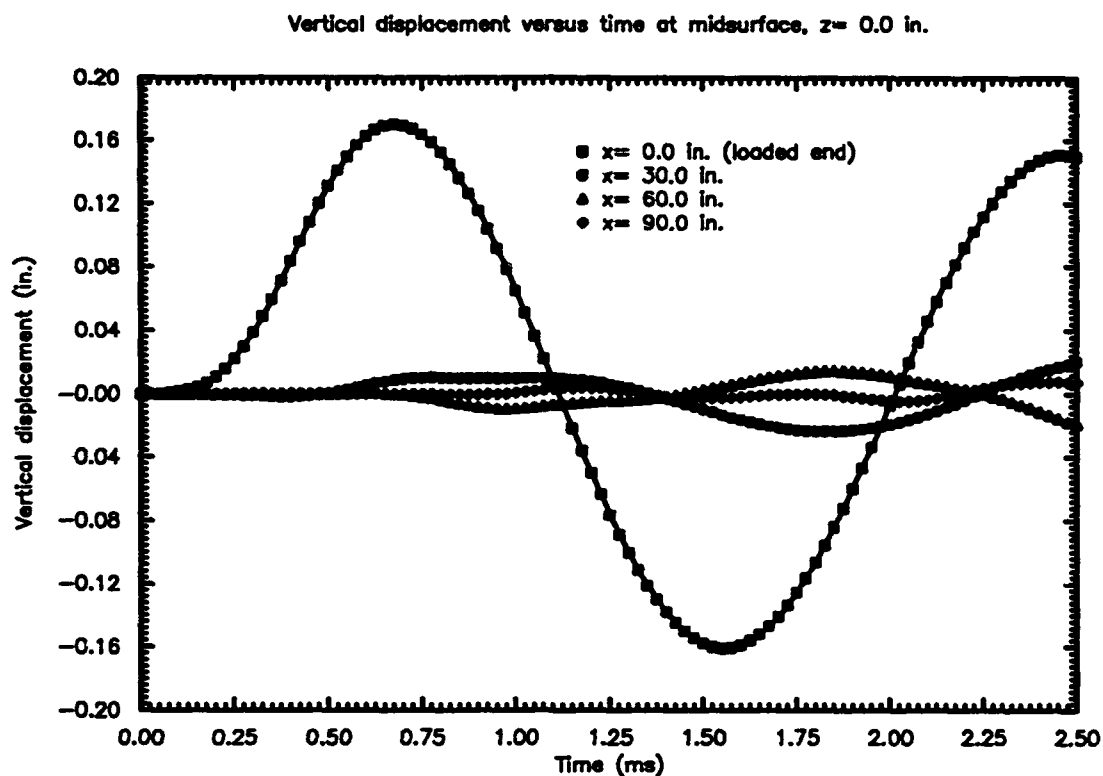


Fig. (3.4-5), displacement vs. time at various points along cylinder

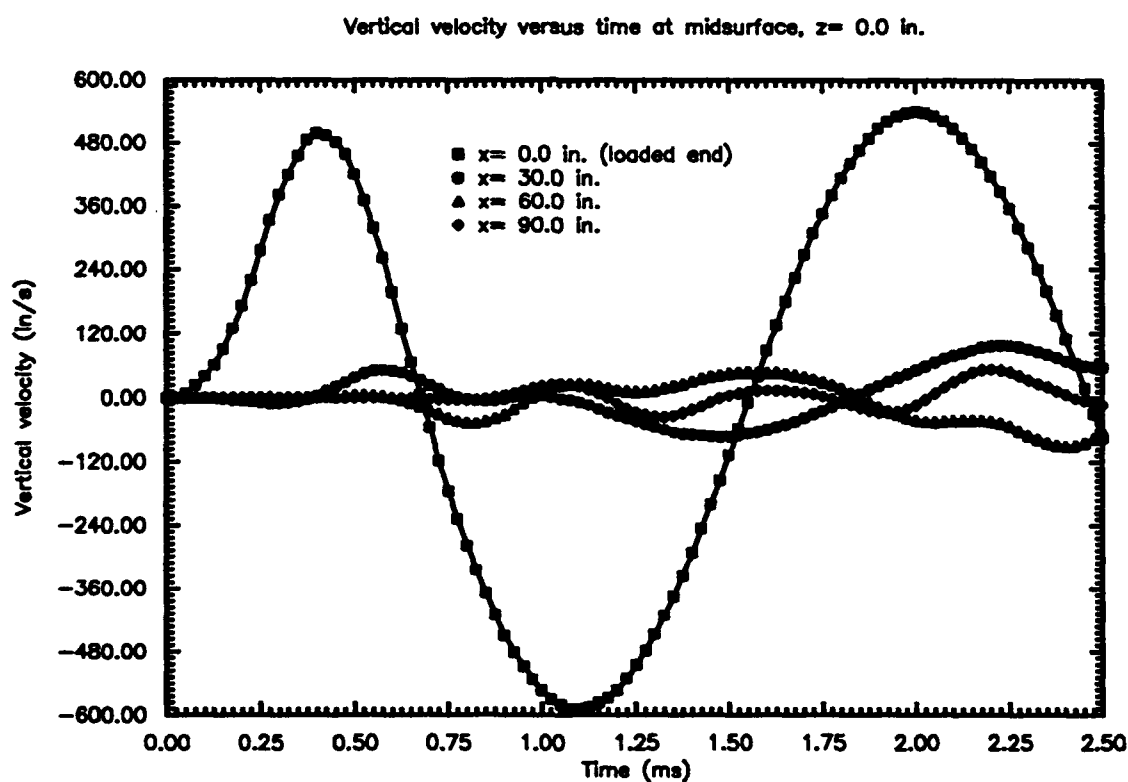


Fig. (3.4-6), velocity vs. time at various points along cylinder

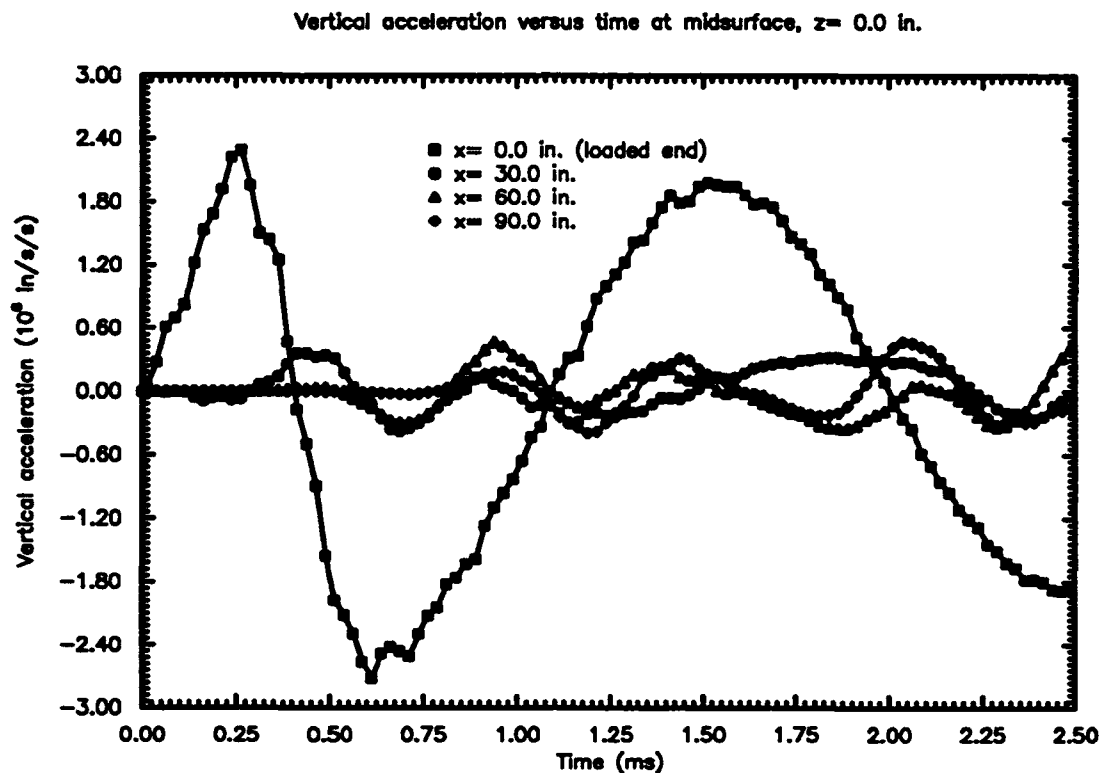


Fig. (3.4-7), acceleration vs. time at various points along cylinder

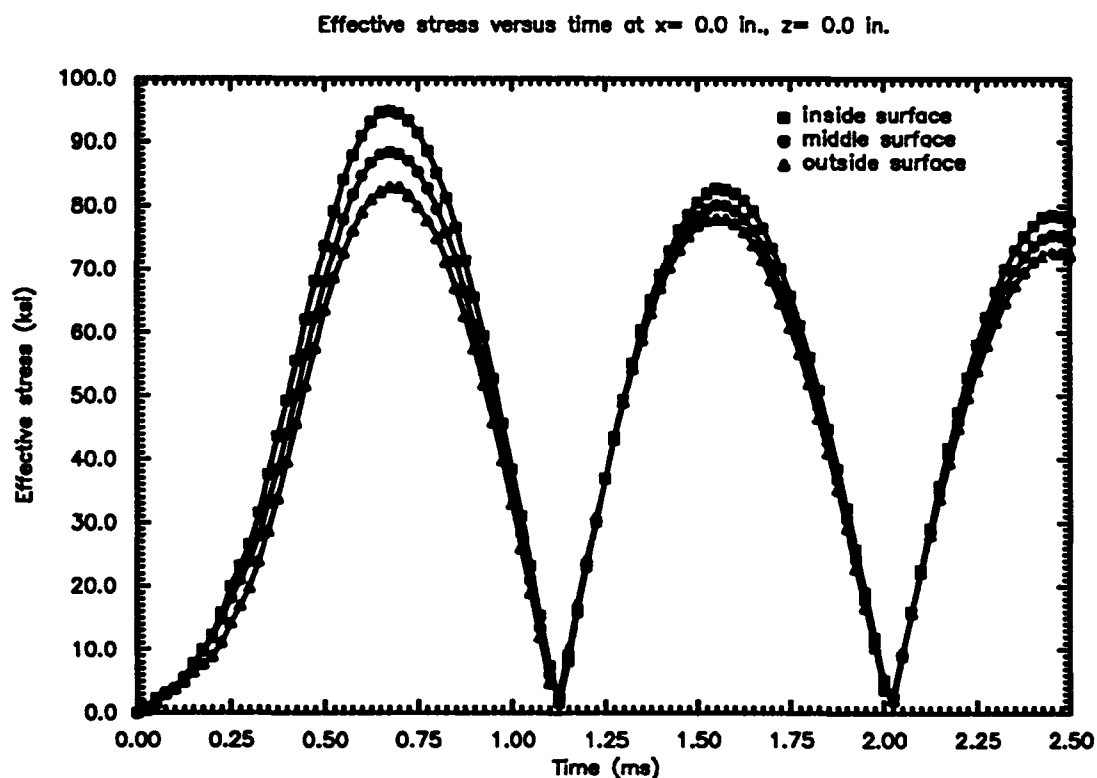


Fig. (3.4-8), effective stress vs. time through thickness of cylinder, $x = 0.0$ inches

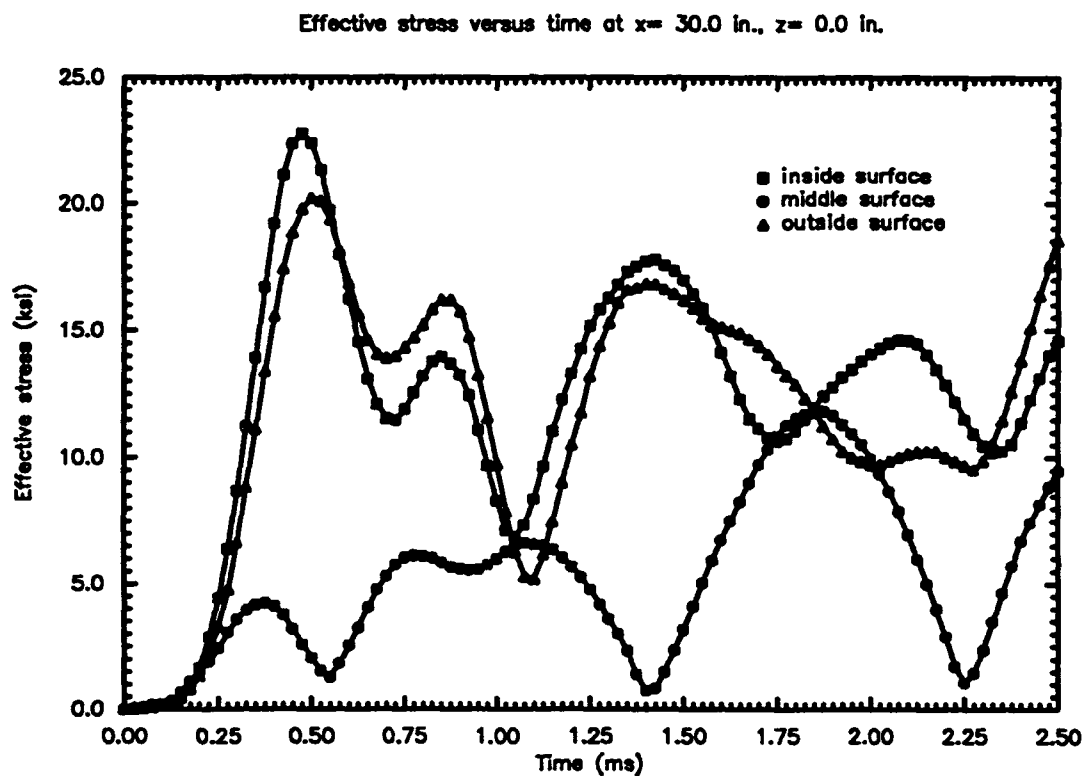


Fig. (3.4-9), effective stress vs. time through thickness of cylinder, $x = 30.0$ inches

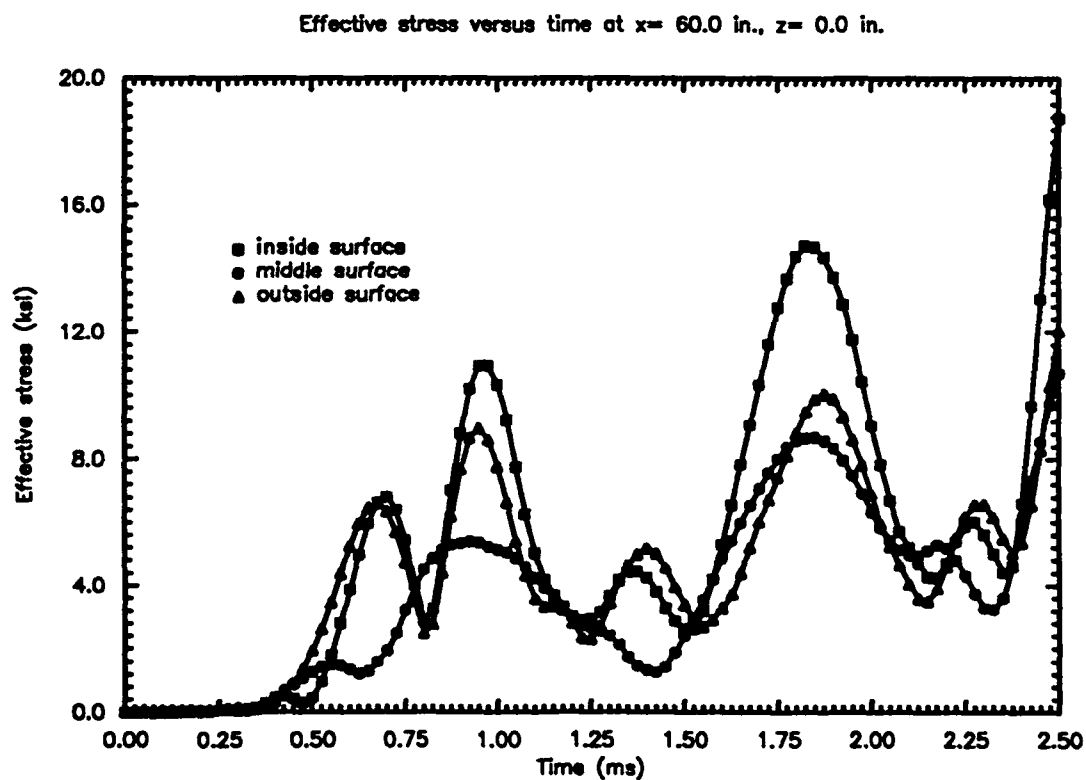


Fig. (3.4-10), effective stress vs. time through thickness of cylinder, $x = 60.0$ inches

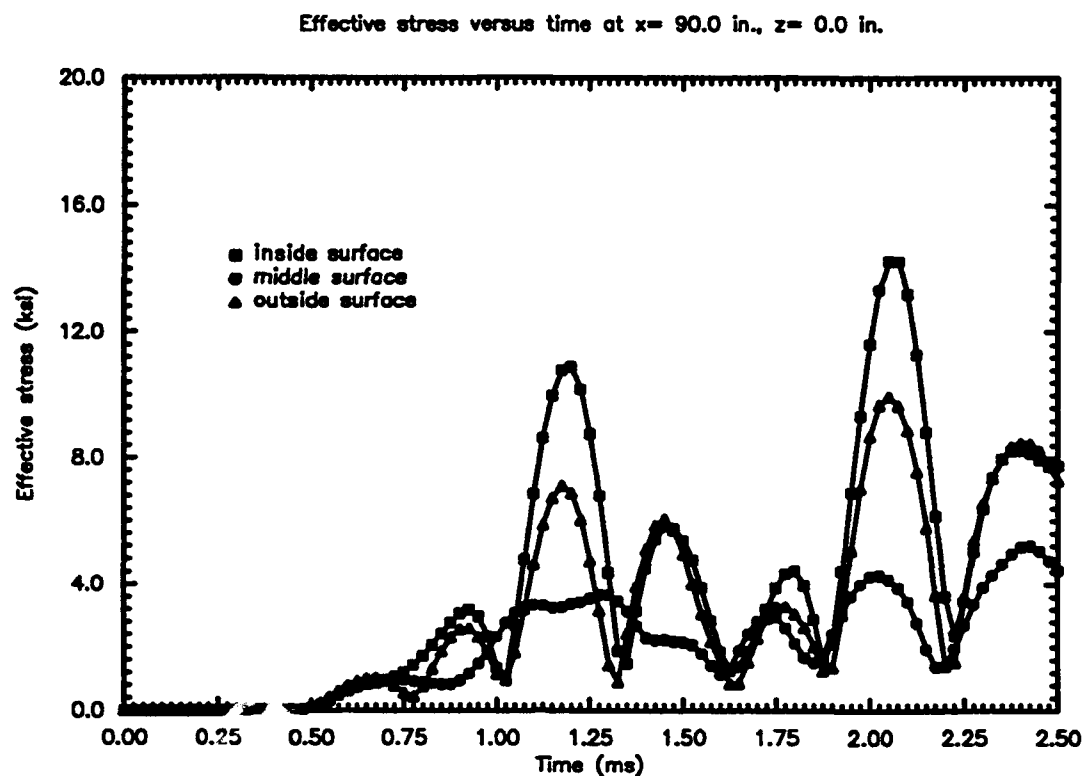


Fig. (3.4-11), effective stress vs. time through thickness of cylinder, $x = 90.0$ inches

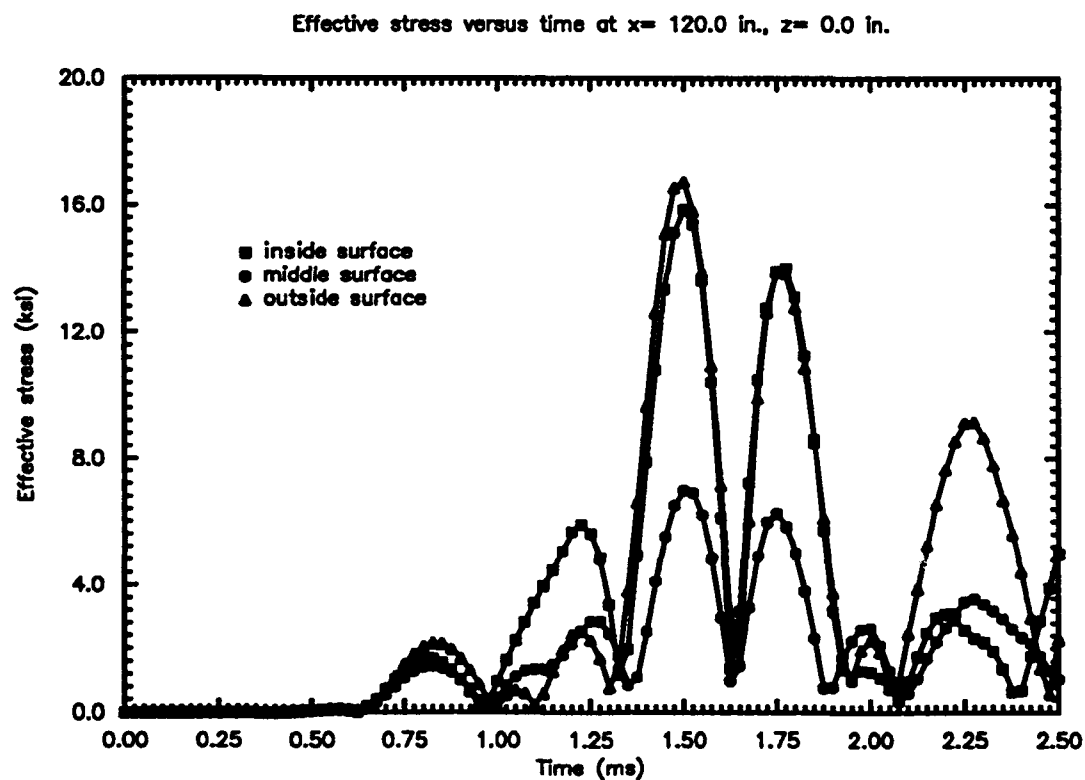


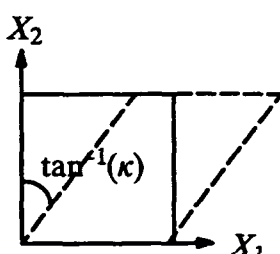
Fig. (3.4-12), effective stress vs. time through thickness of cylinder, $x = 120.0$ inches

3.5 MATERIAL MODEL

In this section several numerical examples involving the finite strain plasticity models are presented. Two examples are given to verify the accuracy of the computer implementations versus simple, known solutions. These are the finite extension and the finite simple shear problems. Then, the boundary layer small scale yield problem of computational fracture mechanics is studied to demonstrate the robustness and reliability of the material models.

3.5.1 THE FINITE SIMPLE SHEAR PROBLEM

The kinematic description of the finite simple shear problem is characterized by the equations (1). The parameter κ , a linear function of time, is a measure of the finite shear de-



$$\begin{aligned} x_1 &= X_1 + \kappa(t)X_2 \\ x_2 &= X_2 \\ x_3 &= X_3 \end{aligned} \quad (3.5-1)$$

$\beta = \tan^{-1}(\kappa/2)$

formation. The closed form solutions for the non-zero Cauchy stresses in terms of κ and the angle β for an incremental linear elastic material law coupling the unrotated stress rate and the unrotated rate of deformation, the Truesdell stress rate and the rate of deformation, and the Jaumann stress rate and the rate of deformation are presented in equations (2). The finite element mesh consists of one unit cube eight node brick element with material properties $E = 206000$ Mpa and $\nu = 0.33$. The shearing was applied at a constant rate through displacement control in increments of κ equal to 0.0001 for 30 load steps and 0.01 for 970 load steps adding up to a final value of 9.703. This pattern of load steps was designed to minimize the error inherent in the Truesdell stress rate formulation seen below in the finite extension problem. A graph of the shear stress versus κ for a linear elastic material is given in Fig. 3.5-1. The agreement between computed and theoretical results for both the Truesdell and unrotated stress rates is excellent. The analytical curve for the Jaumann stress rate shows why the finite simple shear problem, with its concomitant material rotation, is a watershed example when comparing stress rates. The shear stress (as well as the axial stresses) oscillates with increasing shear deformation, with the shear stress actually reversing sign. This is obviously not a

physically plausible response. Although this behaviour appears only at larger shear deformations, it is these large deformations that are of interest. The difficulty with the Jaumann stress rate results from its dependence on the rate of spin tensor which remains constant throughout the deformation. The actual material rotation decreases with the deformation, which approaches finite extension in the limit. This accentuates the need to account for the true material rotation in the choice of an objective stress rate.

unrotated stress rate:

$$T_{11} = -T_{22} = 4G(\cos 2\beta \ln(\cos \beta) + \beta \sin 2\beta - \sin^2 \beta) ;$$

$$T_{12} = 2G \cos 2\beta (2\beta - 2 \tan 2\beta \ln(\cos \beta) - \tan \beta)$$

Truesdell stress rate:

(3.5-2)

$$T_{11} = G\kappa^2 ; \quad T_{22} = 0 ; \quad T_{12} = G\kappa$$

Jaumann stress rate:

$$T_{11} = -T_{22} = G(1 - \cos \kappa) ; \quad T_{12} = G \sin \kappa$$

The finite simple shear problem is also analyzed considering material nonlinearities; a graph of shear stress versus κ is presented in Fig. 3.5-1. To produce this graph, a yield stress of 345 Mpa and a bilinear material with a tangent modulus of 138 Mpa were used. The results are in close agreement with those given by Johnson and Bammann [44]. Notice that the plots involving kinematic hardening have shapes similar to the corresponding nonlinear elastic plot. This is because the evolution equation governing the back stress is hypoelastic in nature and is expressed in terms of an objective rate with the same form as the applied stress rate. Since for a bilinear hardening material the plastic modulus is constant, and recalling that the unrotated formulation can be expressed equivalently in terms of the rate of deformation tensor, the evolution equation is a linear function of the plastic rate of deformation tensor. For the finite simple shear problem, the rate of deformation tensor has only xy shear terms; its deviator tensor has the same form and so does its plastic component. The back stress thus behaves as the nonlinear elastic stress. If the Jaumann stress rate under kinematic hardening were plotted, the shear stress would oscillate just as it does in the nonlinear elastic case [44].

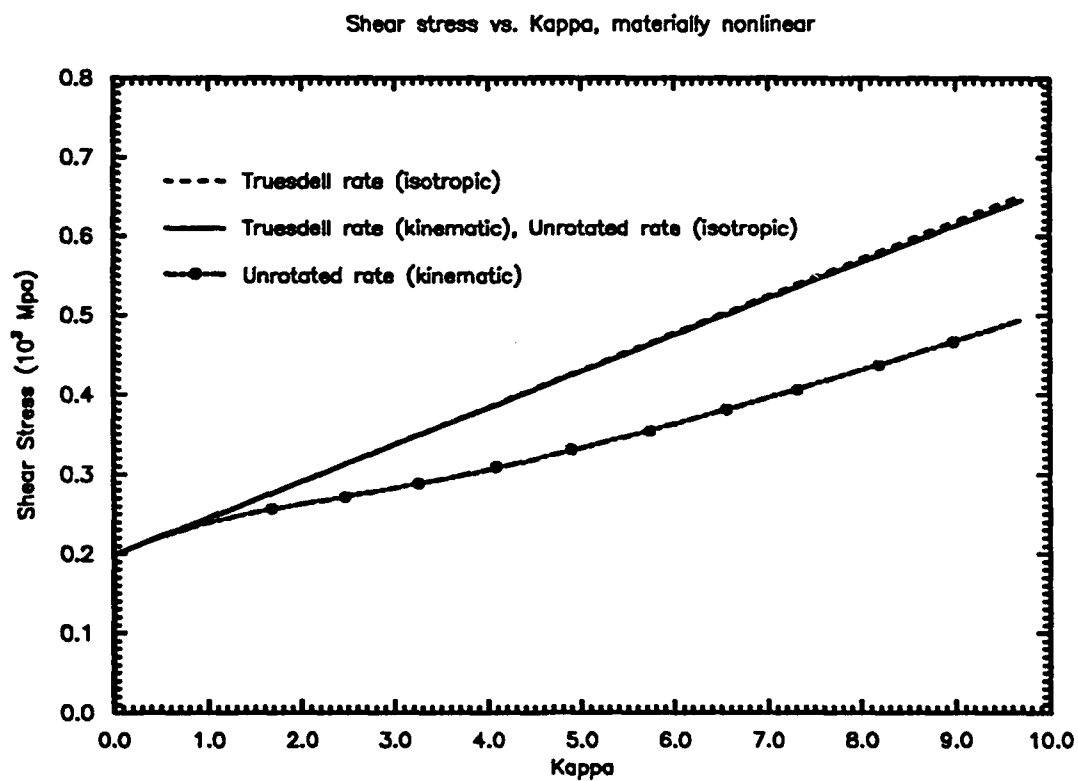
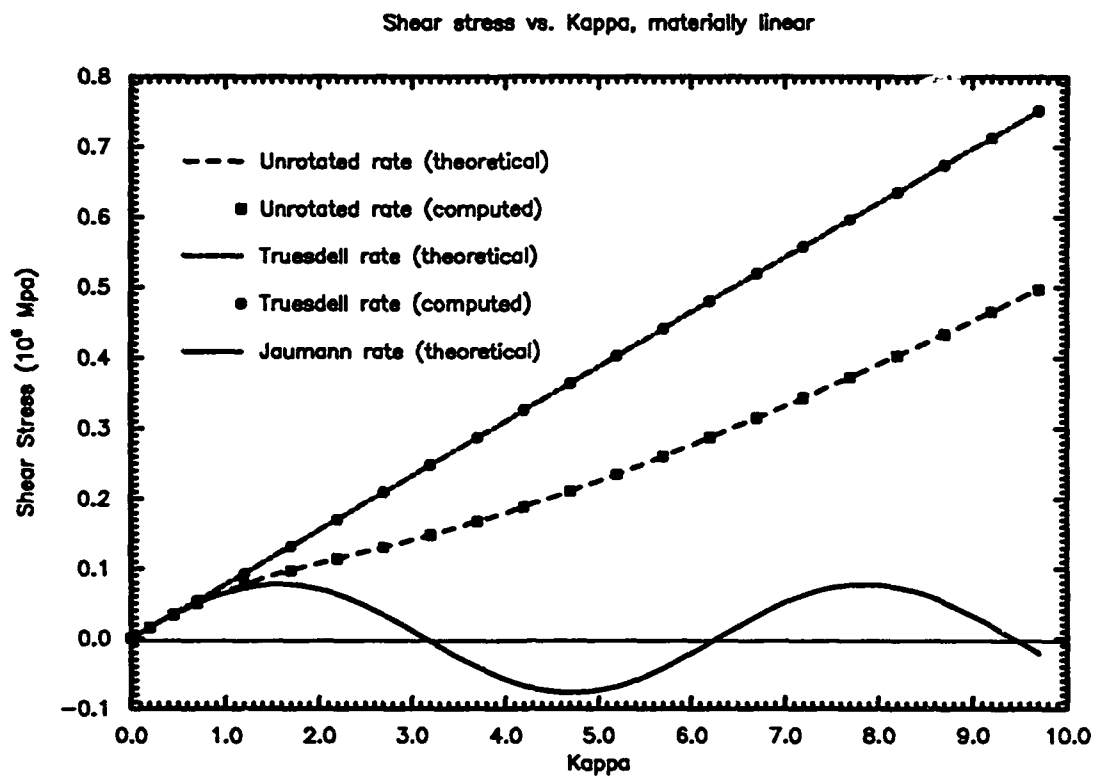
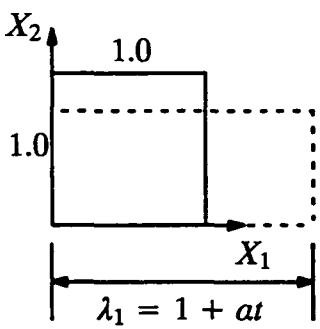


Fig. (3.5-1), finite simple shear results

This illustrates that the insufficiencies inherent in the Jaumann stress rate appear in elastoplastic as well as elastic material models. The isotropic hardening plots of both the unrotated stress rate and the Truesdell stress rate exhibit a bilinear response that resembles the nonlinear elastic response of the Truesdell stress rate, but are not related. The isotropic hardening plots simply reflect the bilinear nature of the material [44].

3.5.2 THE FINITE EXTENSION PROBLEM

The kinematic description of the finite extension problem is given by equations (3). Both the plane stress and plane strain conditions are presented. The stretch ratios λ_i are the



$$\begin{aligned} x_1 &= (1 + at)X_1 \\ x_2 &= (1 + kat)X_2 \\ x_3 &= \begin{cases} X_3 & \text{plane strain} \\ (1 + kat)X_3 & \text{plane stress} \end{cases} \end{aligned} \quad \begin{matrix} a, k \text{ are constants} \\ (3.5-3) \end{matrix}$$

final lengths in the coordinate directions divided by the initial lengths. Their logarithms are the log strains. The closed form solutions for the non-zero Cauchy stresses and the axial force in terms of λ_1 for a linear elastic material in both plane stress and plane strain are shown in equations (4). The finite element mesh consists of one unit cube eight node brick element with

plane strain:

$$\begin{aligned} T_{11} &= \frac{E}{(1-\nu^2)} \ln \lambda_1 ; & T_{33} &= \frac{\nu E}{(1-\nu^2)} \ln \lambda_1 ; \\ P_1 &= \frac{E}{(1-\nu^2)} \lambda_1^{\frac{-\nu}{(1-\nu)}} \ln \lambda_1 \end{aligned} \quad (3.5-4)$$

plane stress:

$$T_{11} = E \ln \lambda_1 ; \quad P_1 = E \lambda_1^{-2\nu} \ln \lambda_1$$

linear elastic material properties $E = 30000$ ksi and $\nu = 0.3$. The axial extension was provided through displacement control on the boundary. The element was stretched quasi-statically to a final length of 6.0 with constant increments corresponding to either 1000 or 50 load steps.

Graphs of the axial force versus λ_1 for plane stress and plane strain in the 1000 load step case are given in Fig. 3.5-2. The agreement between computed and theoretical results is excellent, particularly for the unrotated stress rate, where the accuracy of the computed results is always within 0.0005%. The error of the Truesdell stress rate computed results never exceeds 0.5%. This discrepancy in accuracy is the consequence of using the state n Cauchy stresses in the numerical integration of the Truesdell stress rate, rather than those at state $n + 1/2$ as discussed in Section 2.4.

Fig. 3.5-2 illustrates that the behaviour predicted by the unrotated stress rate and that by the Truesdell stress rate differ significantly. The Truesdell stress rate forecasts that the axial force increases linearly with stretch ratio at large stretch ratios, while the unrotated stress rate suggests that a "limit point" exists after which axial force decreases with increasing stretch ratio. This point, given the above material properties, is at $\lambda_1 = 10.31$ for plane strain and 5.29 for plane stress. The plane stress "limit point" is clearly seen in Fig. 3.5-2. It is not clear which prediction is most correct. Certainly, it would be difficult to test as most materials would yield or break before the unrotated stress rate "limit point" is reached or even before there is an indication as to which path is being followed.

The nonlinear elastic finite extension problem was also solved for 50 load steps in order to study the effect of larger step sizes on numerical accuracy. The results are given in Fig. 3.5-3. The accuracy of the unrotated stress results is virtually unaffected by the larger step size. What little additional inaccuracy that exists is due to the one point integration of the velocity gradient in order to produce the increment of the rate of deformation tensor. A larger step size degrades the accuracy of approximating a curve with a series of straight line segments. However, this does not seem to be a significant effect. The influence of a larger step size on Truesdell stress rate calculations is a far different story. Fig. 3.5-3 shows that the Truesdell stress rate accuracy is quite sensitive to step size. Again, this is due to the dependence on state n in the one point integration of the Truesdell stress rate. The error increases both relatively and absolutely until the computed curve becomes manifestly divergent from the analytical. This indicates that when using the Truesdell stress rate one advantage of implicit computational mechanics, larger allowable step sizes, is rendered less beneficial.

The behaviour of the finite extension problem was also studied with a nonlinear material, using a tangent modulus of 100.0 ksi. Again, displacement control was applied, with the final length of 6.0 being reached in 1000 load steps. The results for both plane stress and plane strain are presented in Fig. 3.5-4. For both cases, the plots for the unrotated rate with both

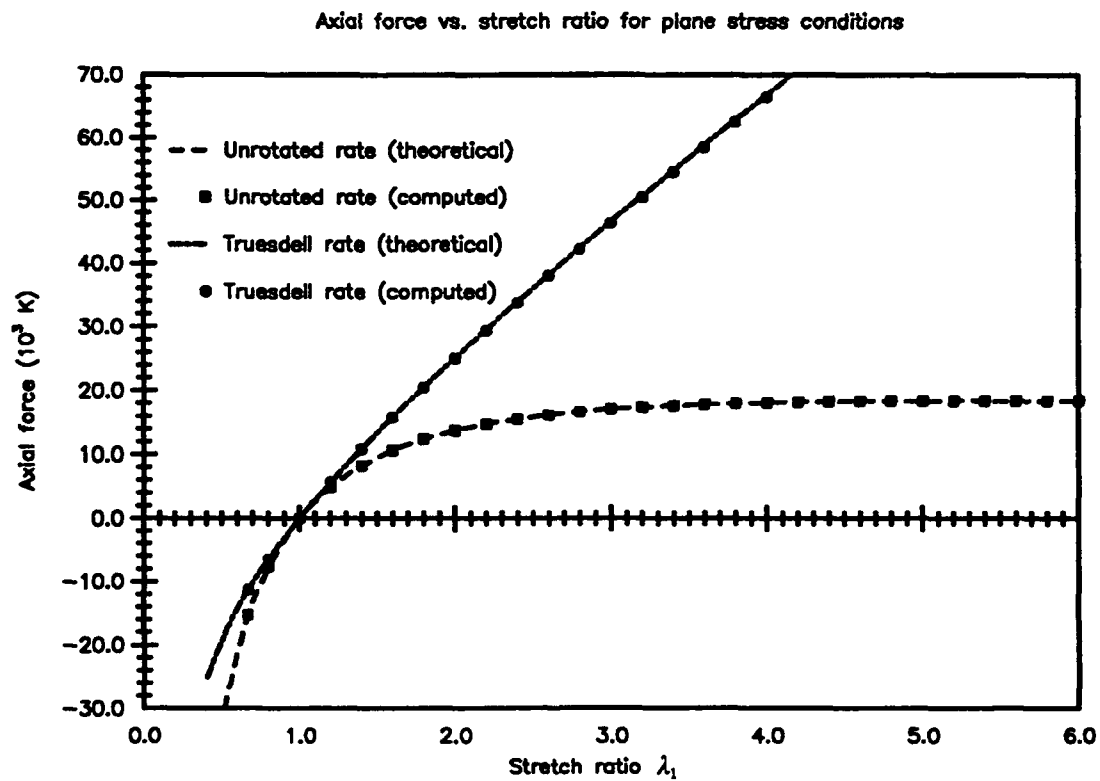
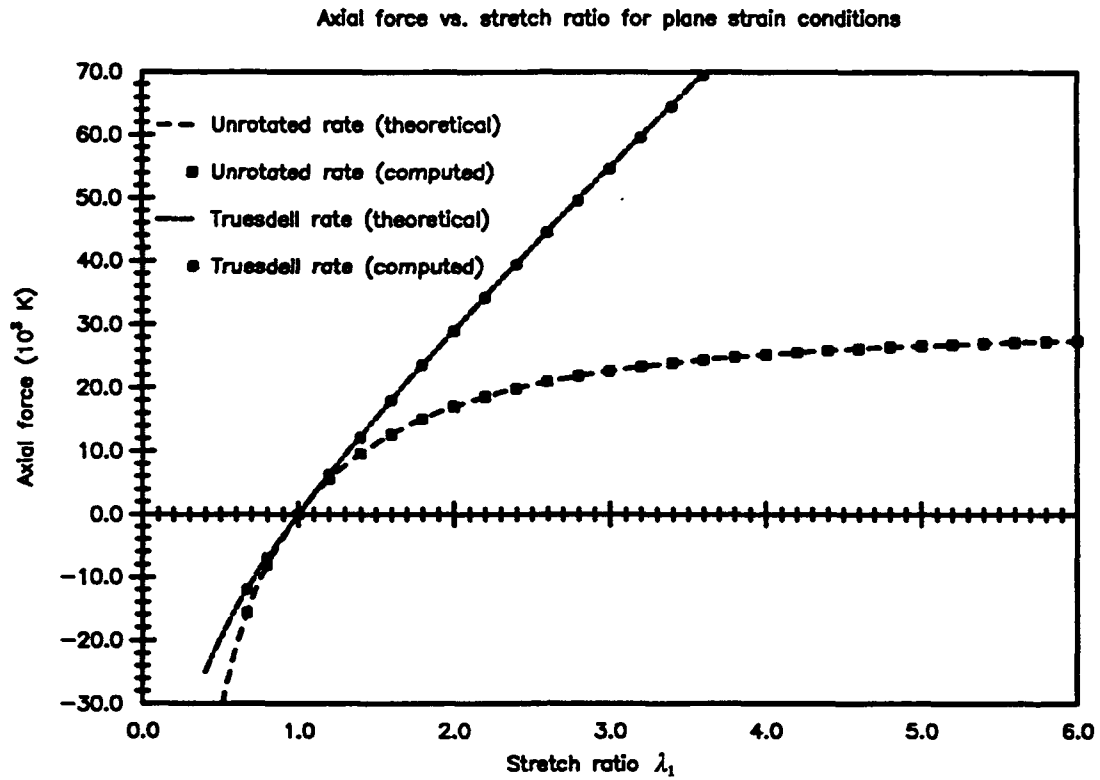


Fig. (3.5-2), nonlinear elastic finite extension results for 1000 load steps

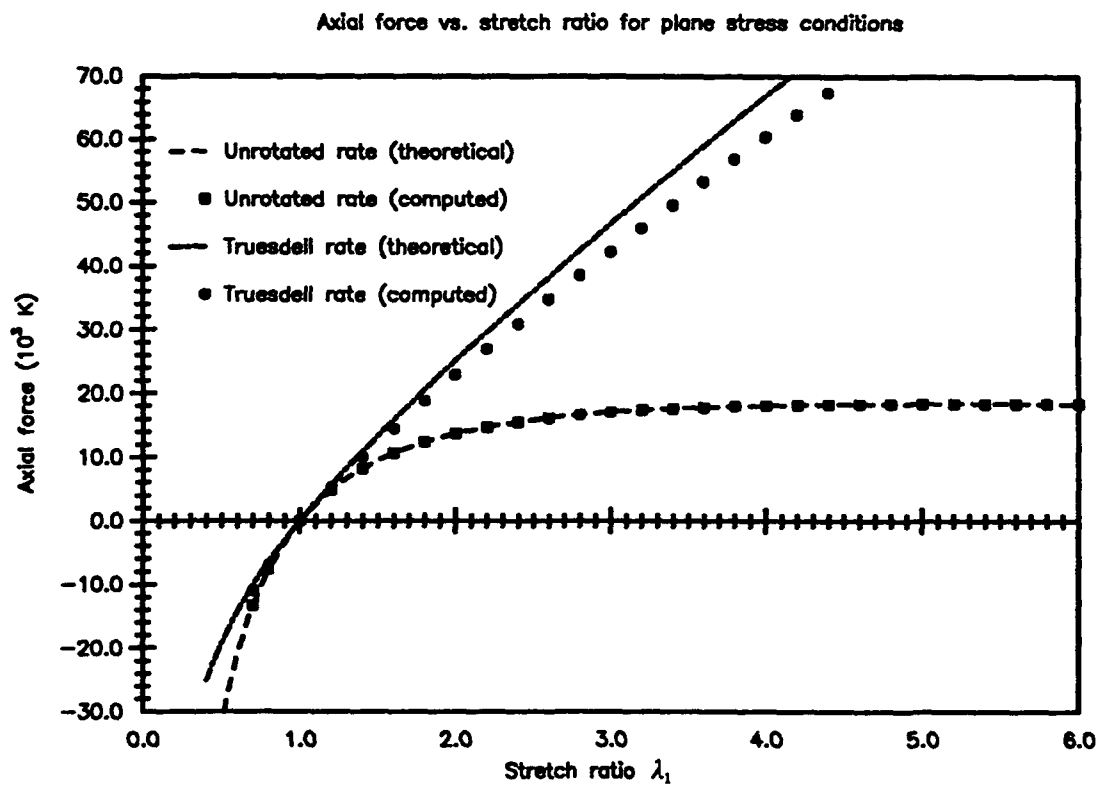
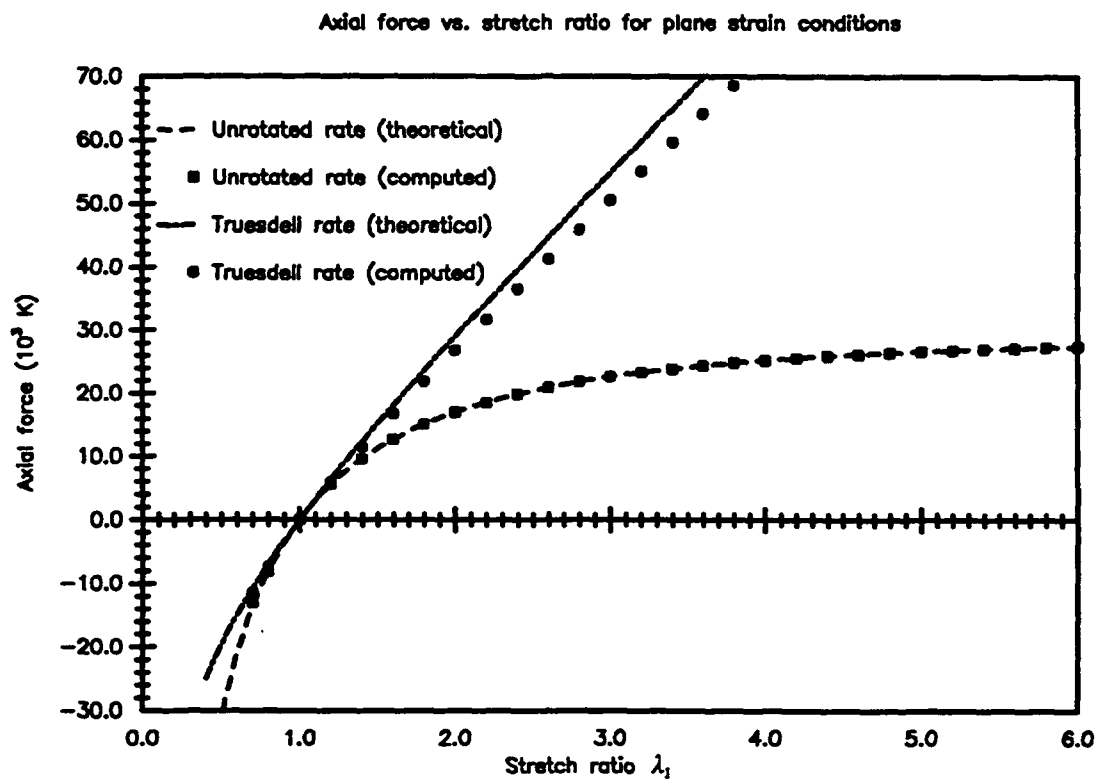


Fig. (3.5-3), nonlinear elastic finite extension results for 50 load steps

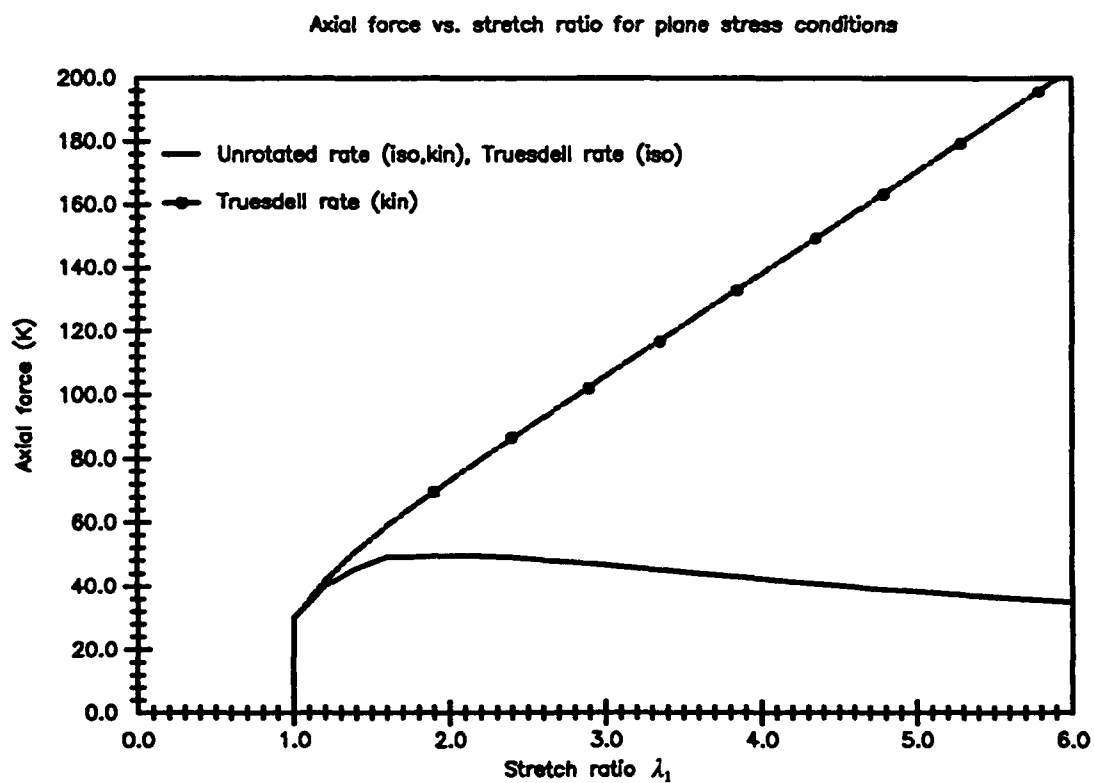
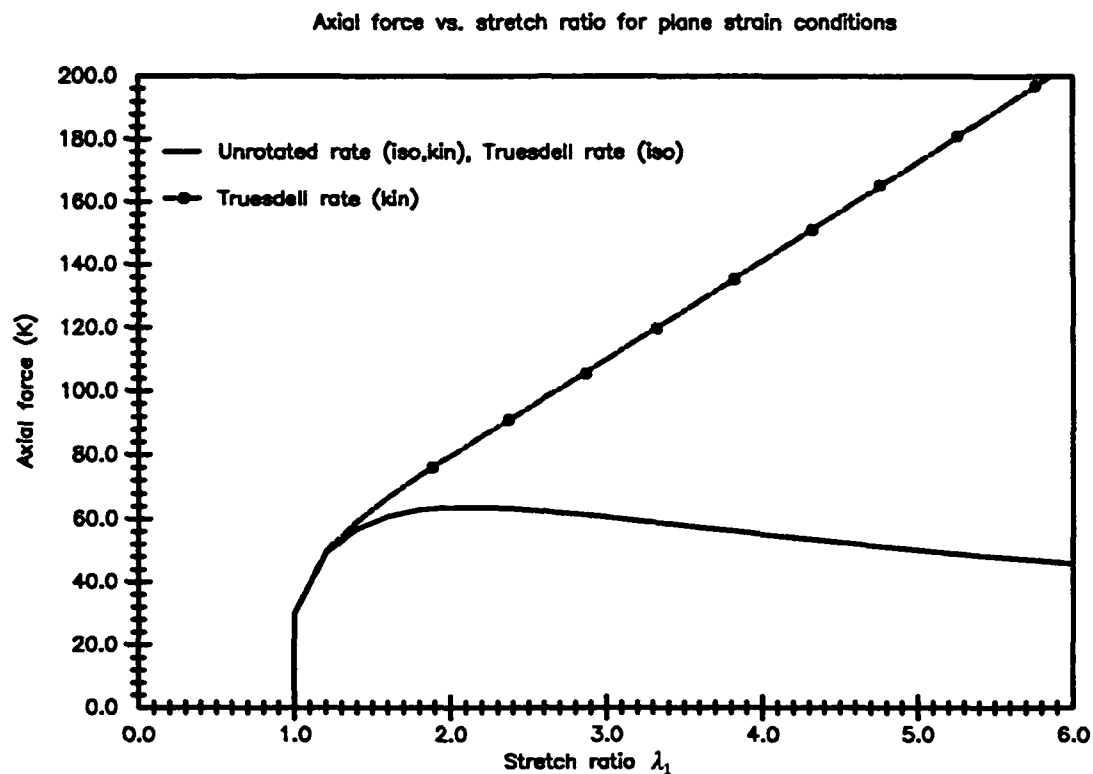


Fig. (3.5-4), finite extension results for a nonlinear material, 1000 load steps

kinematic and isotropic hardening and for the Truesdell rate with isotropic hardening are very nearly identical and show a limit point and a slow decline. The plots for the Truesdell rate with kinematic hardening shows a steady increase with increasing stretch ratio. The Truesdell rate with isotropic hardening closely resembles the unrotated rate with isotropic hardening because in the absence of material rotation the only difference between the Truesdell and unrotated isotropic hardening formulations is the trace of the rate of deformation term in the Truesdell rate. During plastic flow, the plastic trace is zero; the elastic trace is comparatively small. Thus, there is virtually no difference.

As the form of the plastic rate of deformation term is the same as the elastic (diagonal), the back stress for the finite extension problem for both stress rates behaves as the nonlinear elastic stress with a different Poisson's ratio. Since this is also true for the cross-sectional area, the contribution of the back stress to the axial force distribution should mirror the nonlinear elastic axial force distribution. This effect is clearly seen for the axial force plots of the Truesdell stress rate with kinematic hardening, which indeed parallels the nonlinear elastic plots and deviates sharply from the isotropic hardening plots. The plots for the unrotated stress rate with kinematic hardening matches the unrotated isotropic hardening plots because the back stress axial force contribution is similar to the overall plastic distribution and the magnitude of the back stress is not sufficient to cause a significant discrepancy.

As stated previously, the results obtained above were produced by displacement control on the boundary. The convergence of the global equilibrium iterations for displacement control was trivial; only two or three iterations were required even in the 50 load step case. The convergence characteristics for loading control are quite different. Consider the plane stress case for a nonlinear material. For the unrotated stress rate with isotropic hardening, there is a limit point when the axial force reaches the vicinity of 49.6 kips. Loading control was provided so that a final axial force of 49 kips was achieved in 15 variably loaded steps: two steps of 15 kips, twelve steps of 1.5 kips and a final step of 1 kip. Yielding occurs on the second step. The number of equilibrium iterations required for the residual load of each step to converge to within 0.1% of the original residual load is shown in Fig. 3.5-5 for three different analyses. The first is Newton-Raphson. For this analysis, the number of iterations required begins degrading at the onset of plasticity and worsens noticeably as the limit point is approached. The second analysis performs Newton-Raphson with the Crisfield acceleration technique. Here, convergence is seen to improve, especially as the limit point is neared. The final analysis executes the nonlinear preconditioned conjugate gradient algorithm with the tangent stiffness as the linear preconditioner and an accurate line search. Convergence for this analysis is the best of all, most notably near the limit point. Based on Fig. 3.5-5, one can

Step Number	Newton-Raphson	w/ Crisfield acceleration	NLPCG
1	1	1	1
2	2	2	3
3	7	6	5
4	7	6	5
5	7	6	5
6	8	6	5
7	9	5	5
8	9	6	5
9	10	7	5
10	11	7	5
11	13	6	5
12	15	10	5
13	19	11	5
14	27	11	8
15	42	16	8

Fig. (3.5-5), convergence its., finite extension pb., nonlinear material, load control

state that although the convergence associated with Newton-Raphson is disappointing, it can be greatly enhanced by an accelerator or an accurate line search. It may be suspected that the poor performance of Newton-Raphson for the unrotated stress rate in this example is due to the omission of the stress related terms in the transformation of the constitutive relationship to the reference configuration. Indeed, as the solution approaches and exceeds the limit point, the change in the right stretch tensor becomes more and more significant. However, the same convergence trends are observed using the Truesdell stress rate, whose constitutive transformation is not dependent on approximation. Further, the same convergence trends are observed for the nonlinear elastic solution using the unrotated stress rate, which has a "limit point", but not when using the Truesdell stress rate, which does not. Using the Truesdell stress rate, the nonlinear elastic solution converges in two or three iterations for a reasonable step size throughout the analysis. It may be that it is the presence of a limit point which causes convergence problems, which is not unreasonable. Further research is required to discover if

the missing terms of the unrotated constitutive transformation are necessary to proper convergence when using the unrotated stress rate.

3.5.3 THE BOUNDARY LAYER SMALL SCALE YIELD PROBLEM

The boundary layer small scale yield problem is an important one for fracture mechanics researchers who wish to study crack tip parameters [61]. It models an infinite plate in plane strain with a long crack in the center and tensile stress applied at infinity. The crack is long enough so that the crack tips do not interact. The boundary layer consists of an annular region containing the crack tip, which may be either sharp or smoothly blunt. In this study, it is blunt. The infinite plate with including the center crack and the annular boundary layer is illustrated in Fig. 3.5-6, the mesh modeling the annular region is presented in Fig. 3.5-7, and the loading

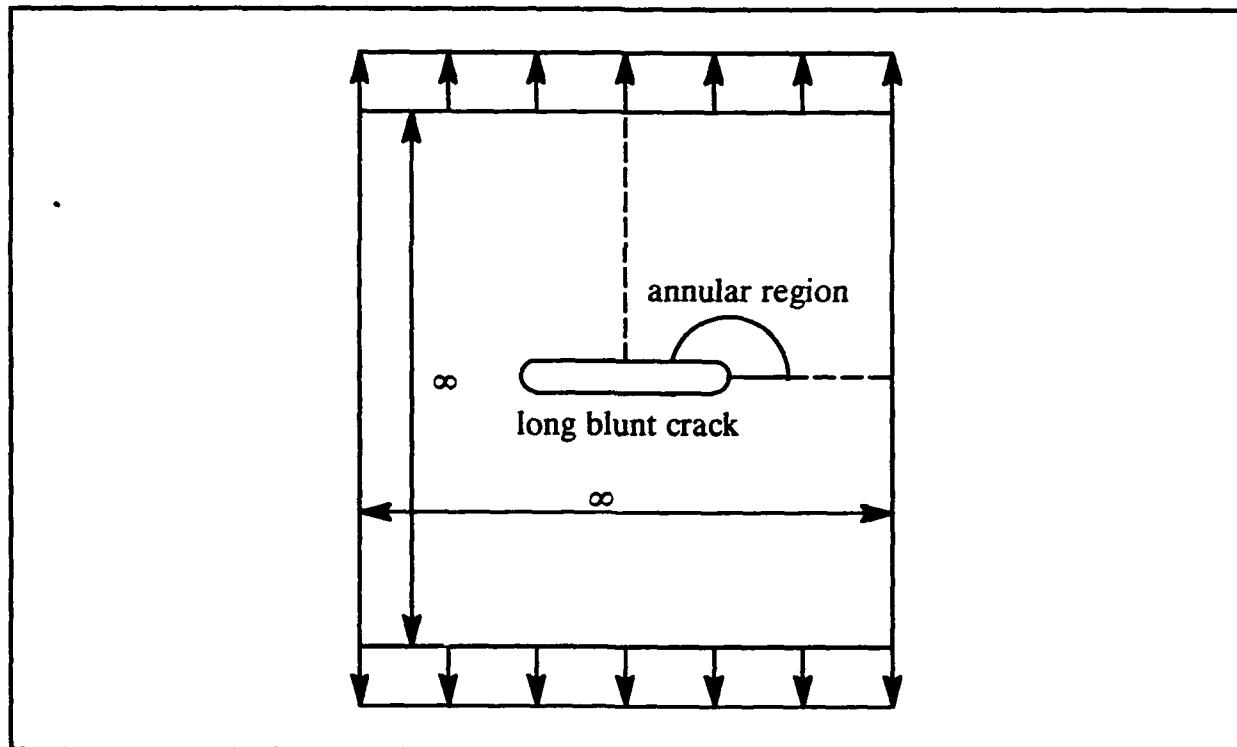


Fig. (3.5-6), location of semicircular mesh

history and material parameters are shown in Fig. 3.5-8. A close-up of the crack tip region is given in Fig. 3.5-9. Increasing displacements that correspond to the linear elastic (Mode I), small strain singular field are applied along the outer circular boundary of the annular mesh. These displacements are given in equations (5) where K_I , the stress intensity factor, is seen to be the loading parameter. The justification for applying displacements along the boundary is

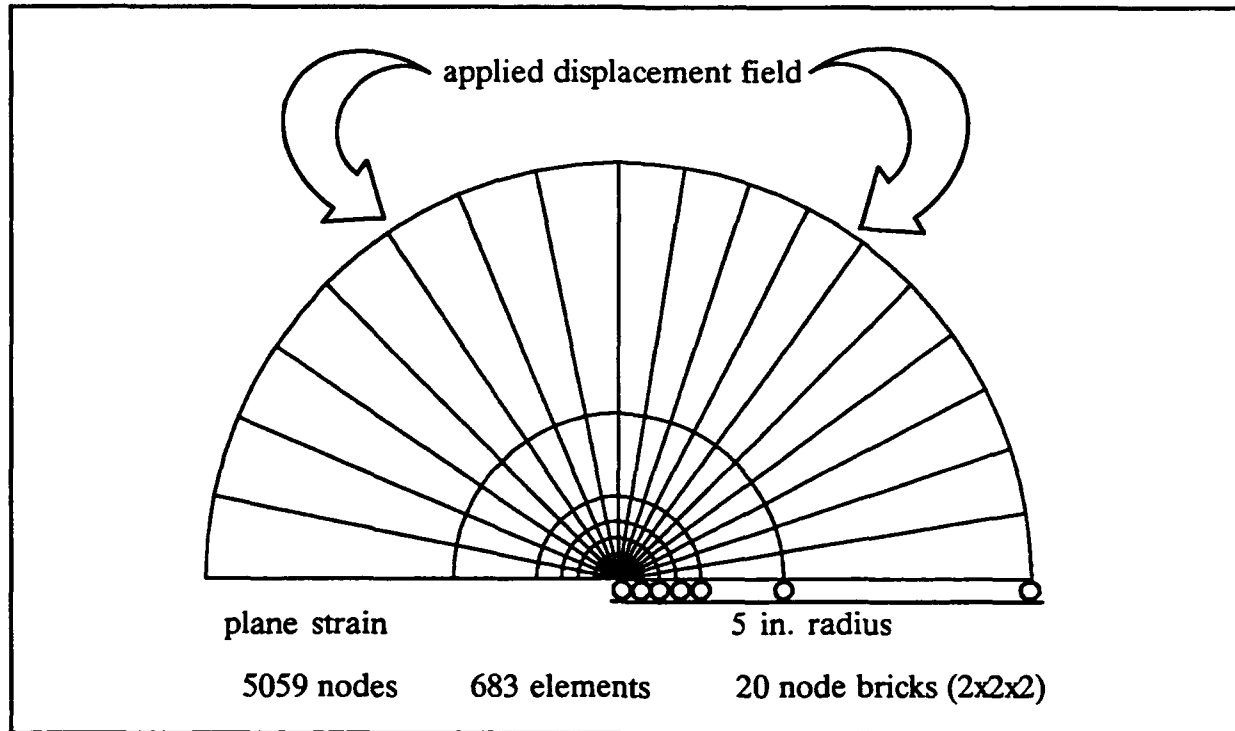


Fig. (3.5-7), mesh for boundary layer small scale yield problem

that the plastic zone around the the crack tip is assumed to be small enough so that the bound-ary is unaffected. In this analysis, the overall radius of the mesh is set to 5 inches, while the initial crack radius is 0.001 inches. The plane strain condition is enforced by constraining all displacements normal to the plane of the plate to be zero. The thickness of the plate is there-fore basically irrelevant, but is set to 0.001 inches so that the transverse edges of the 20 node brick elements cannot deform into zero energy modes, which they do if the thickness is too large. Reduced integration is used on all elements and the value of K_I is increased to a final value of 150 in equal increments. In the primary analysis of the problem, the unrotated stress rate and isotropic hardening are used.

$$u = \sqrt{\frac{r}{2\pi}} \frac{(1+\nu)K_I}{E} (3-4\nu - \cos\theta) \cos\frac{\theta}{2}$$

$$v = \sqrt{\frac{r}{2\pi}} \frac{(1+\nu)K_I}{E} (3-4\nu - \cos\theta) \sin\frac{\theta}{2}$$

(3.5-5)

The displaced shape in the crack tip region for $K_I = 150$ is shown plotted to actual scale in Fig.3.5-10. The crack tip opening displacement δ , defined in Fig. 3.5-10, is nearly four times greater than the initial crack opening (twice the initial crack radius). The crack tip thus increases significantly in both size and bluntness as the solution progresses. The elements along the radius of the crack tip are severely disfigured and experience very large strain. This can be seen in Fig. 3.5-11 where a contour plot of the effective strains about the crack tip is given. The maximum effective strain occurs in the element at the crack tip and reaches a value of 1.59. As the strains result from an accumulation of the rate of deformation tensor, they resemble logarithmic strains and the maximum effective strain corresponds to a stretch ratio of approximately 4.9. This indicates that for the area around the crack tip finite strain effects must be included in the plasticity models.

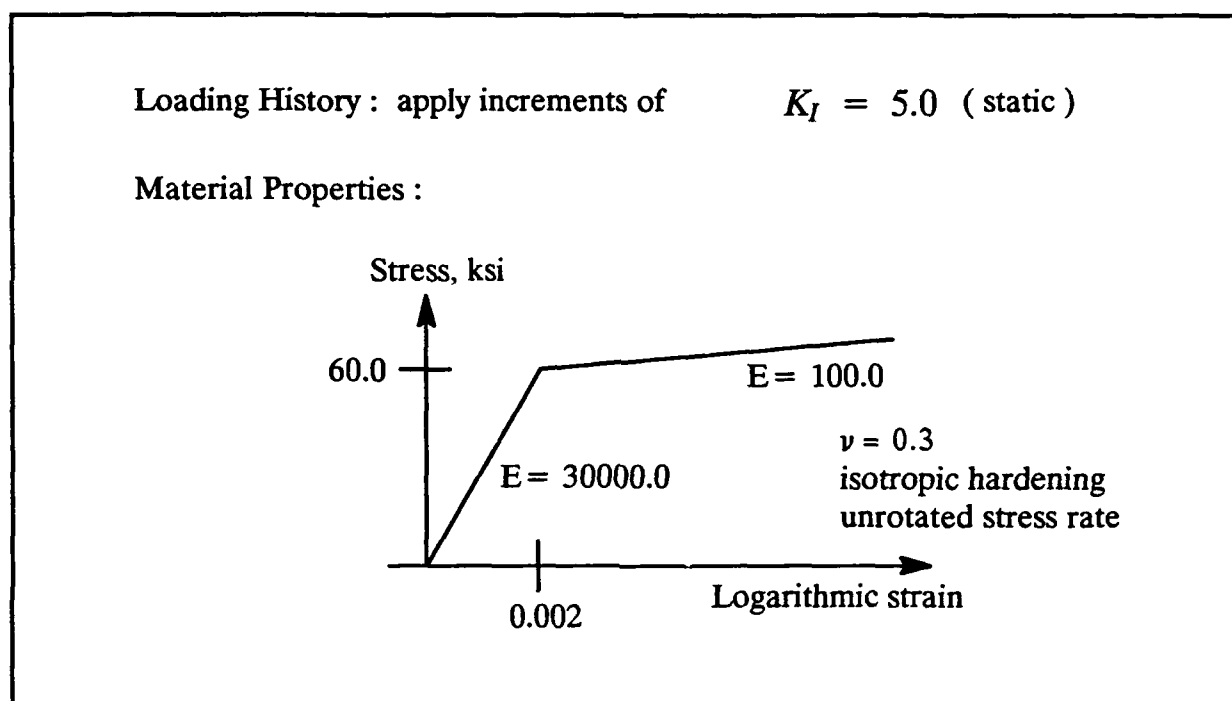


Fig. (3.5-8), problem definition for boundary layer small scale yield problem

The evolution of the plastic zone about the crack tip is shown in Fig. 3.5-12 for K_I values of 50, 100, and 150. An effective strain in the vicinity of 0.002 or greater indicates the region of plastic flow. The extent of the plastic zone for the three cases is about 0.1, 0.4, and 0.8 inches, respectively. The plastic zone for $K_I = 150$ appears to be somewhat affected by the boundary, but an analysis for a mesh with an outside radius of 10 inches reveals that this effect is actually insignificant. Notice that the plastic zone tails to the right or toward the free edge of the plate, which is expected.

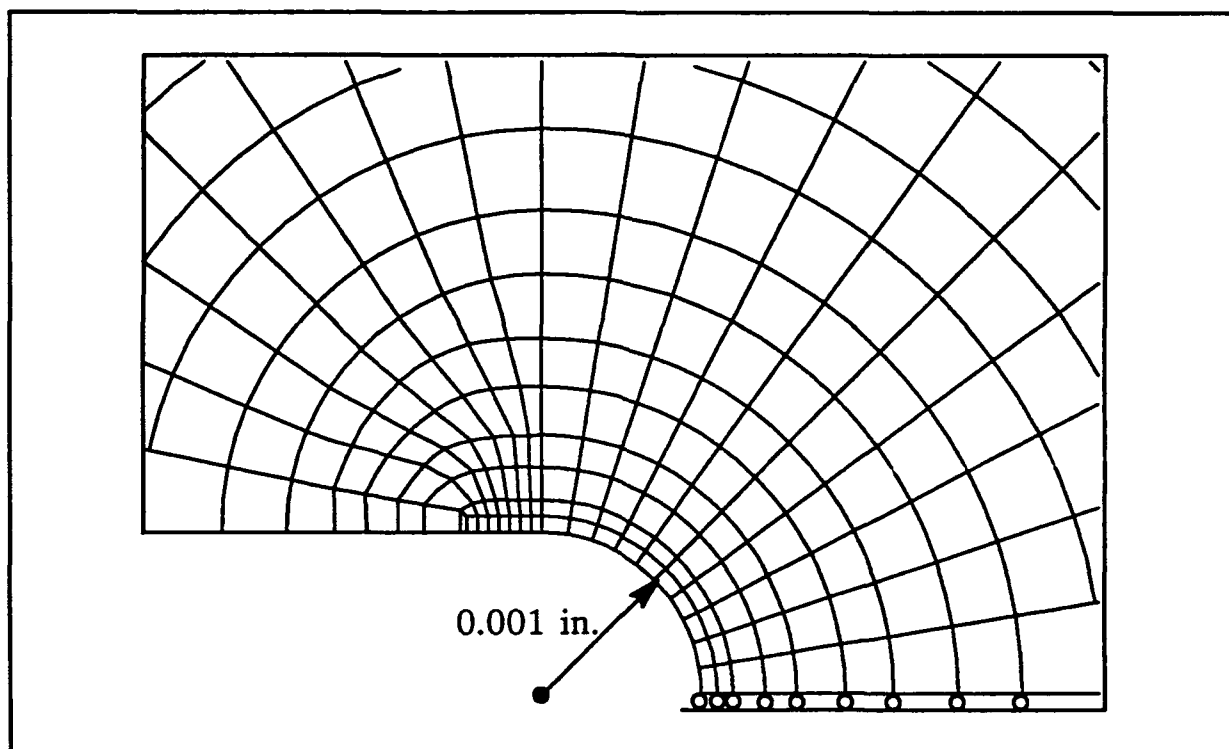


Fig. (3.5-9), close up of mesh around crack tip

One of the principal goals of the analysis of the boundary layer small scale yield problem is to develop a plot of normalized stress versus normalized radius that is independent of the size of the plastic zone. Thus, the normalized stress distribution for any sufficient size of the plastic zone would be identical and only one graph would be needed to represent all. Such graphs are shown in Fig. 3.5-13 for the radial and hoop Cauchy stress along the radius at $\theta = 0$ (the shear stress along this radius is zero by symmetry, which is verified by computation). The stress of interest is normalized by the uniaxial yield stress and is plotted on the vertical axes. The radius is normalized by the parameter J divided by the uniaxial yield stress and is plotted on the horizontal axes. The parameter J [61] is the J -integral, which is a contour integral that is essentially path-independent for contours beyond the finite strain zone. The theoretical value of J , given in equation (6), is used to normalize the radius.

$$J = \frac{(1 - \nu^2) K_I^2}{E} \quad (3.5-6)$$

In the radial stress plot of Fig. 3.5-13, the curves for all values of K_I dive to zero at the crack tip. This is because the crack is stress-free. This boundary condition is also responsible for the downturn in the hoop stress plot near the crack tip, because the radial stress drives the hydrostatic stress down. The hoop stress must then decrease in order to satisfy the yield condi-

tion. The upturn in the hoop stress plot near the crack tip is currently a controversial topic among fracture mechanics researchers. It appears that it results from the bilinear hardening assumption in the material model. As the bilinear hardening material near the crack tip becomes more and more deformed, it continues to pick up stress even at extremely large strains. This effect counteracts the effect of the decreasing radial stress and drives the hoop stress up at the crack tip. For materials governed by a power law hardening rule, where at very large strains the tangent modulus is virtually zero, the upturn is not seen [61].

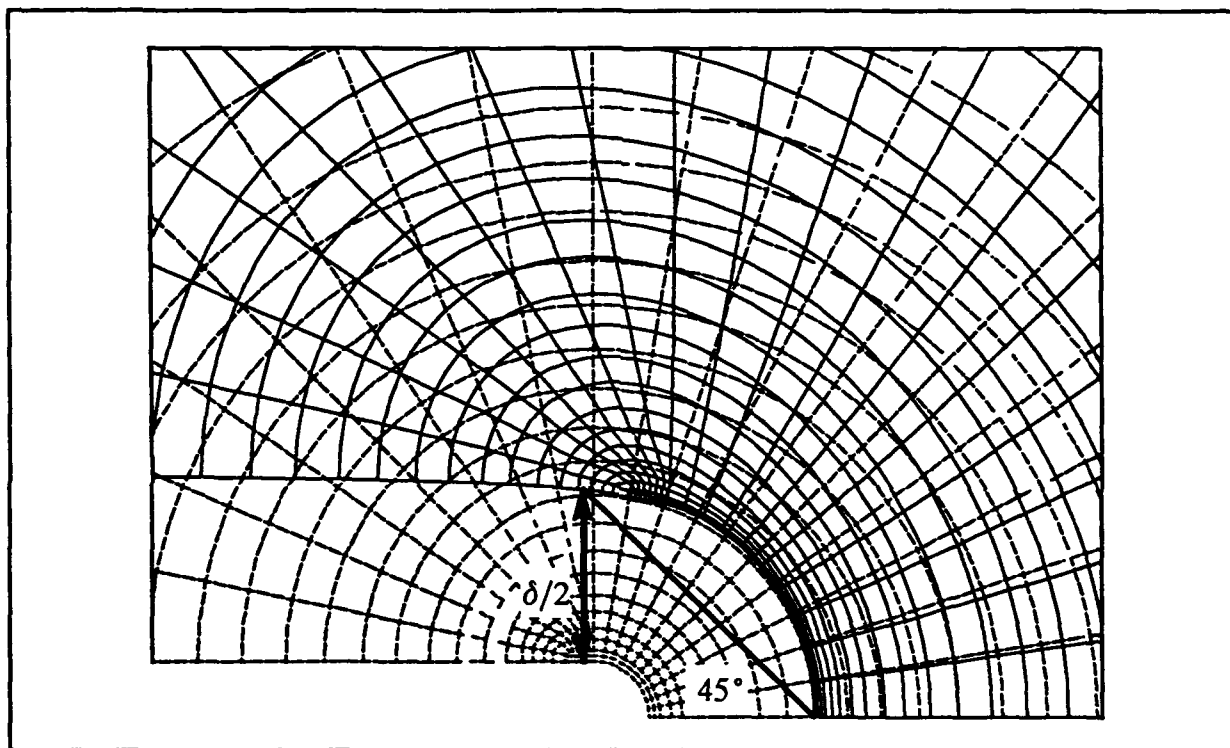


Fig. (3.5-10), crack tip displacement

An important feature of Fig. 3.5-13 is that while the normalized stress distributions for $K_I = 100$ and 150 coincide after a certain distance from the crack tip they do not coincide prior to this point. This indicates that the parameter J is not sufficient to normalize the radius in the near crack region where finite strain effects are important.

Normalized stress plots for solutions corresponding to the Truesdell stress rate with isotropic hardening and the unrotated stress rate with kinematic hardening are practically identical with those of Fig. 3.5-13. The similarity for different stress rates is probably an indication that the material rotation is not significant enough to differentiate between the stress rates. The similarities for a given stress rate and varied strain hardening exist because for this case isotropic and kinematic hardening are not clearly distinguishable. There is an observable

difference in the finite simple shear and finite extension problems because the plastic rate of deformation tensor has the same form as the rate of deformation tensor itself. Here, the spatial and deformation gradients are different for plastic flow and elastic deformation, so that the back stress in kinematic hardening does not resemble the nonlinear elastic solution.

In Fig. 3.5–14 normalized stress plots for the Truesdell stress rate with kinematic hardening are presented. While the radial stress plot is similar to that of Fig. 3.5–13, the hoop stress plot is quite different near the crack tip for $K_I = 100$ and 150. The upturn in the vicinity of the crack tip is greatly exaggerated. Because the deviator values of the back stress are not preserved by the Truesdell rate transformation of the state n back stress to the current configuration, the hydrostatic back stress is augmented by the transformation and the hydrostatic Cauchy stress is driven up in kind. This effect is only significant in the area of high deformation surrounding the crack tip and points out the problems associated with using the Truesdell stress rate and kinematic hardening together.

In Fig. 3.5–15 the convergence of the global Newton–Raphson equilibrium iterations for finite strain plasticity versus small strain plasticity is examined. The boundary layer small scale yield problem was analyzed assuming both finite and small strains and the convergence characteristics of the load step increasing K_I to 100 were studied. For the geometrically nonlinear plots, the unrotated stress rate with isotropic hardening was used. All other combinations of stress rate and strain hardening type exhibited similar performance. Convergence for both the finite and small strain solutions were excellent, each requiring only three Newton–Raphson iterations to reduce the residual to 0.01% of the original step residual. The small strain or geometrically linear solutions display the quadratic rate of convergence characteristic of the consistent tangent operator. The finite strain solutions diverge from the small strain solutions, but converge to the 0.01% tolerance in the same number of iterations. Therefore for the boundary layer small scale yield problem, where the degree of overall geometric nonlinearity is not large, no serious deterioration in the convergence rate is observed for a geometrically nonlinear solution.

Finally, the computation time for the boundary layer small scale yield problem required by the unrotated stress rate formulation, due to the performance of the polar decompositions, was consistently about 67% greater than that of the Truesdell rate formulation. However, this difference represented less than 1% of the overall solution time for 30 load steps, which was about three hours on a Convex 240 computer. Thus, there is no significant extra cost incurred by the use of the unrotated formulation in a real problem.

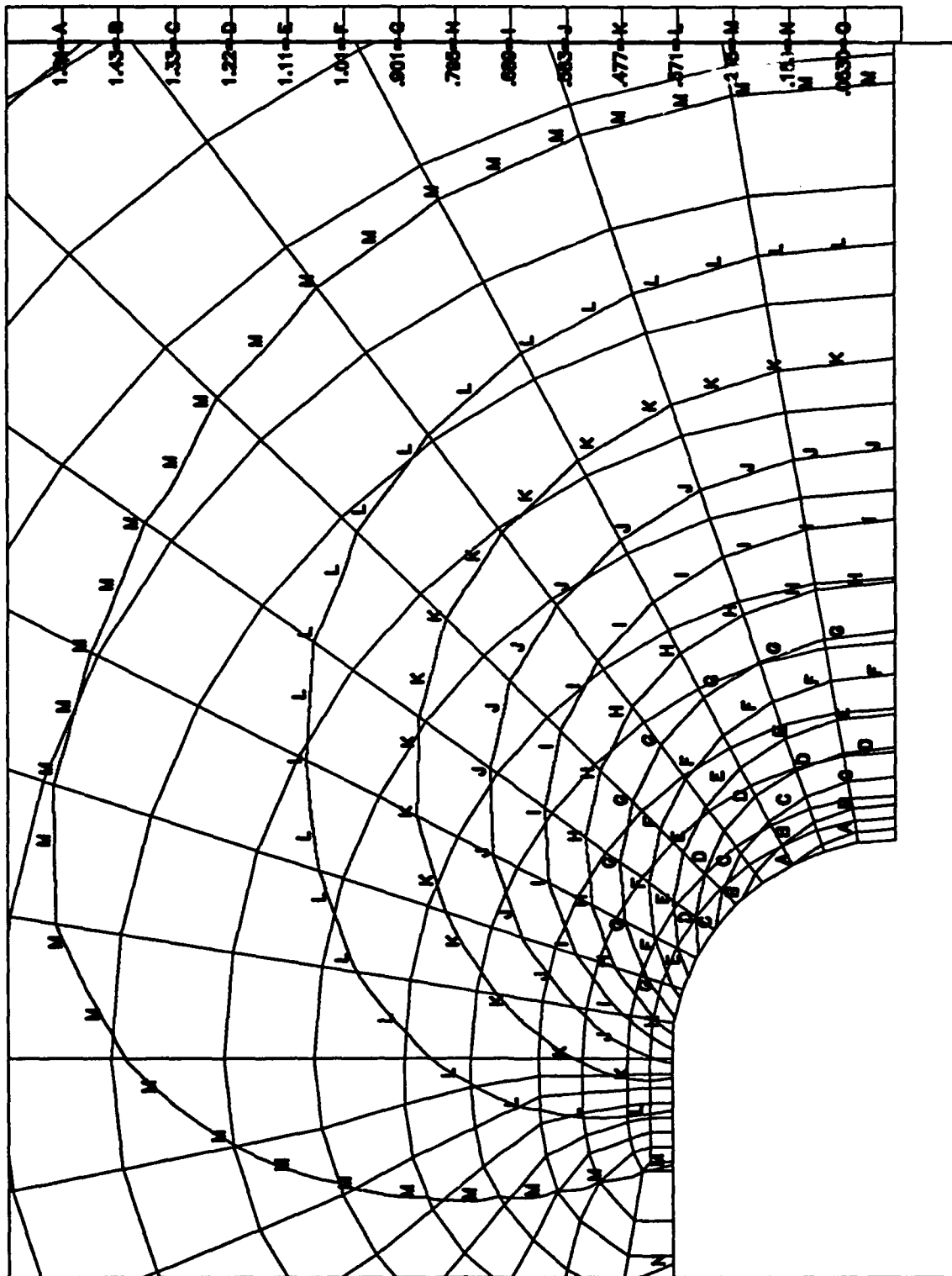


Fig. (3.5-11), effective strain around the crack tip for $K_I = 150$

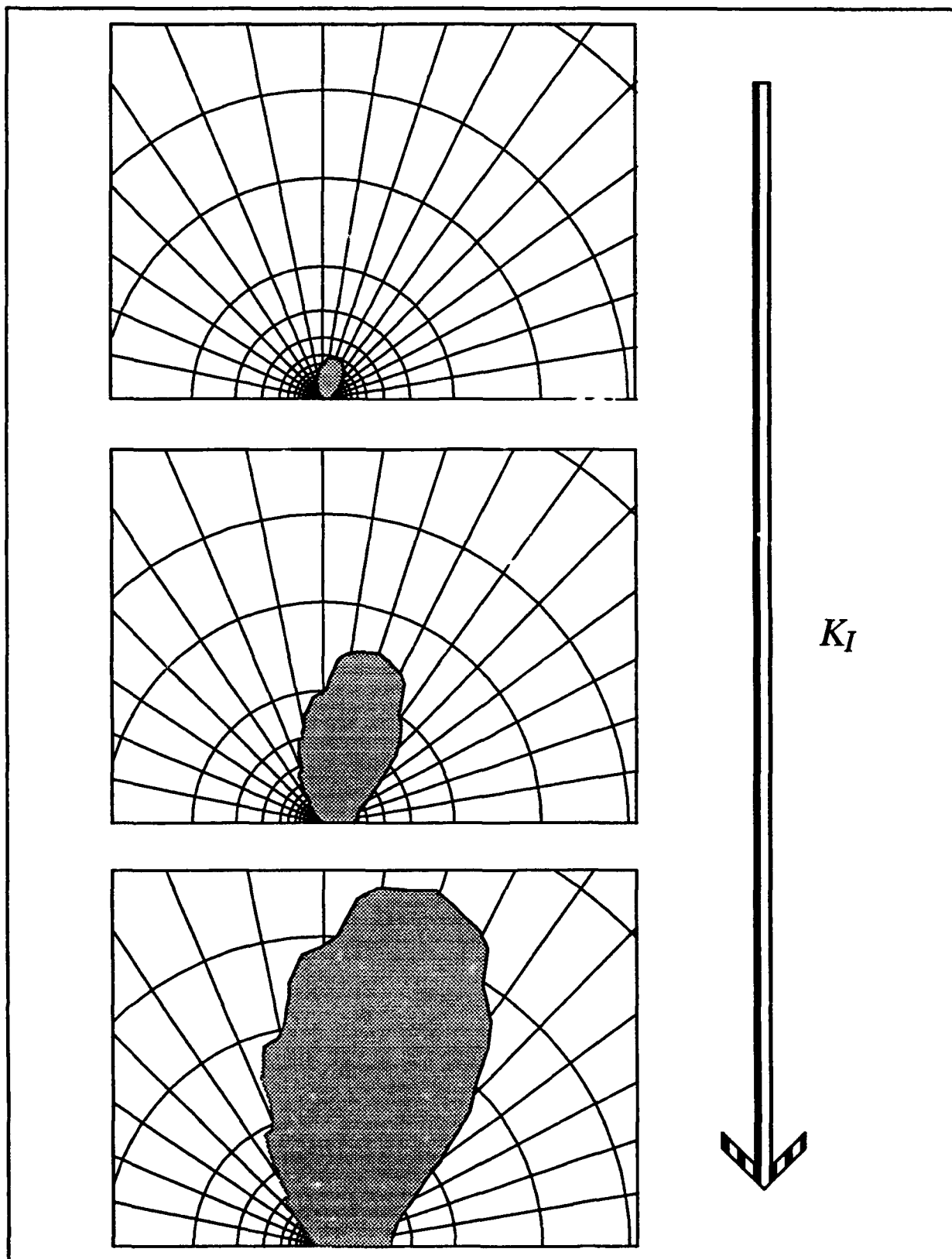


Fig. (3.5-12), size of plastic zone for $K_I = 50, 100, 150$

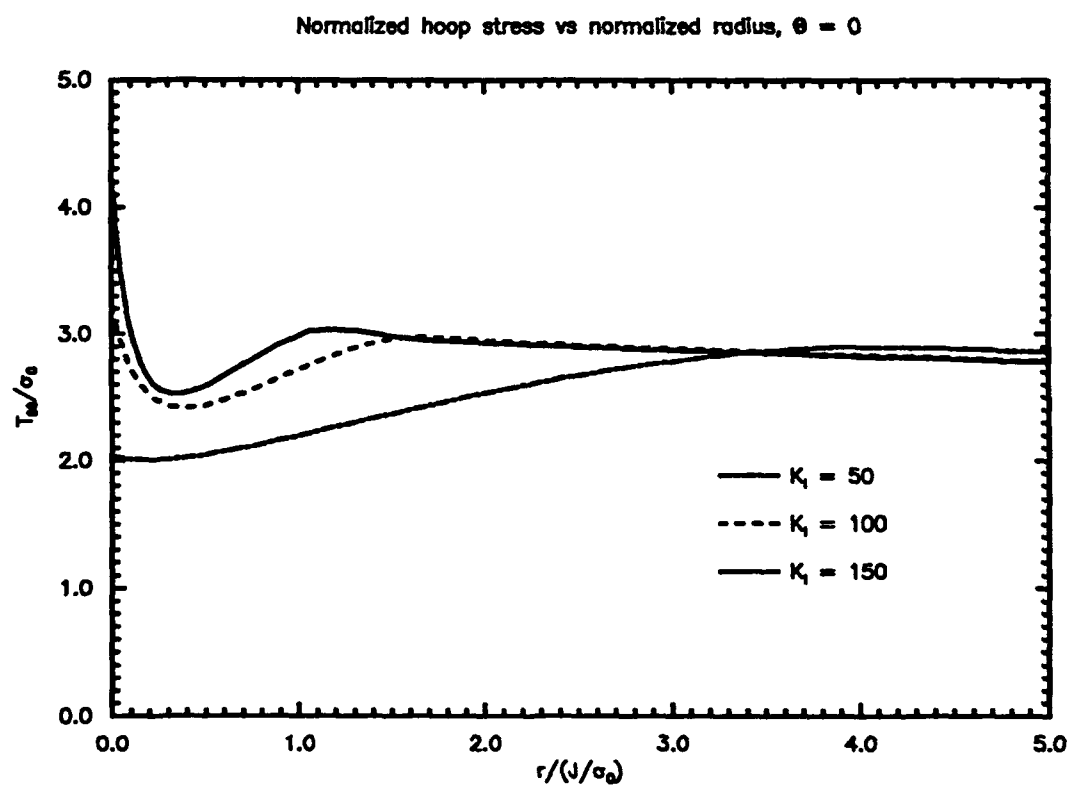
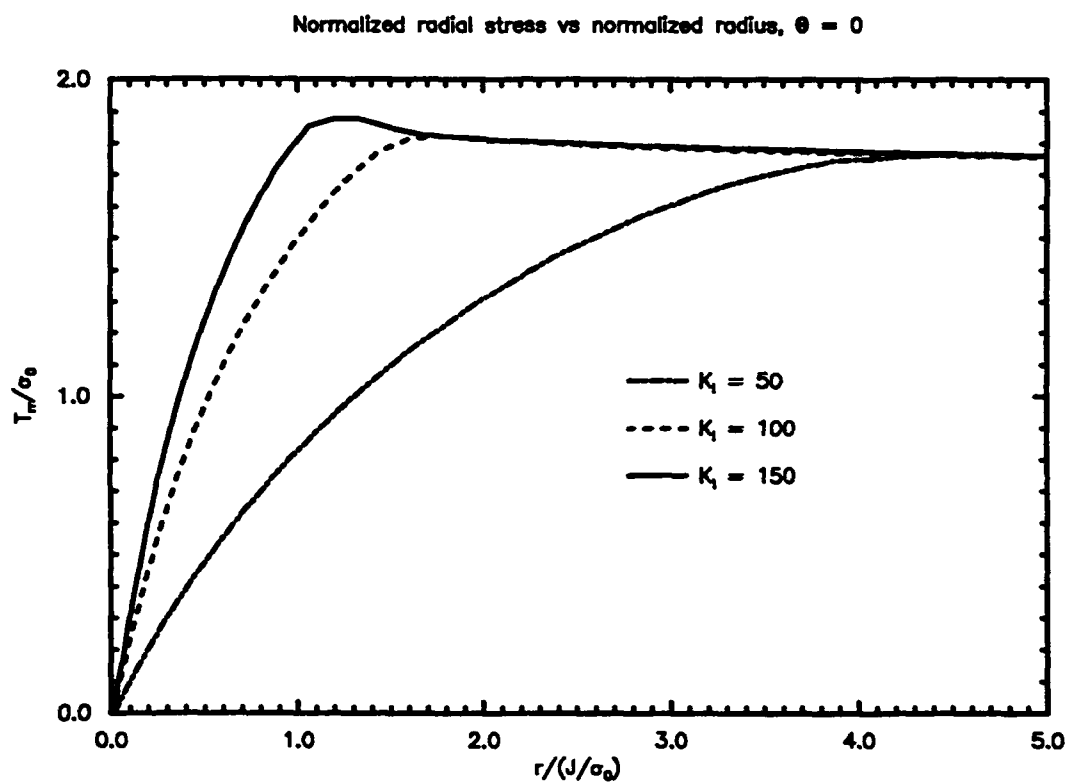


Fig. (3.5-13), normalized stress plots for unrotated stress rate, isotropic hardening

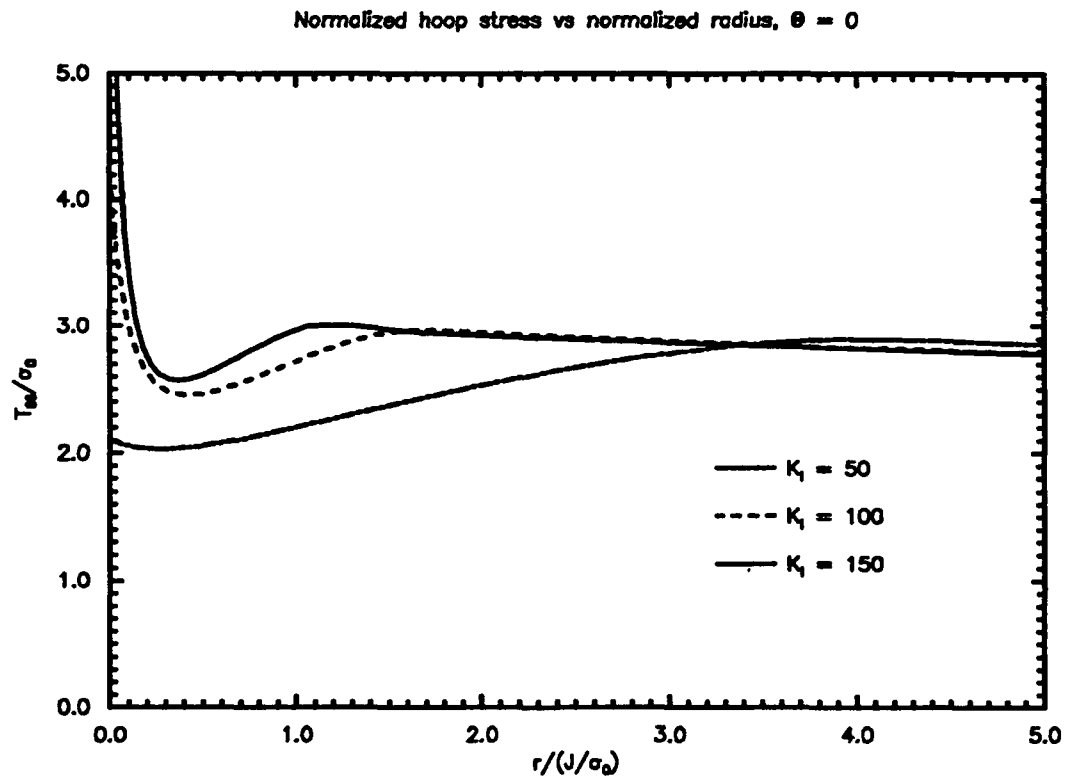
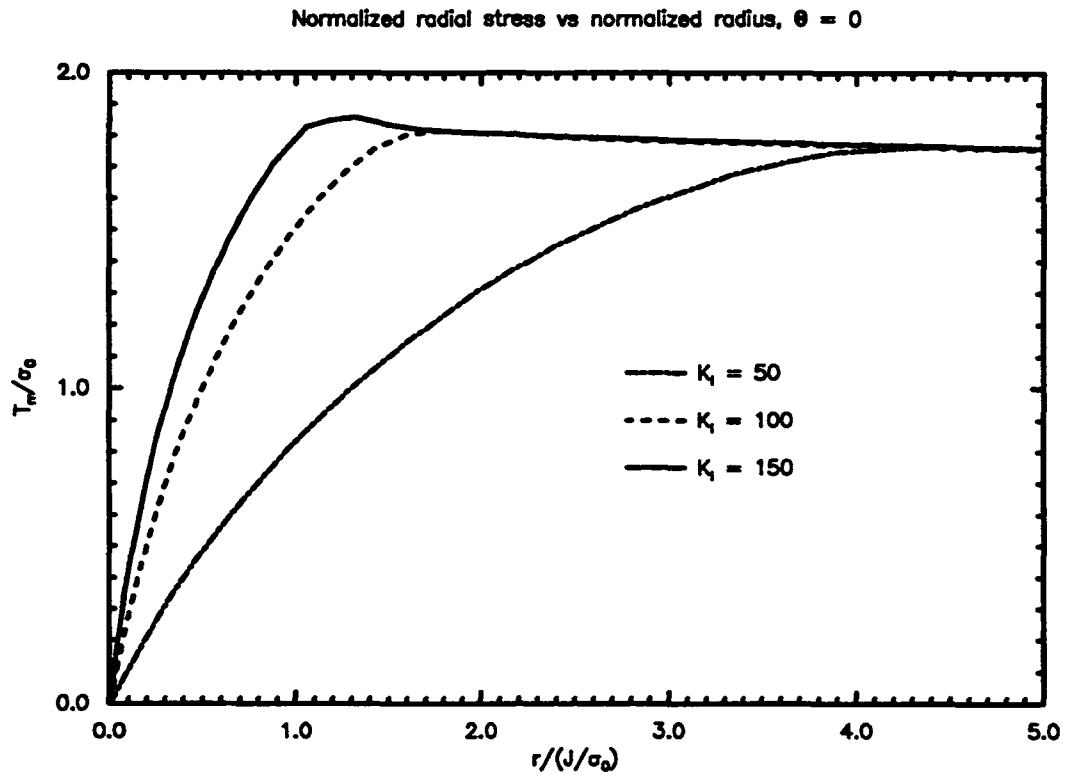


Fig. (3.5-14), normalized stress plots for Truesdell stress rate, kinematic hardening

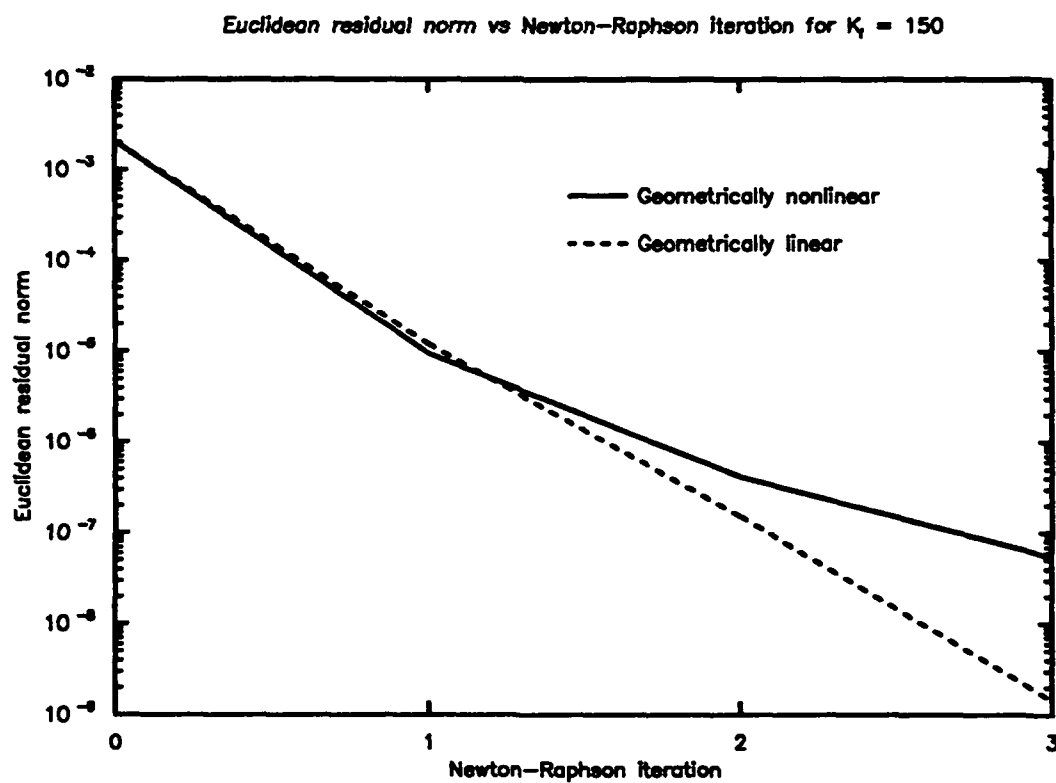
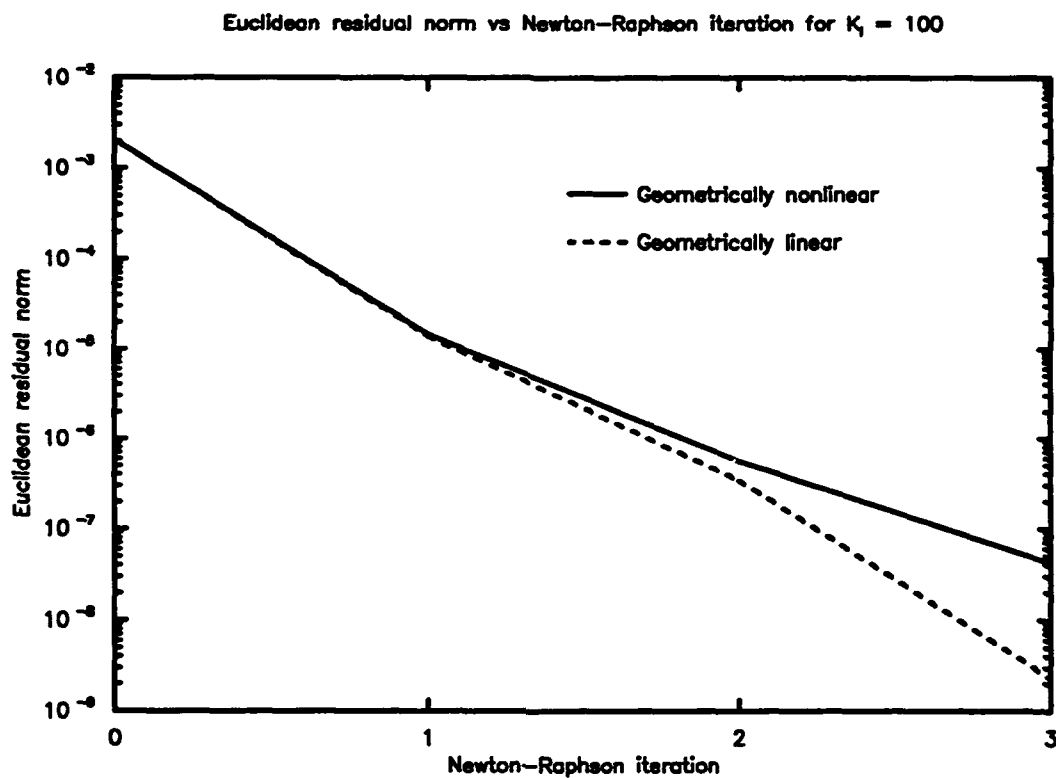


Fig. (3.5-15), convergence studies for unrotated stress rate, isotropic hardening

4 NUMERICAL RESULTS: TIMINGS

This chapter deals with timing studies performed on various example problems so that the performance of the element computation and linear and nonlinear solution algorithms can be assessed.

All timing studies executed for the purposes of comparing element computation algorithms and linear solvers within nonlinear solution algorithms employed modified Newton-Raphson (MNR) with Crisfield acceleration (secant-Newton). A manageable slice, rather than the whole, of each example problem was timed on both the Alliant FX/8 and the Convex C240. An attempt was made to capture the flavor of each example problem in the slices so that their natures are mirrored in the timings. All speedup comparisons of element computation algorithms are on a per call basis, while actual timings and total time speedups reflect differing numbers of linear preconditioned conjugate gradient (LPCG) and (nonlinear) equilibrium iterations.

Comparisons of nonlinear solution algorithms were performed on the Convex C240 because of the difficulty associated with using the Alliant FX/8 in multiuser mode: In general, the example problems are too large to run in multiuser mode and must execute in single user mode. Larger slices of each example problem were normally required to suitably test the nonlinear solution algorithms, due to the need to proceed further to reach telling nonlinear behaviour. Thus, it was not feasible to procure enough Alliant FX/8 single user time to adequately treat the nonlinear solution comparisons, so only the Convex C240, which could easily handle the example problems in multiuser mode, was used in this phase of the research. Only example problems containing meshes with a three dimensional aspect were used in the nonlinear solution comparisons, as the effect of dimensionality on behaviour is pertinent primarily to the linear solution algorithms. The nonlinear solution comparisons employed the LPCG solver with diagonal (diag) or element-by-element (ebe) preconditioning. This was done because the LPCG solver is a primary focus of this research, so that its performance in conjunction with the nonlinear solution algorithms is of interest, and because the nonlinear preconditioned conjugate gradient (NLPCG) algorithm is dependent on the use of LPCG so that assembly of the global dynamic tangent stiffness matrix is not necessary. The most effective preconditioner in a given example problem was used. The Sorenson form of the search direction multiplier was used in all NLPCG analyses.

All timings were made using the FORTRAN utility etime. As such, timings include time spent in the system executing on behalf of the user and reflect the amount of virtual memory paging. This is in contrast to timings obtained from a profile produced by the utility

gprof, where system time is not included. It is important to include the system time because this becomes significant in larger problems. Also, use of profiling increases the executable size of the program and makes it more difficult to run.

In each of the example problems, the relationship between the condition number of the linear elastic dynamic tangent stiffness matrix and the time step size is explored. The condition number, defined as the product of the norm of a matrix and the norm of its inverse, is estimated using the LINPACK routine DPBCO. The condition number defined on $\| \cdot \|_2$ [34], the ratio of the maximum and minimum eigenvalues of the matrix, is used in Section 2.2. The condition number estimated with DPBCO is defined on $\| \cdot \|_1$ [23]. The details of how this estimation is made is given in the LINPACK User's Guide [23].

Periodically, mention is made of the square root of the number of active degrees of freedom ($\sqrt{\text{ndof}}$) in conjunction with the convergence characteristics of LPCG. This is done because it has been observed that a good preconditioner can often lead to convergence after this number of iterations, a convergence rate which is considered rapid [34].

For the purpose of comparison, the critical time step size for the second central difference explicit time integration method for each example problem is mentioned in conjunction with the study of the effect of the time step size on the convergence characteristics of LPCG presented in Section 4.2. This critical time step size is estimated using the relationship

$$\Delta t_{cr} = \frac{\Delta x}{c} \quad (4.1-1)$$

where Δx is the smallest dimension of any element in the structure and c is the velocity of a dilatational wave in an unbounded medium, defined as [50]

$$c = \sqrt{\frac{E(1-\nu)}{\rho(1+\nu)(1-2\nu)}} \quad (4.1-2)$$

E is the modulus of elasticity, ν is Poisson's ratio, and ρ is the mass density. This is known as the Courant stability criterion and is used in explicit dynamics codes such as SPECTROM-331 [47].

4.1 PROBLEM DESCRIPTIONS

In this section, descriptions of the problem slices testing the linear solution and element computation algorithms and the nonlinear solution algorithms in each example problem are provided.

In the problem slices addressed in this section, two key considerations are constant. First, unless otherwise specified, LPCG iterations terminate when the linear residual load test falls below a tolerance of 0.01%. In this study, no attempt was made to exhaustively explore the effects of varying the LPCG tolerance. Second, where applicable, the same test and tolerance used in the engineering analysis of each example problem was used to terminate MNR iterations in the linear solution/element computation algorithm problem slices.

Fig. 4.1-1 summarizes some of the characteristic of the example problems described below. The categories considered are the half band width and the number of elements, nodes, and active degrees of freedom after constraint; the mesh topology and key attributes; and the predominant type of nonlinearity.

4.1.1 BAR EXTENSION PROBLEM

The bar extension problem (1DBE) is an example of a one dimensional mesh employing linear elements, where all of the elements are arrayed in a single dimension. This type of problem is tested to benchmark the behaviour of the algorithms when three dimensional elements are applied to one dimensional problems. The mesh, material properties, and loading history for the engineering analysis of 1DBE are given in Section 3.1. In this analysis, significant nonlinearity occurred in 72 of the 80 time steps. Even with the tangent stiffness updated every iteration, the solution often required upwards of 10 equilibrium iterations to converge to a tolerance of 0.01% using the residual load test. Without such frequent updates, the solution often did not converge at all.

For reasons to be made evident in Section 4.2, the linear solution and element computation algorithm timing studies of 1DBE are performed with a time step of 0.1 milliseconds, as opposed to the time step of 1.0 milliseconds used in Section 3.1. The loading history of Fig 3.1-2 is redefined accordingly, with the load reaching the constant, maximum value in 1.0 rather than 10.0 milliseconds. All other parameters remain the same. The problem slice used for the comparison of the linear solution and element computation algorithms consists of twenty time steps, ten up to the maximum load and ten along the loading flat top. This problem slice was small enough to execute feasibly on the Alliant FX/8 yet large enough to accurately represent the nonlinear behaviour of the problem. For the first 8 time steps, while the material behaviour is elastic, the tangent stiffness was only updated before the first iteration. For the remaining steps, the tangent stiffness was updated on every iteration to reflect the engineering analysis.

	# elem.	# nodes	hbw	active dof	mesh topology	mesh attributes	nonlinearity
1DBE	512	2052	24	6144	one dimensional	linear elements	material
2DSB	508	3799	381	7372	two dimensional	center hole; quadratic elements	material and localized geometric
3DBB	2440	3216	438	9040	significantly three dimensional	crack flaw; linear elements	material and localized geometric
3DCS	768	4049	1197	11168	significantly three dimensional	curved shell geometry; quadratic elements	basically linear
3DL12	1728	2197	552	6084	three dimensional	linear elements	material and geometric
3DL18	5832	6859	1146	19494	three dimensional	linear elements	material and geometric
3DQ8	512	2673	1005	7344	three dimensional	quadratic elements	material and geometric
3DQ10	1000	4961	1491	13860	three dimensional	quadratic elements	material and geometric

Fig. (4.1-1), example problem characteristics

4.1.2 SHEAR BAND PROBLEM

The shear band problem (2DSB) is defined by a two dimensional mesh of quadratic elements, where all of the elements are arranged in a plane. A problem in this category is analyzed to establish the response of the algorithms when three dimensional elements are applied to two dimensional problems. The mesh, material properties, and loading history for the engineering analysis of 2DSB are given in Section 3.2. In the engineering analysis, significant material nonlinearity from the shear band appeared almost immediately and continued throughout the 100 time steps solved. Similar to 1DBE, during most of the analysis tangent stiffness updates were required on every iteration to insure convergence of the solution, again to a tolerance of 0.01% in the residual load test.

For the comparison of the linear solution and element computation algorithms, only two time steps were solved in order to limit the execution time to a manageable amount on the Alliant FX/8. To include the effects of plastic flow, the end load for the first time step was set to 4 MN and for the second time step was increased to 6 MN. The time step size employed equaled 0.1 milliseconds, the same size specified in the engineering analysis. With this loading, significant material nonlinearity appeared on the first step, and to handle this and reflect the bulk of the engineering analysis, the tangent stiffness was updated before every iteration of both time steps.

4.1.3 3D BEND BAR PROBLEM

The 3D bend bar problem (3DBB) is characterized by a mesh of linear elements with a significant three dimensional topology and a flawed crack tip geometry. The mesh, material properties, and loading history for the engineering analysis of the 3DBB are presented in Section 3.3. Before performing the actual engineering analysis, the solution of a few preliminary time steps indicated that in the range of nonlinear behaviour, at least two and possibly three tangent stiffness updates were required to make certain that adequate convergence was maintained. Again, the tolerance of the residual load test for MNR iterations was 0.01%. Further, if the tangent stiffness was updated before every iteration, usually only three iterations were performed. To ensure that a runaway solution did not occur and because it did not appear to be overly costly, the tangent stiffness was updated before every iteration in the non-linear range of the full analysis.

The linear solution and element computation algorithms were tested by solving just two time steps, again to render the execution time practical on the Alliant FX/8. The first time step encompasses the first 5 time steps of Fig. 3.3-2, and the second time step corresponds to the sixth. This is done because the first noticeable signs of yielding about the crack

tip occur on the sixth time step of Fig. 3.3-2 and the attempt was made to incorporate this influence. Accordingly, the first time step size is 1.07712×10^{-4} seconds and the second is 0.35×10^{-4} seconds. To represent the frequent tangent stiffness updates of the engineering analysis, the tangent stiffness in the problem slice is updated before every iteration. On the second step of the problem slice, enough of the elements about the crack yielded to reflect the material nonlinearity of the engineering analysis and to produce a representative nonlinear convergence rate. Although the change in time step size would undoubtedly produce a spurious impact effect if one was interested in an accurate stress analysis, for the purposes of timing the problem slice is adequate.

The problem slice utilized in the comparison of the nonlinear solution algorithms conforms to the loading history of Fig. 3.3-2 up to the point where the loading flat top is reached in order to reflect the plastic zone about the crack tip. This point is reached in either 10 equal time steps of 0.03457712 milliseconds, or in one time step of 0.3457712 milliseconds. MNR, secant-Newton, and NLPCG runs were made with the tangent stiffness updated before every iteration, and with the tangent stiffness updated before just the first two iterations for time steps in the range of significant nonlinear behaviour. The residual load test was used for the equilibrium iterations of all the algorithms, with a tolerance of 0.01%. For those NLPCG runs including a line search, the dot product test with a tolerance varying from 10% to 0.01% was employed. The LPCG convergence tolerance was varied from 1% to 0.01%.

4.1.4 CYLINDRICAL SHELL PROBLEM

The mesh of the cylindrical shell problem (3DCS) has a significant three dimensional topology modeled by quadratic elements and a conditioning degraded by combined membrane and bending shell behaviour. The mesh, material properties, and loading history for the engineering analysis of 3DCS are presented in Section 3.4. Since the material in 3DCS was maintained in the elastic range, no difficulties in convergence were encountered and the tangent stiffness was updated only before the first iteration of each time step.

Evaluation of the linear solution and element computation algorithms is accomplished by the solution of the first two time steps in the loading history of Fig. 3.4-2. This problem slice resulted in total execution times on par with previous problems and, as the entire solution is in the materially elastic range, even one time step would adequately depict the problem behaviour. The time step size used in the problem slice is 0.025 milliseconds, as specified in Section 3.4. The tolerance used in the residual load test for MNR iterations here and in the engineering analysis was 0.01%.

Actually, given the absence of nonlinearity in this problem, it could have been analyzed as a linear problem. Nonlinearity was included so that it would be reflected in the element computation algorithm timings. 3DCS was not used to test the nonlinear solution algorithms because of its linear nature.

4.1.5 3D LINEAR FINITE EXTENSION PROBLEM

In the previous example problems, meshes of varying dimensionality have been used in conjunction with the three dimensional elements of this research. As such, problems described by a purely three dimensional mesh topology employing both linear and quadratic isoparametric elements are desired as a benchmark against which to measure the previous analyses. Further, these problems should exhibit tangible effects of geometric and material nonlinearities so that a fair test of the nonlinear solution algorithms can be made. In Section 3.5, the finite extension problem was analyzed for the purpose of evaluating the finite strain plasticity models. The results showed that, under loading control, there was poor convergence of the equilibrium iterations for the Newton-Raphson nonlinear solution algorithm due to the debilitating effects of coupled geometric and material nonlinearity. Consequently, the finite extension problem modeled with a cubic mesh is adopted as the benchmark problem. The 3D linear finite extension problem (3DLFE) is modeled with two meshes of different size. The first mesh (3DL12) consists of 2197 nodes and 1728 linear elements arrayed in a 12 by 12 by 12 division of a unit cube. This mesh is analyzed on both the Convex C240 and the Alliant FX/8. The second mesh (3DL18) consists of 6859 nodes and 5832 elements arranged with 18 elements to a side. This second mesh is far too large to be studied on the Alliant, so that it is considered only on the Convex. Both meshes are constrained in a plane stress format, allowing freedom of expansion and contraction in the directions orthogonal to the loading. All told, there remain after constraint 6084 active degrees of freedom for 3DL12 and 19494 for 3DL18. The modulus of elasticity is 30000 ksi, Poisson's ratio is 0.3, and the yield stress is 30 ksi. The meshes are axially loaded to produce a uniform axial stress throughout. In order to maintain the behaviour observed in Section 3.5 and also to exercise the computational aspects associated with a dynamic analysis, in general the problem was analyzed with a mass density of 0.0 and a time step of 1.0. In this way, the problem behaves statically while performing the computations necessary for a dynamic analysis.

In order to limit the execution time of the problem slice comparing the linear solution and element computation algorithms, the load of the first step was set to 30.5 ksi and either four (using mesh 3DL12) or two (using mesh 3DL18) equilibrium iterations attempting to balance this load were performed. This allows the effects of the nonlinearity to be felt while permitting the slower element computation algorithms and the sequential application to fin-

ish in a reasonable time. As a consequence of the extreme nonlinearity of the problem, the tangent stiffness is updated before every iteration. No convergence test is necessary for the MNR iterations, as the solution is arbitrarily terminated after four iterations.

So that the full flavor of the nonlinearity of the problem is experienced, the problem slice testing the nonlinear solution algorithms encompasses either 5 or 10 load steps: two steps of 15 ksi each, then either three steps of 4.0 ksi or eight steps of 1.5 ksi. In the range of nonlinear behaviour, the tangent stiffness was updated before every iteration. The tolerance used in the residual load test for the equilibrium iterations in the nonlinear solution problem slice was 0.1%; the dot product test for line search iterations in NLPCG was either 1% or 0.01%. The LPCG convergence tolerance was again varied from 1% to 0.01%. Only 3DL12 was used in this study.

4.1.6 3D QUADRATIC FINITE EXTENSION PROBLEM

The 3D quadratic finite extension problem (3DQFE) is also analyzed using two separate meshes. The first mesh (3DQ8) contains 2673 nodes and 512 quadratic elements in a 8 by 8 by 8 array. The second mesh (3DQ10) contains 4961 nodes and 1000 quadratic elements in a 10 by 10 by 10 array. Again, plane stress boundary conditions prevail, leaving 7344 active degrees of freedom for 3DQ8 and 13860 for 3DQ10. The material properties, loading, mass density, and time step size are as specified above. 3DQ8 is examined on both the Alliant FX/8 and the Convex C240, and 3DQ10 is studied on only the Convex because of its size.

The linear solution and element computation problem slice is identical to that for 3DLFE, except that only two equilibrium iterations are performed in order to limit the anticipated execution time of the sequential application.

The nonlinear solution problem slice is also identical to that of the linear problem, except that the slice is reduced to three or five load steps – two of 15 ksi and one of 4.5 ksi or three of 1.5 ksi – considering the much greater computational effort required by the quadratic mesh and the desire to limit the execution time so that the analysis could be realistically completed in one night. This was done to avoid times of heavy machine load and possible downtimes. Accordingly, only 3DQ8 was used in this study.

4.2 LINEAR SOLUTION ALGORITHMS

In this section the results of the linear solution algorithm comparisons for all of the example problems are presented. The results are organized by memory requirements, comparisons of LPCG and the direct solver, and a study of the convergence characteristics of LPCG. A summary is provided at the end of the section.

4.2.1 MEMORY REQUIREMENTS

Central to the question of which linear solver is most preferable are the memory requirements of each approach. The basic memory requirements for the solution of a linear system of equations and the memory consumed by the main common area for all of the example problems are listed in Fig. 4.2-2. The definition of the basic memory requirements for the direct solver, LPCG with ebe preconditioning (LPCG w/ ebe), and LPCG with diagonal preconditioning (LPCG w/ diag) are displayed in Fig. 4.2-1. The memory needed by the main common area is taken from the version of NLD FEP incorporating the element computation algorithm BEC on the Alliant FX/8. All other versions have main common sizes with trivial differences.

required for all linear solvers:

element tangent stiffness matrices in upper triangular vector form

element tangent mass matrices in upper triangular vector form

required for direct solver:

assembled global (dynamic) tangent stiffness matrix in banded form

required for LPCG w/ ebe:

factored element preconditioning matrices in upper triangular vector form

required for LPCG w/ diag:

global preconditioning vector

Fig. (4.2-1), definition of basic memory requirements for linear solution

Fig. 4.2-2 reveals that for 1DBE, the memory requirements for the direct solver are slightly less than for LPCG w/ ebe and only marginally more than LPCG w/ diag. This is due to the small half band width of the problem, which equals 24. This half band width remains constant as the number of elements in the problem increases. Thus, as the size of the problem increases, the direct solver memory needs will always nearly equal those of LPCG w/ ebe and minimally exceed those of LPCG w/ diag. For this reason alone, LPCG is not ideally suited to one dimensional problems.

The memory requirements are far greater for 2DSB, the consequence of using quadratic isoparametric elements and a much increased half band width (381). Because of

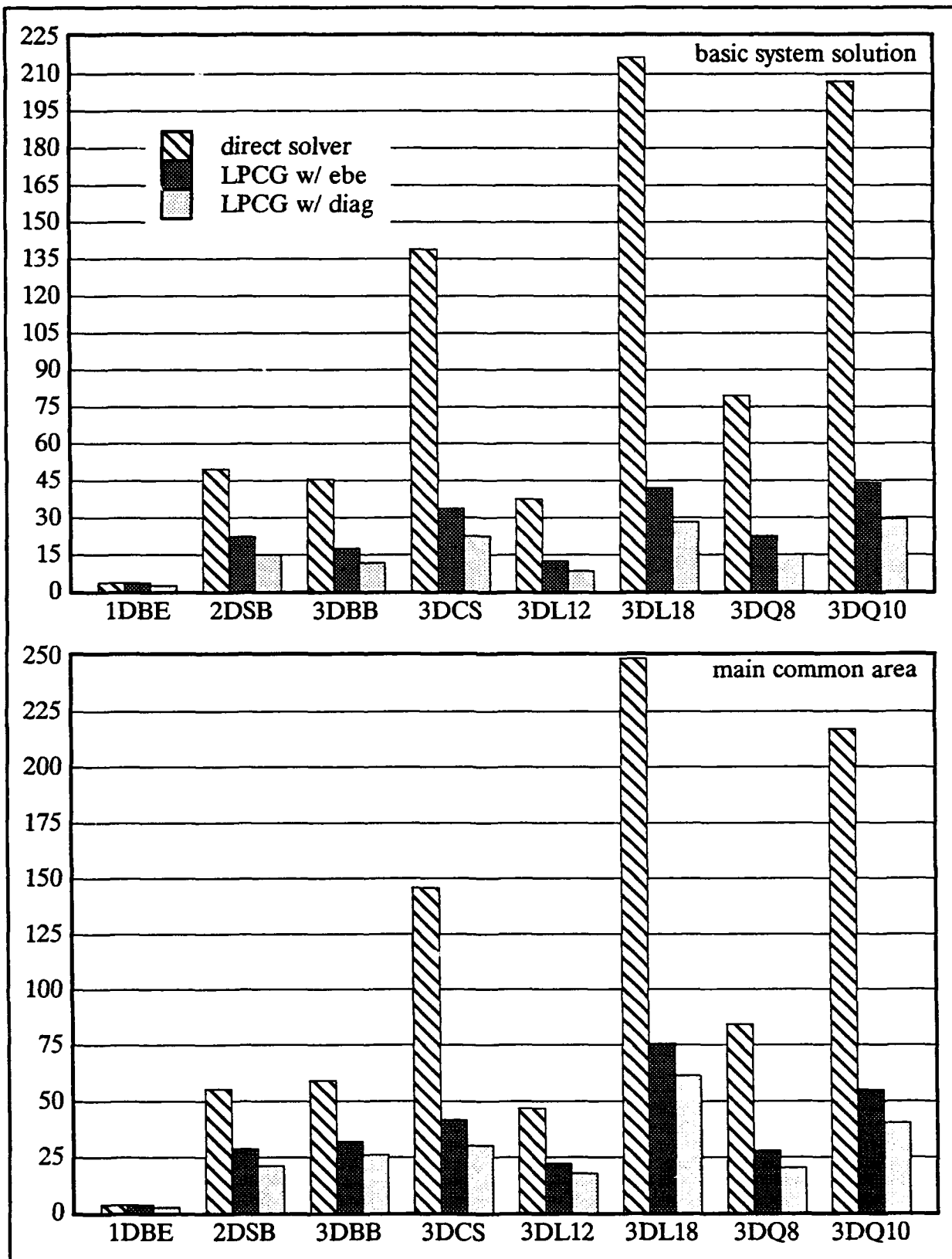


Fig. (4.2-2), memory requirements for all example problems, in megabytes

this larger half band width, there is a significant difference in the requirements for the direct solver and the LPCG solvers. For 2DSB, there is a definite advantage in memory to using LPCG in lieu of the direct solver, especially for the diagonal preconditioner. In this respect, LPCG is far more suited to two dimensional than one dimensional problems.

The memory demands of 3DBB are quite similar to those of 2DSB. The 3DBB half band width is slightly larger (438), but fewer structural nodes (3216 to 3799) makes the 3DBB direct solver system solution memory somewhat smaller. While linear rather than quadratic elements are used, the memory advantage incurred is partially offset by the much larger number of elements (2440) so that the system solution memory of the LPCG solvers remains in nearly the same proportion to the direct solver as it was in 2DSB. The 3DBB main common memory for each linear solver is significantly larger in comparison to the system solution memory than it was for either 1DBE or 2DSB. This is again the result of the large number of elements in 3DBB, which increases the storage necessary for quantities such as the global stresses and strains. Still, Fig. 4.2-2 shows that LPCG has a definite memory advantage for 3DBB.

For 3DCS, an overwhelming memory advantage exists for the LPCG solvers, with LPCG w/ ebe requiring less than one-fourth and LPCG w/ diag less than one-sixth the basic system solution memory. This advantage stems from the use of quadratic elements and the three dimensional aspect of the mesh; 3DCS possesses the relatively large half band width of 1197 and over 4000 structural nodes, which greatly increases the memory requirements of the global dynamic tangent stiffness matrix. The LPCG solvers benefit from the relatively small number of elements compared to the number of nodes, characteristic of meshes employing quadratic elements, which offsets the increase in quadratic element data storage and limits the memory necessary for the LPCG solvers.

Due to a larger half band width (552) but fewer elements (1728) and structural nodes (2197), there is similar discrepancy in Fig. 4.2-2 between the 3DL12 memory requirements of the direct solver and the LPCG solvers to that observed in 3DBB. The number of elements in 3DL12 is still sufficient to cause a significant amount of memory to exist beyond the scope of the linear solution requirements, due to the storage necessary for such quantities as the stress and strain vectors. This is especially true for LPCG w/ diag, where the basic system solution requirements are less than half the main common memory. The memory advantage of the LPCG solvers is greatly accentuated as the number of elements increases, such as in 3DL18. The half band width (1146) is more than doubled and the number of structural nodes (6859) is more than tripled, contributing to a nearly a 175 megabyte (MB) difference between the basic system solution requirements of the direct solver and LPCG w/ ebe and

almost 200 MB between the direct solver and LPCG w/ diag. Because of the large number of elements (5832), the main common memory of the various solvers exceeds the basic system solution requirements by more than 30 MB.

Fig. 4.2-2 indicates a significant memory advantage for the LPCG solvers in 3DQ8 and a tremendous memory advantage for 3DQ10, similar to that seen for 3DL18. In fact, as the number of elements less than doubles between 3DQ8 and 3DQ10, the discrepancy between the basic system solution requirements of the direct solver and those of the LPCG solvers nearly triples. Likewise, as the number of elements increases by a factor less than 4 between 3DL12 and 3DL18, the discrepancy multiplies by a factor of nearly 7. Thus, for a cubic mesh containing either linear or quadratic elements, the memory advantage of LPCG increases geometrically with increasing mesh size.

4.2.2 LPCG VERSUS DIRECT SOLVER

Timing data comparing the use of the direct solver and LPCG with ebe and diagonal preconditioning is displayed in Figs. 4.2-3 to 4.2-8. Bar charts for each of the example problems are presented corresponding to the system factorization, system back substitution, total execution time, the percentage of the total execution spent in system solution, the speedups versus the sequential application, and the relative speedups. The data pertains to the BEC implementation of NLD FEP on the Alliant FX/8 and the BEV version on the Convex C240. System factorization for LPCG w/ ebe entails Crout factorization and Winget regularization, while for LPCG w/ diag the diagonal vector is merely assembled. Thus, the system factorization times for LPCG w/ diag are not even included in Fig. 4.2-3 as they are very small. For the direct solver the system factorization is the Cholesky decomposition. System back substitution for LPCG w/ ebe involves element forward reduction and back substitution and the vector operations endemic to LPCG, including the matrix-vector product necessary for the determination of the step length. For LPCG w/ diag inversion of the diagonal vector replaces element forward reduction and back substitution. The direct solver system back substitution combines the vector reduction and back substitution using the Cholesky factors. System solution times combine system factorization and system back substitution. Speedups versus sequential application are obtained for each linear solver from the sequential application of that solver. Relative speedups are computed with respect to the slowest linear solver for a given example problem. BEC is chosen to represent the Alliant FX/8 because it is the fastest implementation on that machine, as is discussed more fully in Section 4.3.

In Fig. 4.2-7, the speedups on the Convex C240 are seen to be in general reduced from those on the Alliant FX/8. This is due in large part to the absence of applied concurrency

and the greater sequential computational speed of the Convex C240. Also, the direct solver speedups on the Convex are generally greater than those of the LPCG solvers. As the efficiency of the LPCG solvers hinges primarily on the efficiency of the element computation algorithms, this implies that the efficiency of Convex BEV is not on par with the direct solver operations. On the Alliant FX/8, the opposite is generally true, indicating that BEC takes good advantage of the architecture of the Alliant. The lone exception is 2DSB, where the direct solver shows a sequential speedup of approximately 18. It appears that the dimensions of the 2DSB dynamic tangent stiffness matrix are highly conducive to the direct solver operations on the Alliant.

Comparing Figs. 4.2-5 and 4.2-6, one can observe that the relative total execution times of the direct solver and the LPCG solvers in each example problem is almost invariably mirrored in the percentages of total execution time spent in system solution. This indicates that the solution of each problem is dominated by the solution of the linear system of equations and that the linear solver which is most efficient in terms of percentage will provide the fastest solution. From Fig. 4.2-6, the percentage exhibited by the fastest linear solver in the example problems is generally 70-80%. The notable exceptions are 1DBE and 3DQ8 (as well as 3DQ10 on the Convex). For 1DBE, the direct solver analysis exhibits an abnormally low percentage, just over 20% for both machines, which is less than half that exhibited by the LPCG analyses. The advantage of the direct solver lies in the very small half band width of the problem; relatively little effort is required for system factorization. For 3DQ8 (and 3DQ10), the high percentages of nearly 90% or more results from the rather poor convergence rates of both preconditioners. These relatively poor convergence rates are due to the severe non-linearity of the problem, as is discussed later on.

Fig. 4.2-5 and 4.2-8 show that for those example problems exhibiting a significant three dimensional mesh topology LPCG with either ebe or diagonal preconditioning is generally the fastest linear solver on both machines. The principal difference between the Convex and the Alliant results is the degree of the LPCG advantage. This difference exists primarily because the Convex possesses a far greater memory capacity than the Alliant. The smaller Alliant memory capacity leads the direct solver solution to require an amount of paging to and from the swap space that is significantly higher than that necessary on the Convex. Thus, LPCG has a greater advantage on the Alliant. This trend is exemplified in the relative speedups for the 3D finite extension problems. On the Alliant, the LPCG solvers show a relative speedup versus the direct solver of about 3.6 for 3DL12, and LPCG w/ diag is about 5 times faster than the direct solver for 3DQ8. On the Convex, these speedups drop to about 1.5 and 1.3 respectively.

On both machines, for those example problems employing linear isoparametric elements – 1DBE, 3DBB, 3DL12, and 3DL18 – LPCG w/ ebe is faster than LPCG w/ diag for 3DBB, but LPCG w/ diag is nearly equal to or faster than LPCG w/ ebe for the remaining problems. For those problems consisting of quadratic elements – 2DSB, 3DCS, 3DQ8, and 3DQ10 – LPCG w/ diag and LPCG w/ ebe are nearly equal for 2DSB, but LPCG w/ diag is clearly faster for the remaining problems. Thus, it's basically a toss up between the two preconditioners for linear elements, but the diagonal preconditioner appears superior for quadratic elements.

In Figs. 4.2-3 to 4.2-7, direct solver data is absent from the analysis of 3DCS on the Alliant FX/8. This absence is caused by the inability of the Alliant FX/8 to perform the direct solver solution. Unfortunately, the size of this problem was large enough so that the swapping file space was exceeded and the machine crashed during the direct solver solution. Thus, memory considerations not only make the LPCG solvers an attractive alternative for this problem on the Alliant FX/8, they are the only alternative. The greater memory capacity of the Convex C240 allows the direct solver solution to be performed with no particular difficulty, although LPCG w/ diag is about 4.5 times faster than the direct solver. Similarly, the solutions of 3DL18 and 3DQ10 were not even attempted on the Alliant, but were analyzed without difficulty on the Convex. These solutions clearly demonstrate that increasing the cubic mesh size amplifies the execution time advantage of LPCG on the Convex C240. The relative speedups of LPCG w/ diag versus the direct solver increase to 3.9 and 2.4 for 3DL18 and 3DQ10, respectively. Figs. 4.2-3 and 4.2-4 indicate that the direct solver system factorization costs per equilibrium iteration for the linear finite extension problems grow almost 3 times faster than the LPCG w/ diag system back substitution costs, and those of the quadratic finite extension problems increase nearly twice as fast. Coupled with the rates of escalating memory requirements established earlier, for large three dimensional meshes LPCG is a very attractive option.

The timings for 3DBB are based on 5 equilibrium iterations; those of 3DL12 are based on 4. Accordingly, the data in Figs. 4.2-3 and 4.2-4 implies that on both machines 3DL12 requires considerably less system back substitution time and more system factorization time per equilibrium iteration than does 3DBB. This results from a greater half band width in 3DL12 (552 vs. 438) and a smaller number of elements (1728 vs. 2440). Thus, even though the ratio of the ebe preconditioner convergence rate (84 average linear iterations) to the $\sqrt{\text{ndof}}$ (78) is similar to that in 3DBB (108 and 95, respectively) which also employs linear elements, LPCG w/ ebe is more effective relative to the direct solver. This indicates

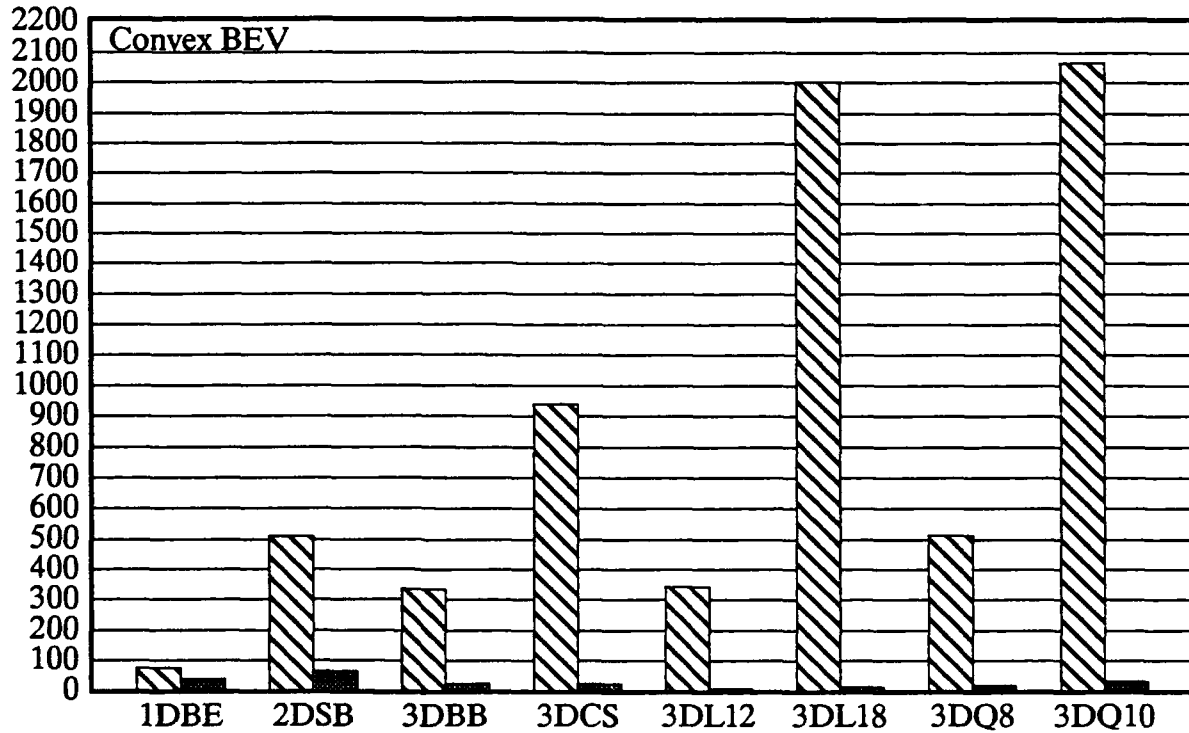
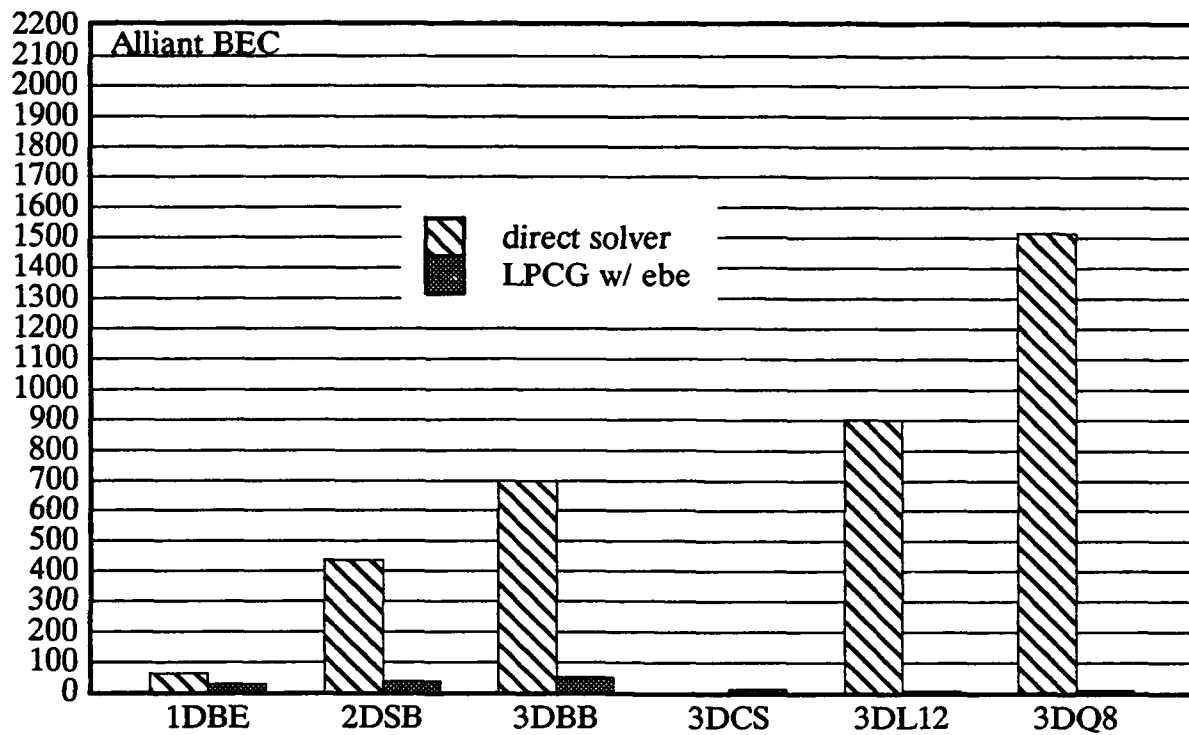


Fig. (4.2-3), system factorization, in CPU seconds

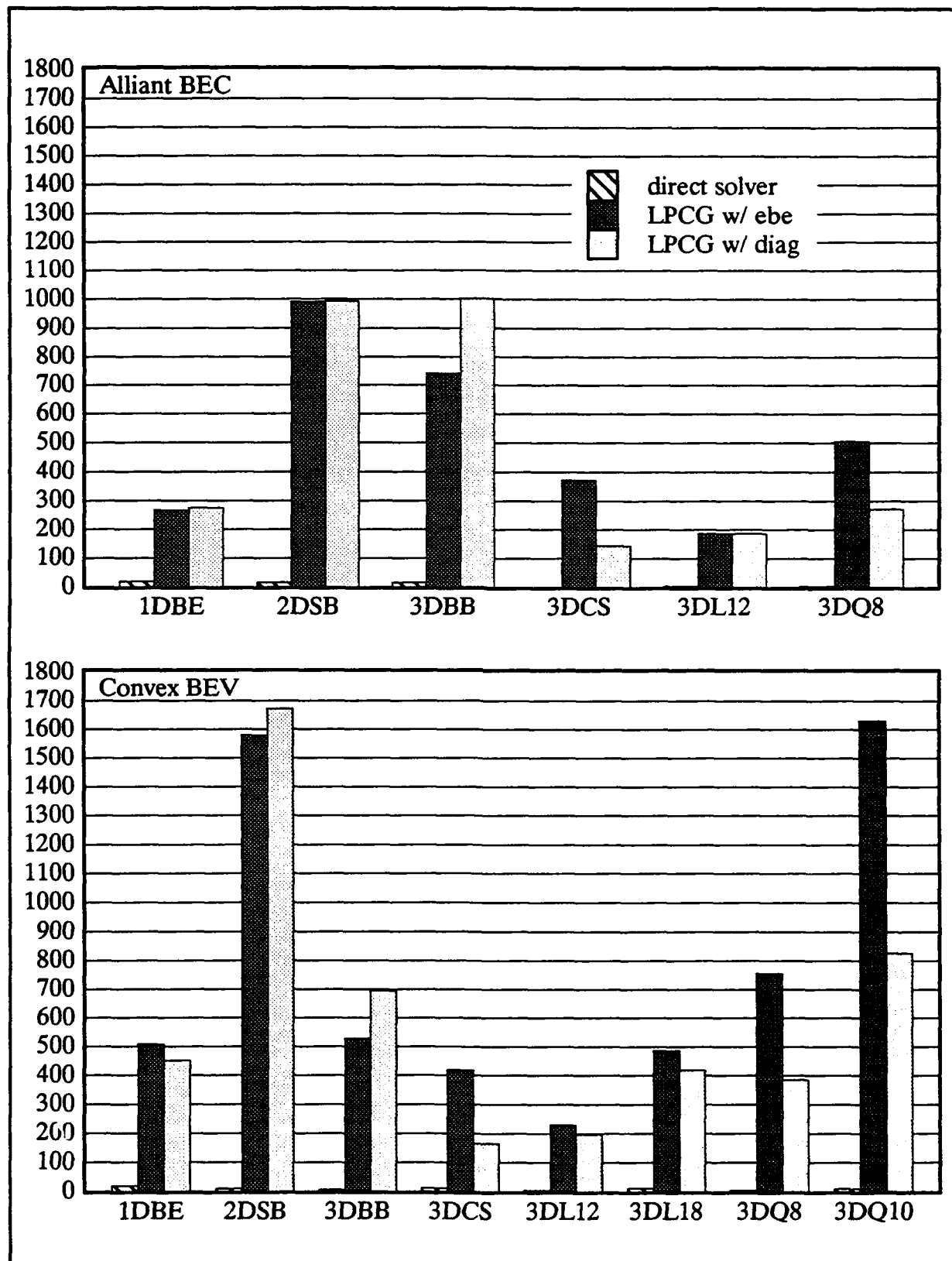


Fig. (4.2-4), system back substitution, in CPU seconds

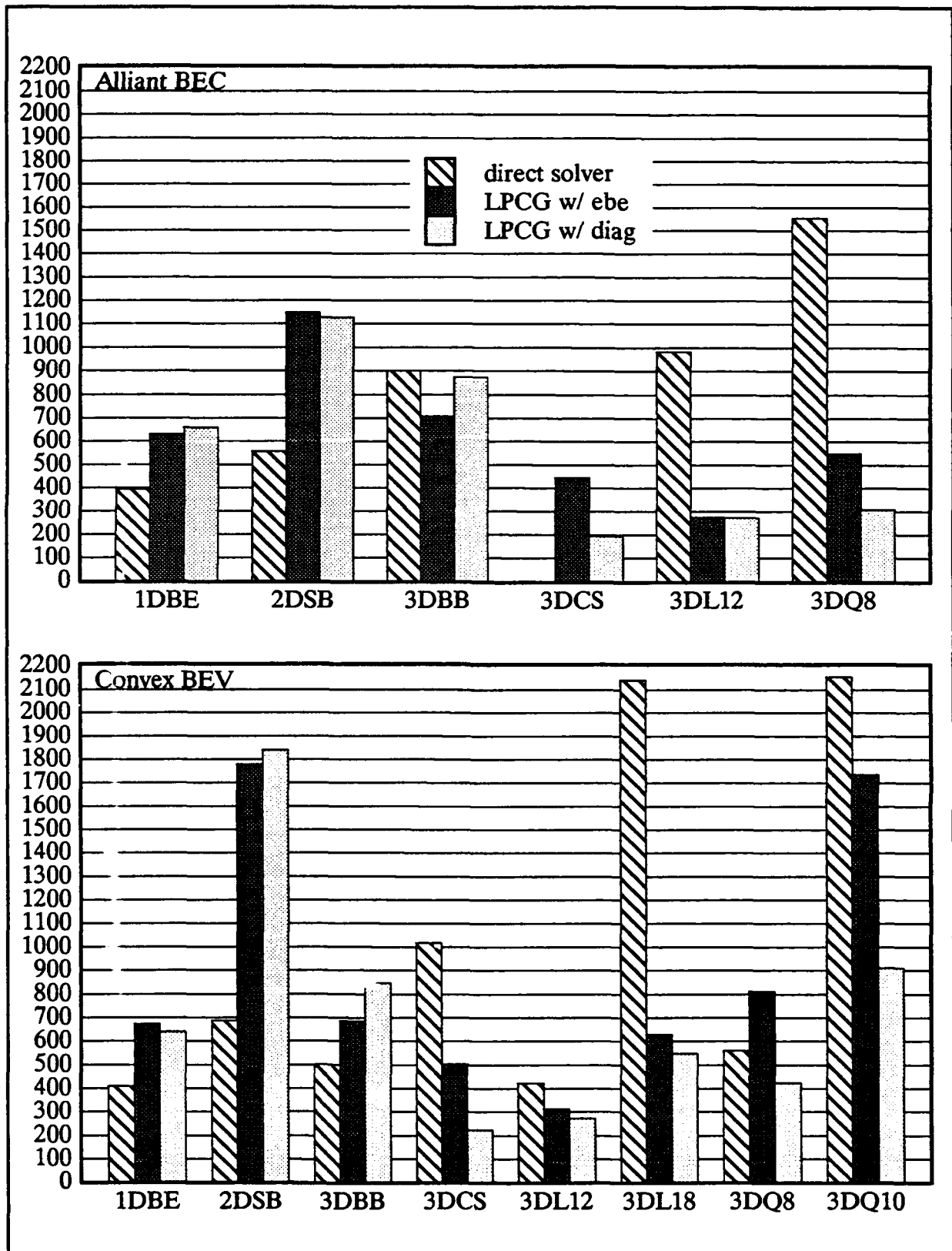


Fig. (4.2-5), total execution time, in CPU seconds

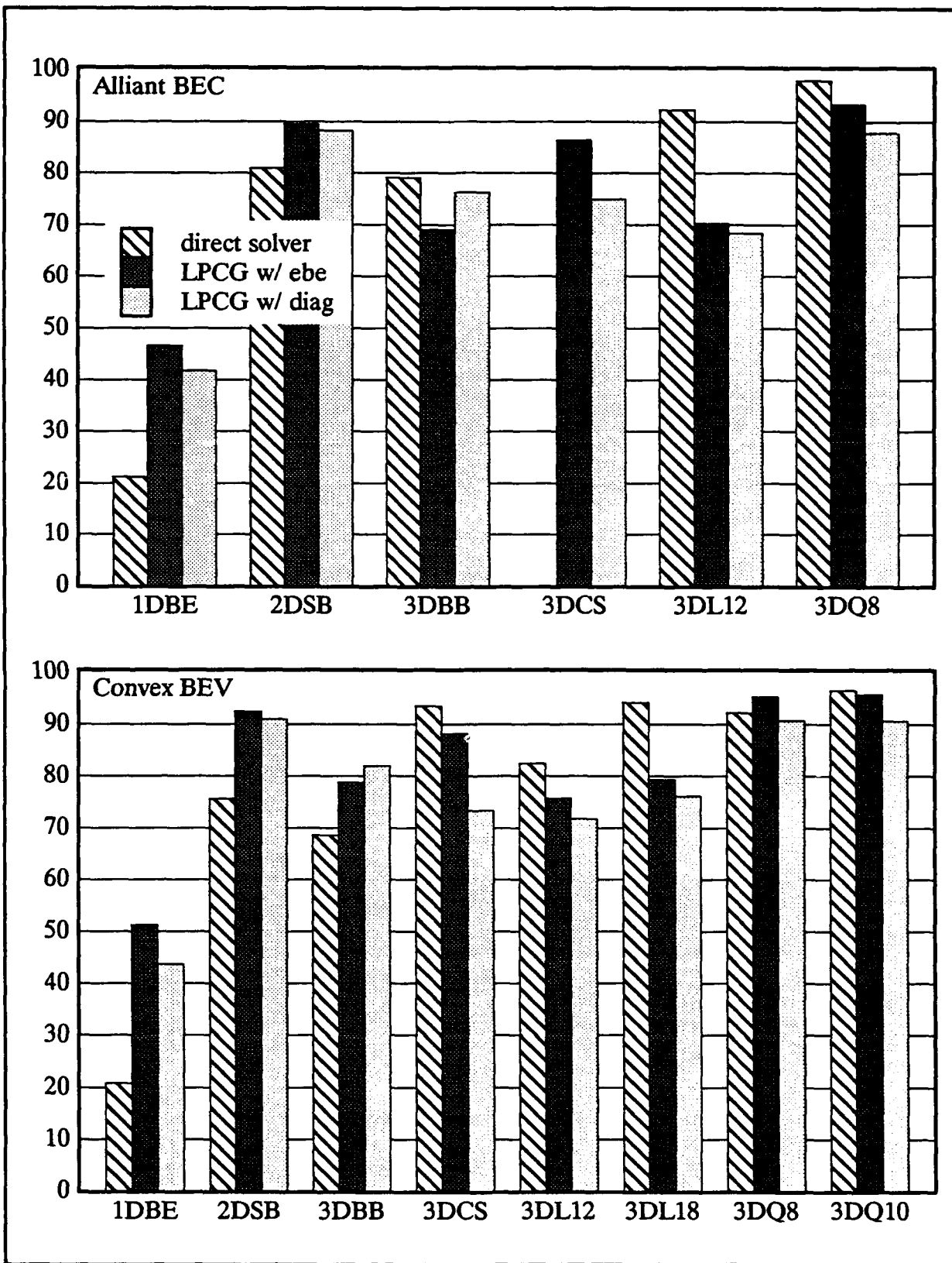


Fig. (4.2-6), % of total execution time spent in system solution

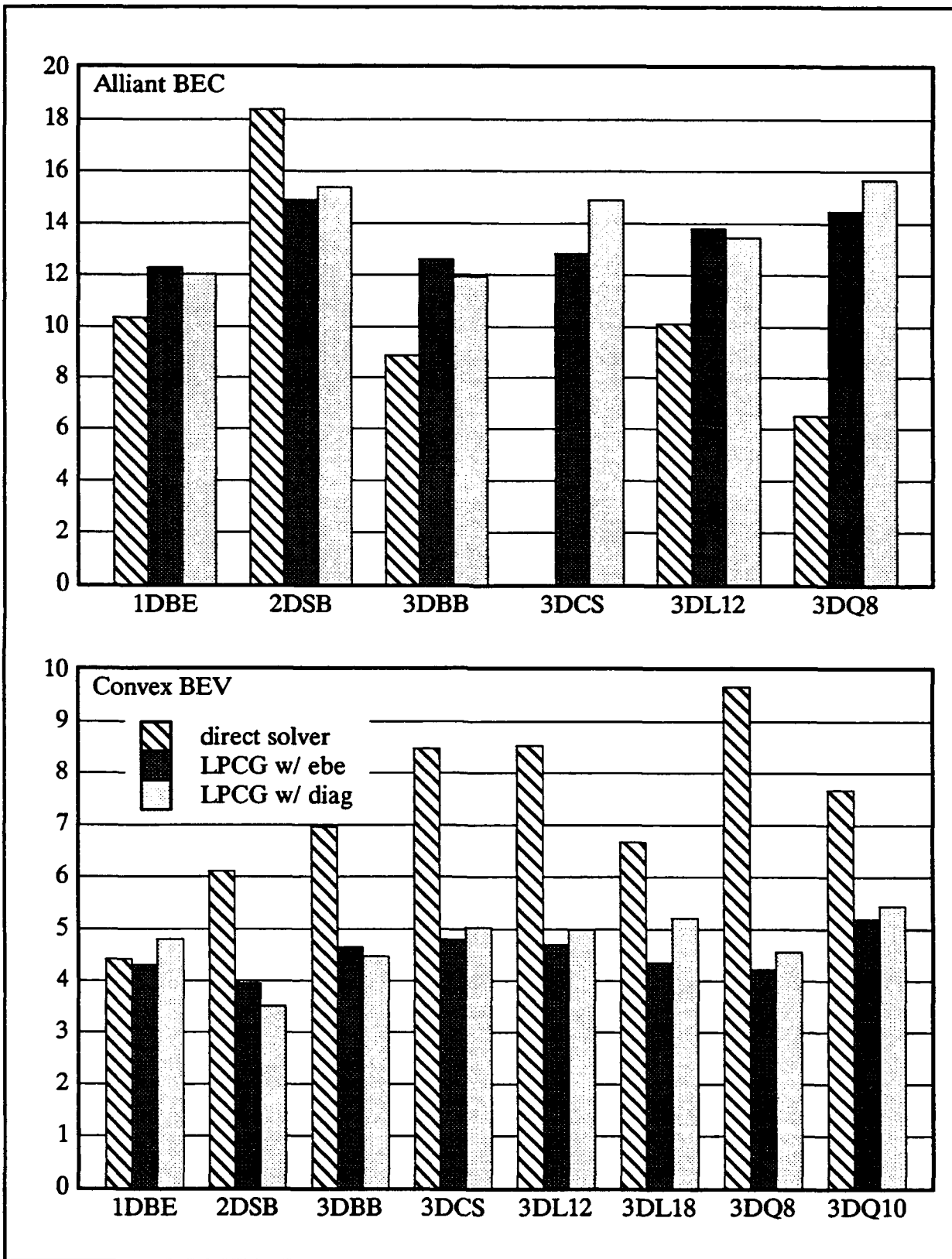


Fig. (4.2-7), speedups vs. sequential application

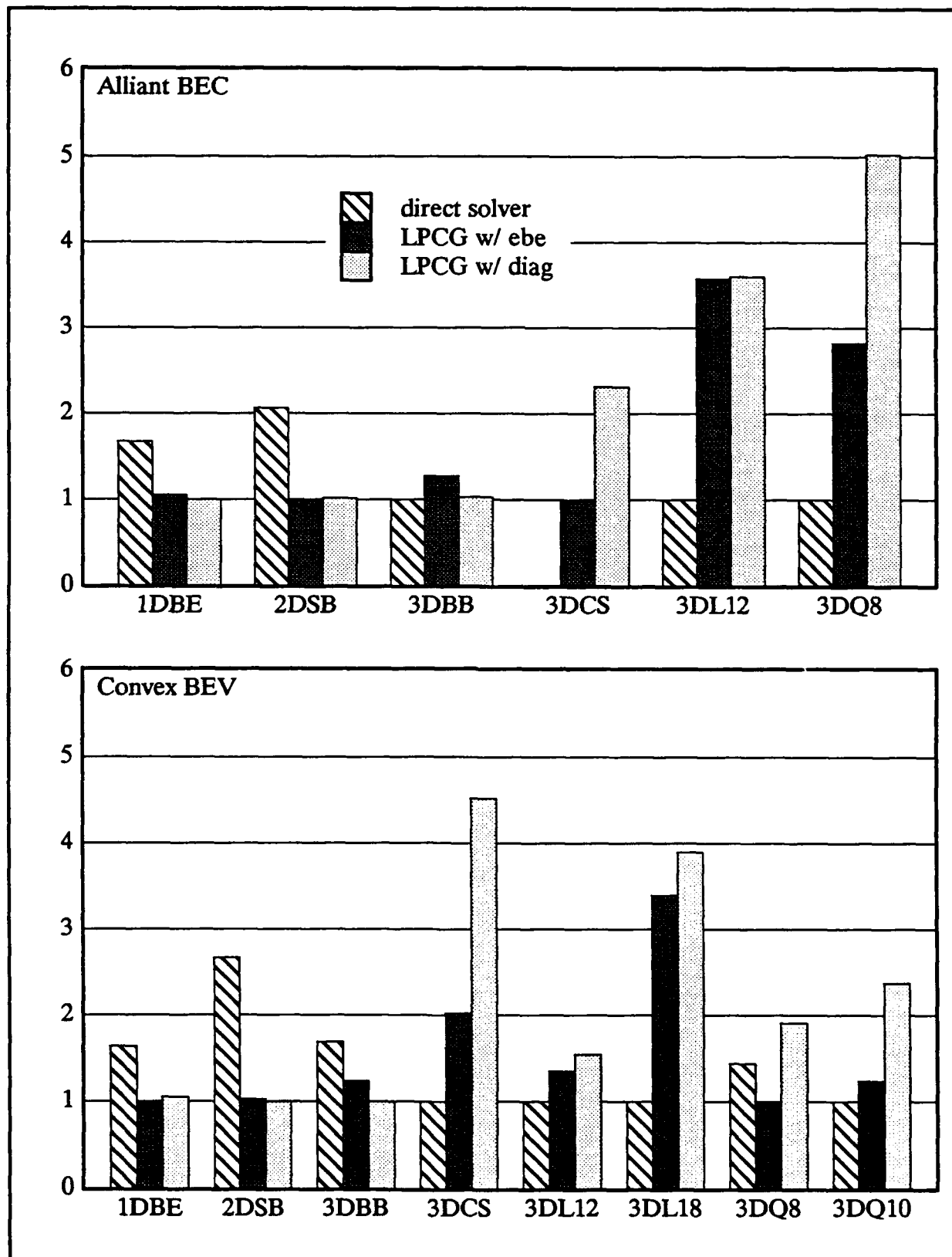


Fig. (4.2-8), relative speedups

that as the mesh topology becomes more three dimensional, for similar convergence rates LPCG gains on the direct solver in total execution time.

From Figs. 4.2-3 and 4.2-4, the comparison of the direct solver and the LPCG algorithms is seen basically to hinge upon the system factorization of the direct solver versus the system back substitution of the LPCG solvers. This is anticipated, as ideally the system factorization associated with the preconditioner should be as small as possible and the system back substitution of the direct solver should be relatively insignificant compared to its factorization. Because the cost of system factorization essentially dominates the direct solver performance, the analyses performed here – which rely on frequent tangent stiffness updates – do not necessarily represent the maximum achievable performance of the direct solver. To maximize this performance the number of tangent stiffness updates, and thus system factorizations, should be minimized. Conversely, to maximize the performance of the LPCG solvers the number of equilibrium iterations, and thus system back substitutions, should essentially be minimized. In this sense, the timings presented here tend to favor LPCG. Accordingly, the trends observed and the judgements made concerning the comparison of LPCG and the direct solver in this section should carry the qualification that they are based on similar numbers of system factorizations and system back substitutions.

The tolerance specified on the LPCG convergence test – 0.01% – used for these timings is reasonably strict and often used to terminate conjugate gradient iterations [39]. However, this tolerance may not lead to optimal LPCG performance. Lessening the LPCG convergence tolerance can lead to significantly fewer linear iterations and thus lower the LPCG system back substitution times. Of course, accuracy is sacrificed so that a greater number of equilibrium iterations may be required as compensation. This consideration is explored for 3DBB, 3DL12, and 3DQ8 in Section 4.4. Thus, while the qualification above is still necessary, it should be supplemented with the proviso that the comparisons in this section are tied to the tolerance applied.

4.2.3 LPCG CONVERGENCE CHARACTERISTICS

The study of the LPCG convergence characteristics is separated into two aspects: the variation of the condition number and the LPCG convergence rate with the size of the time step, and the effect of the example problem nonlinearity and the applied element computation algorithm on the LPCG convergence rate.

4.2.3.1 EFFECT OF TIME STEP SIZE

One very interesting feature of this research is the extent to which the condition number of the dynamic tangent stiffness matrix, and thus the number of LPCG iterations re-

quired for convergence, is a function of the time step size employed. Log-log plots of condition number versus time step size for each example problem are given in Fig. 4.2-9. The condition numbers pertain to the first equilibrium iteration of the first time step, or to the linear elastic dynamic tangent stiffness. The time step sizes employed in the engineering analysis of the example problems are clearly marked. Semi-log plots of LPCG iterations to convergence for both the diagonal and ebe preconditioners versus time step size, again for the first equilibrium iteration of the first time step, are shown in Figs. 4.2-10 to 4.2-15. Also pictured are the time step sizes employed in the LPCG timing runs – identified by the subscript t – and the critical time step for explicit dynamics – identified by the subscript c – for each example problem. The iteration data was obtained from the sequential version of NLD FEP. In all plots, the time step size is varied from 1.0 to 10^{-6} seconds. Convergence is defined as satisfying the linear residual load test to a tolerance of 0.01%. Figs. 4.2-9 to 4.2-15 do not account for influence of significant nonlinearity on the condition numbers and convergence rates, but do furnish a guideline as to the influence of the time step size. Likewise, a smaller tolerance will reduce the number of linear iterations required for convergence, but the basic trends should remain the same. The effects of the nonlinearities seen in the example problems are discussed further on.

1DBE is a problem where the dependence of the condition number on the time step size is pronounced. Fig. 4.2-9 describes virtually a linear relationship and a dramatic decrease in the 1DBE condition number with diminishing time step size. This decrease occurs because as the time step size declines, the influence of the mass matrix increases quadratically. The mass matrix, especially if lumped, is well-conditioned; if the tangent stiffness is ill-conditioned, then the condition of the dynamic tangent stiffness will improve with decreasing time step size.

Since the condition number is so affected by varying time step size, one would expect the LPCG iterations to be similarly affected. This is indeed the case. The performance of the diagonal preconditioner reaches a near flat top at a time step size of 10^{-3} seconds and ceases to degrade significantly for greater time step sizes. The performance of the ebe preconditioner continues to degrade as the time step size increases, and in fact falls below that for the diagonal preconditioner at higher time step sizes. This is probably due to the diagonal dominance of the one dimensional mesh. From Fig. 4.2-10, it is seen that at a time step size of 10^{-4} seconds, the size used in the timing runs, LPCG w/ ebe converges in 20 iterations and LPCG w/ diag converges in 50 iterations. Both of these convergence rates fall below the $\sqrt{\text{ndof}}$. At 10^{-3} seconds, those numbers increase to 186 and 496 respectively. The above decrease in convergence iterations is the reason for employing 10^{-4} seconds in the timing runs. If 10^{-3}

seconds is used, the problem slice described above – which produces a viable duration for timing runs using the direct solver – is far too costly to expeditiously complete all the LPCG runs necessary for this research. In order to evaluate the element computation algorithms – comparisons which don't necessarily depend on the good behaviour of LPCG – and to develop some idea of the relative behaviour of the linear solvers, 10^{-4} seconds was chosen. In the engineering analysis of 1DBE, 10^{-3} seconds was used in Section 3.1 so that the progress of the plastic wave could be readily observed. The required iterations for both the diagonal and ebe preconditioners are about 10 times greater at 10^{-4} seconds than at 10^{-3} seconds, so that approximately the same overall computational effort is required for both time step sizes (a time step size of greater than 10^{-3} seconds could be used without further deterioration of the LPCG convergence rates, but then the plastic wave could not be followed). Thus, no advantage would accrue from using 10^{-4} seconds in the engineering analysis. In order to make the LPCG algorithms reasonably competitive with the direct solver in reality, then for reasons of simulation accuracy a time step size less than or equal to 10^{-4} seconds would be required. However, given that the critical time step of second central difference is about 100 times smaller than the 10^{-3} seconds used in the engineering analysis, the use of LPCG may still allow implicit dynamics to compete with explicit dynamics.

2DSB is another instance where the condition number of the dynamic tangent stiffness matrix is a function of the time step employed. The 2DSB plot of Fig. 4.2-9 does not exhibit the linear relationship characteristic of 1DBE, as the condition number is not as sensitive to the time step size and holds fairly constant until the time step size reaches 10^{-3} seconds before falling off in more or less a linear fashion. Also, the 2DSB condition numbers at larger time step sizes are not as large as they were for 1DBE, the condition number at the time step size of 1.0 seconds reaching approximately 10^5 as opposed to about 10^{10} for 1DBE. Both the insensitivity and the smaller condition numbers at larger time step sizes result from the two dimensional mesh and plane strain conditions of 2DSB. For three dimensional elements, extending the mesh in two dimensions and constraining the third is inherently more well-conditioned than linking elements in just a single dimension and allowing the others to freely oscillate, and because the tangent stiffness is not as diagonally dominant the influence of the mass matrix is not felt until the time step size is much smaller. In the same vein, at smaller time step sizes, the condition numbers for 2DSB are not as small as they are for 1DBE, so that lesser benefits are reaped by employing moderately small time step sizes.

The above discussion is reflected in Fig. 4.2-11. The linear iterations required for both preconditioners remain nearly constant at time step sizes larger than 10^{-3} seconds and drops as the influence of the mass matrix is manifested. Fig. 4.2-11 shows that the conver-

gence rates do not approach the $\sqrt{\text{ndof}}$ until the time step size reaches 10^{-4} seconds, the size used in the 2DSB engineering analysis. Even then, the direct solver was faster in execution time for similar numbers of system factorizations and system back substitutions. This implies that only in those situations where a small time step size is required to maintain the accuracy of the simulation is LPCG competitive with the direct solver. Arbitrarily choosing a small time step size would not necessarily lead to competitive LPCG performance, as one could choose a larger time step size for the direct solver analysis so that the LPCG analysis was outperformed. Considering the data presented for the engineering analysis in Section 3.2, the largest time step size one could employ and still preserve the salient features of the plots would probably be 0.5×10^{-3} seconds. In light of this, the time step size of 10^{-4} seconds actually specified is reasonable, but a smaller time step size that would accelerate the LPCG convergence rate is probably not.

Given that the penalty for using a time step size of 0.5×10^{-3} seconds in Fig. 4.2-11 is only about twice as many LPCG w/ ebe or LPCG w/ diag iterations, and that the critical time step size for second central difference is about 3.45×10^{-6} seconds (about 145 times less), then one might be able to apply this time step size with the result that the implicit solution employing LPCG was faster than the explicit solution. However, the direct solver solution would still be faster.

Fig. 4.2-9 reveals that the 3DBB condition number plot is similar to that of 2DSB. The magnitude of the 3DBB condition numbers are about 20 times greater in the nearly constant range of time step sizes down to 10^{-3} seconds and about 50-100 times greater thereafter. This indicates that 3DBB is more poorly conditioned than 2DSB, but its condition number is somewhat less sensitive to the time step size. It would also lead one to anticipate that the convergence rates of the 3DBB problem would be worse than that of 2DSB.

However, this is definitely not the case. Fig. 4.2-12 shows that the 3DBB convergence rates of each preconditioner are much better. The convergence rates of both preconditioners are more insensitive to the time step size, remaining nearly constant down to 10^{-4} seconds as opposed to 10^{-3} seconds in 2DSB. Even along the flat top, the 3DBB convergence rates compare favorably with the $\sqrt{\text{ndof}}$, especially that of the ebe preconditioner. Thus, the competitive performance of the LPCG w/ ebe observed previously in 3DBB, which were based on time step sizes in the vicinity of 10^{-4} seconds and average convergence rates of 108 linear iterations for the ebe preconditioner and 270 for the diagonal preconditioner, will persist throughout the range of possible time step sizes. Considering that the critical time step size for second central difference is about 2.0×10^{-7} seconds, implicit dynamics using LPCG w/

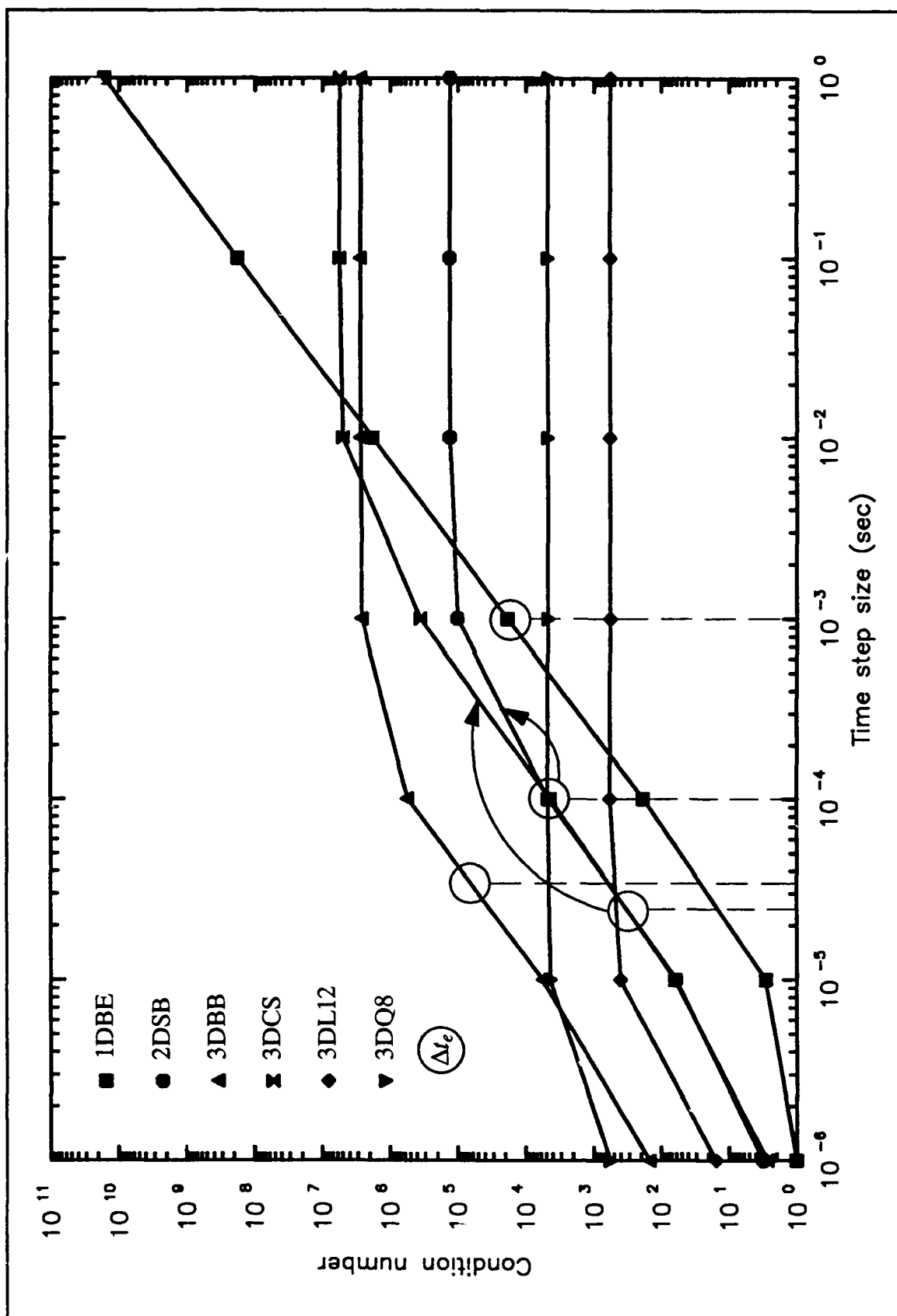


Fig. (4.2-9), condition number versus time step size for all example problems

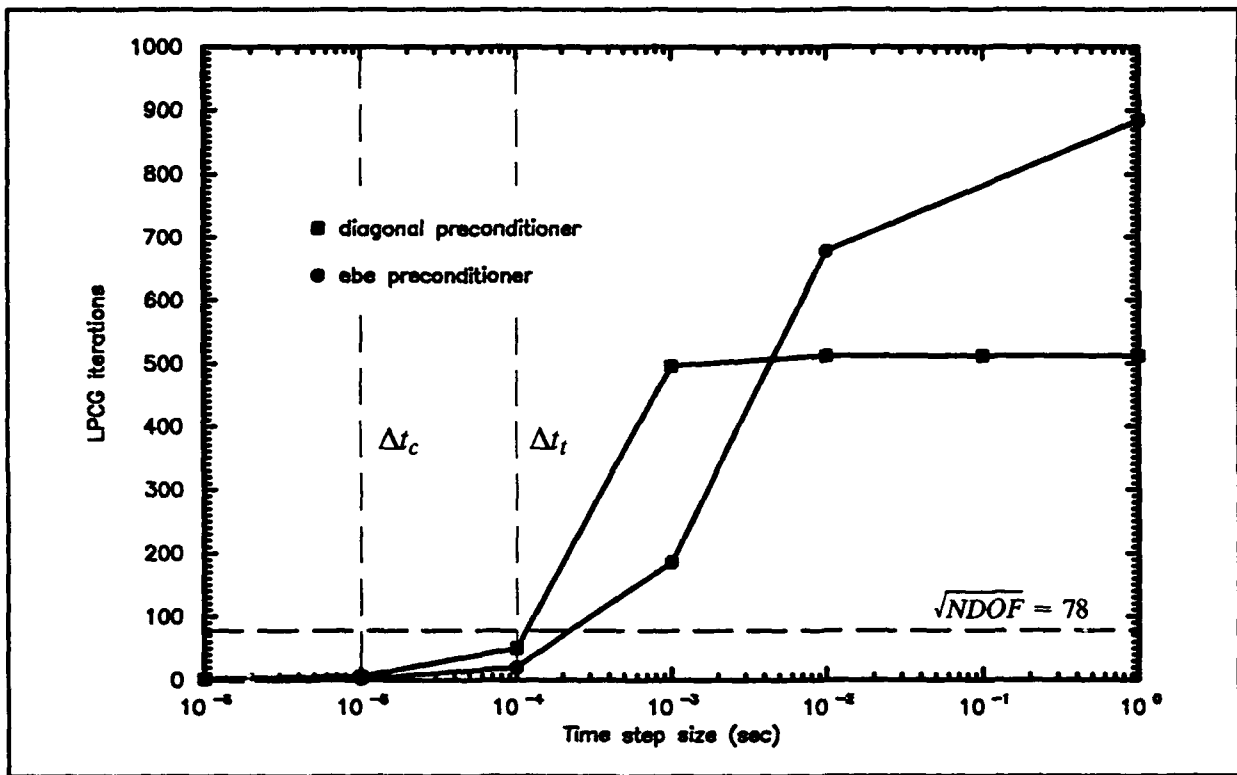


Fig. (4.2-10), LPCG iterations vs. time step size, 1DBE

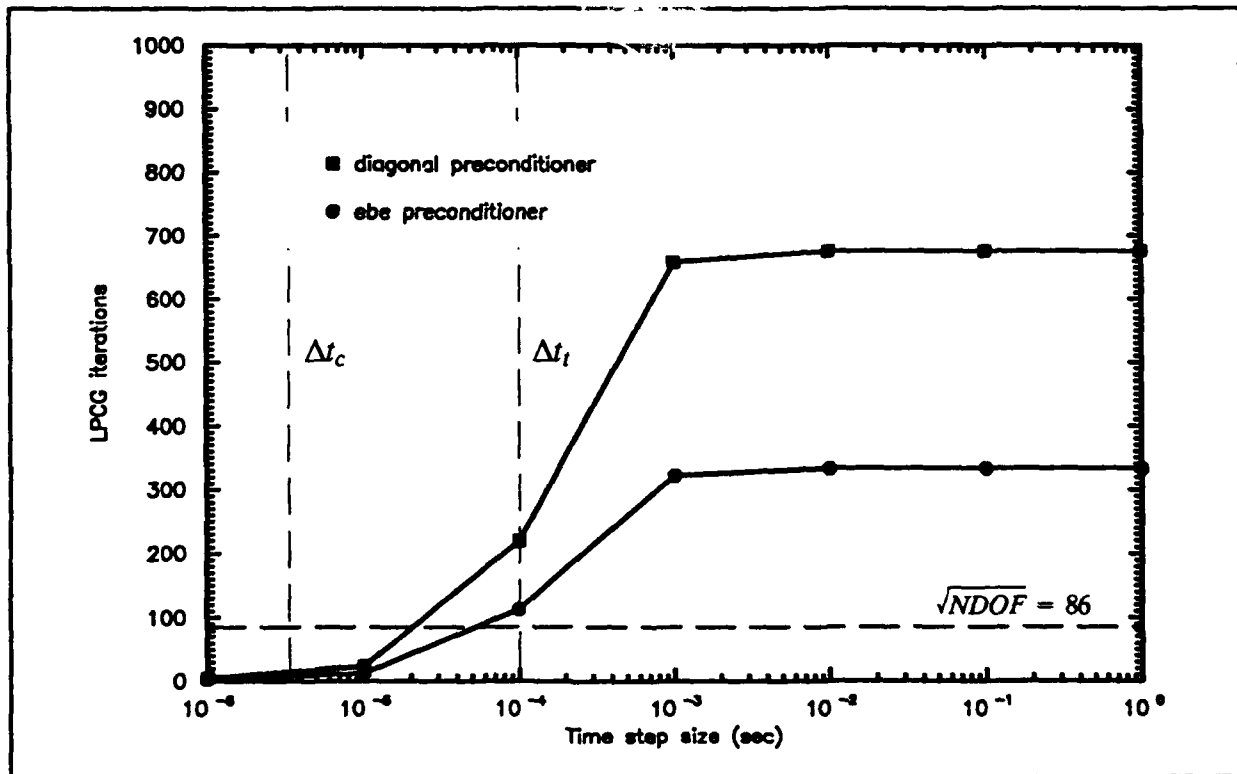


Fig. (4.2-11), LPCG iterations vs. time step size, 2DSB

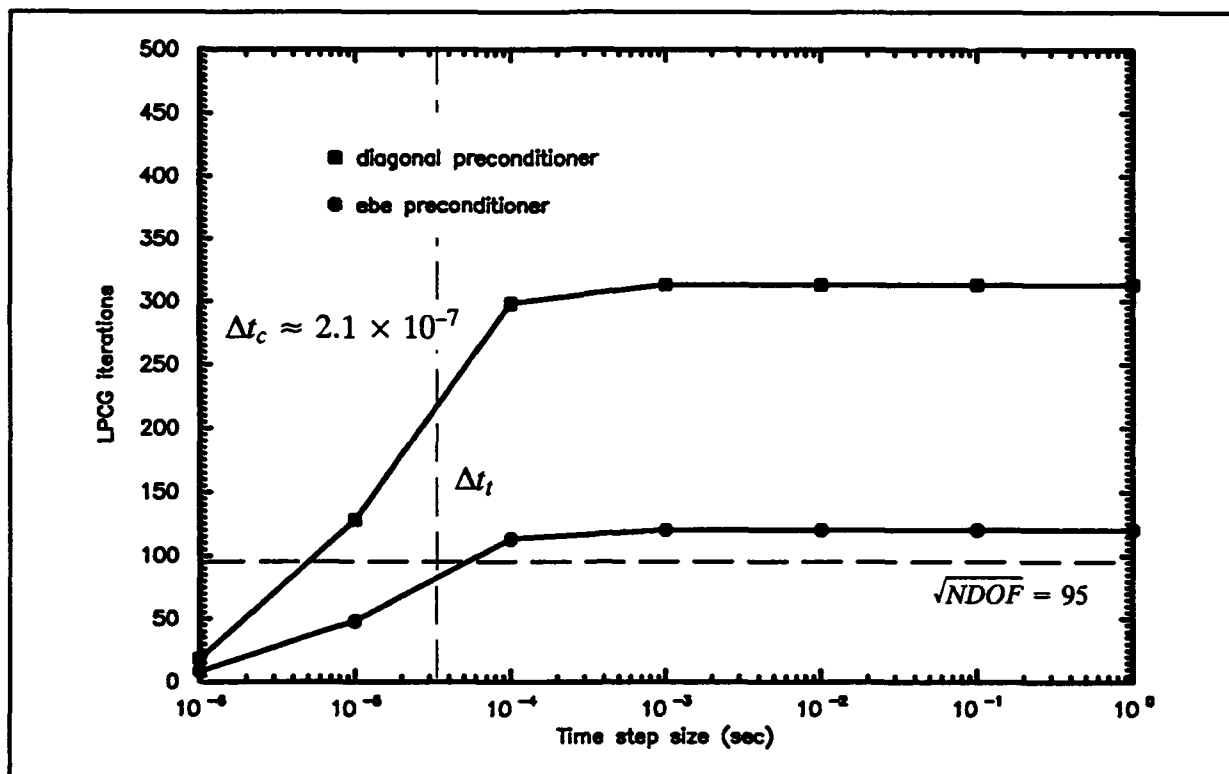


Fig. (4.2-12), LPCG iterations vs. time step size, 3DBB

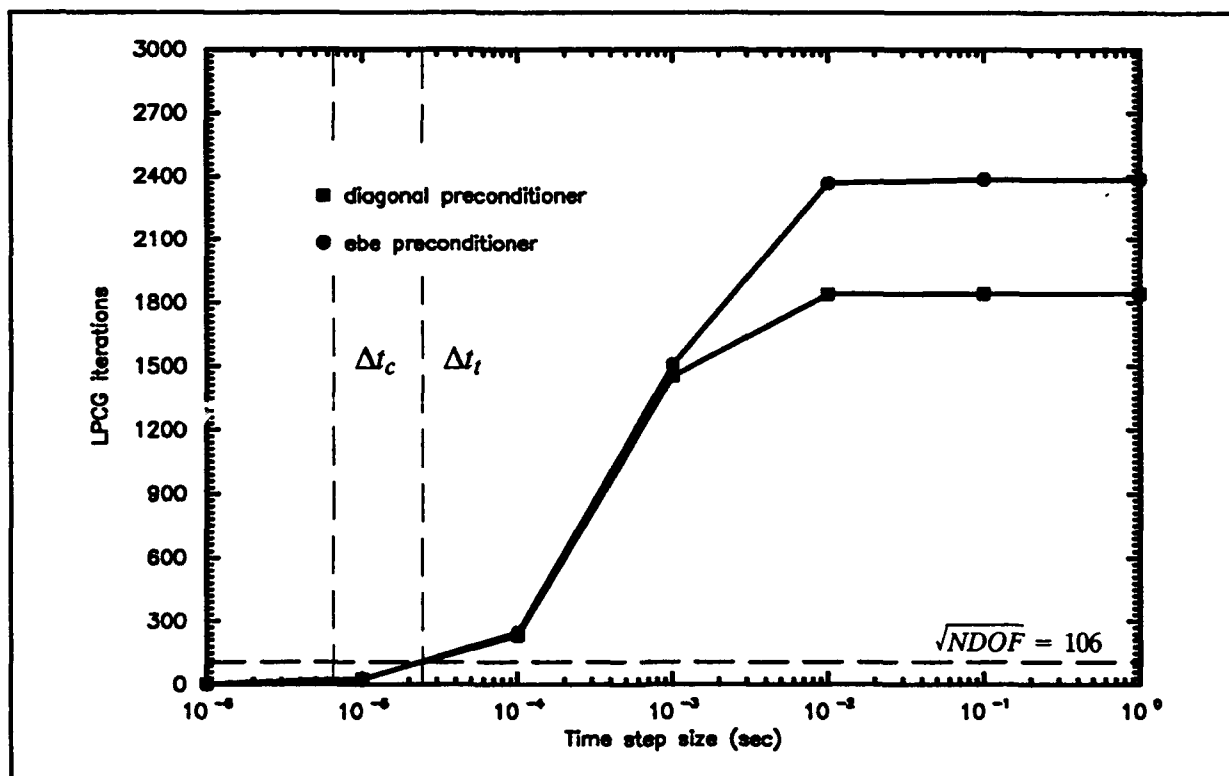


Fig. (4.2-13), LPCG iterations vs. time step size, 3DCS

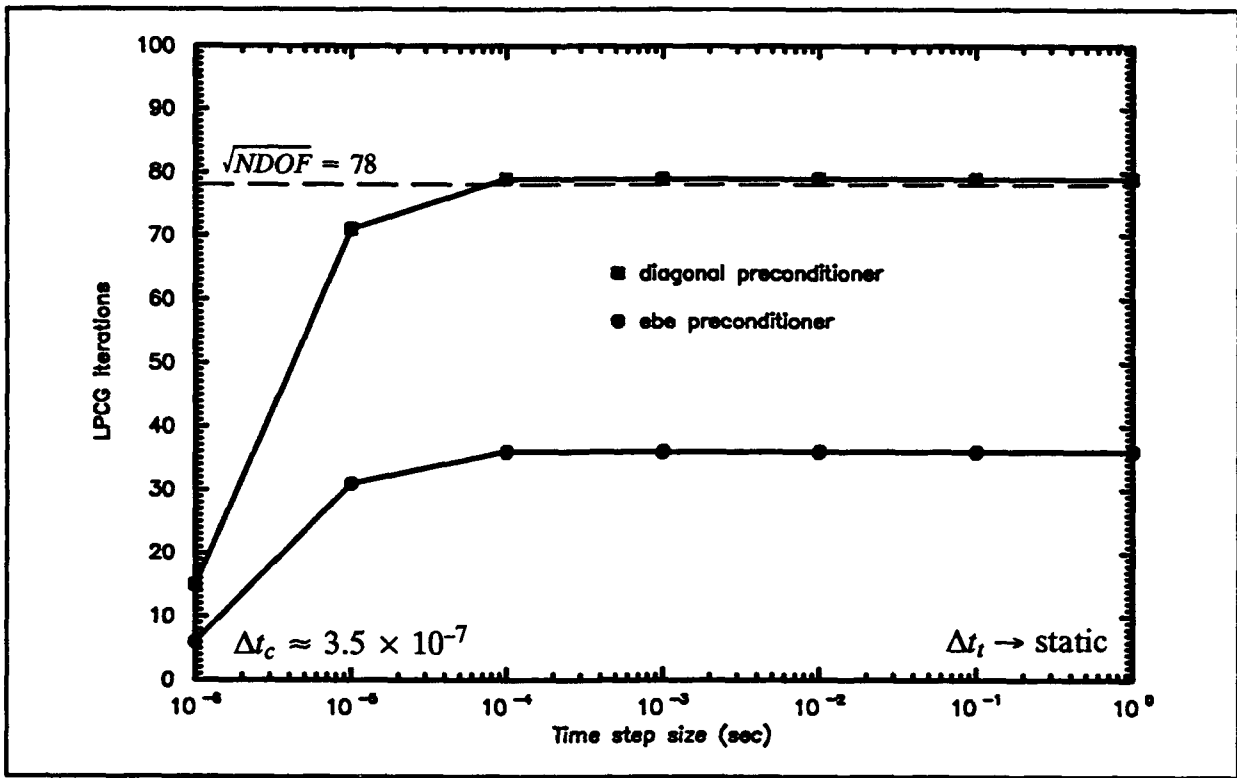


Fig. (4.2-14), LPCG iterations vs. time step size, 3DL12

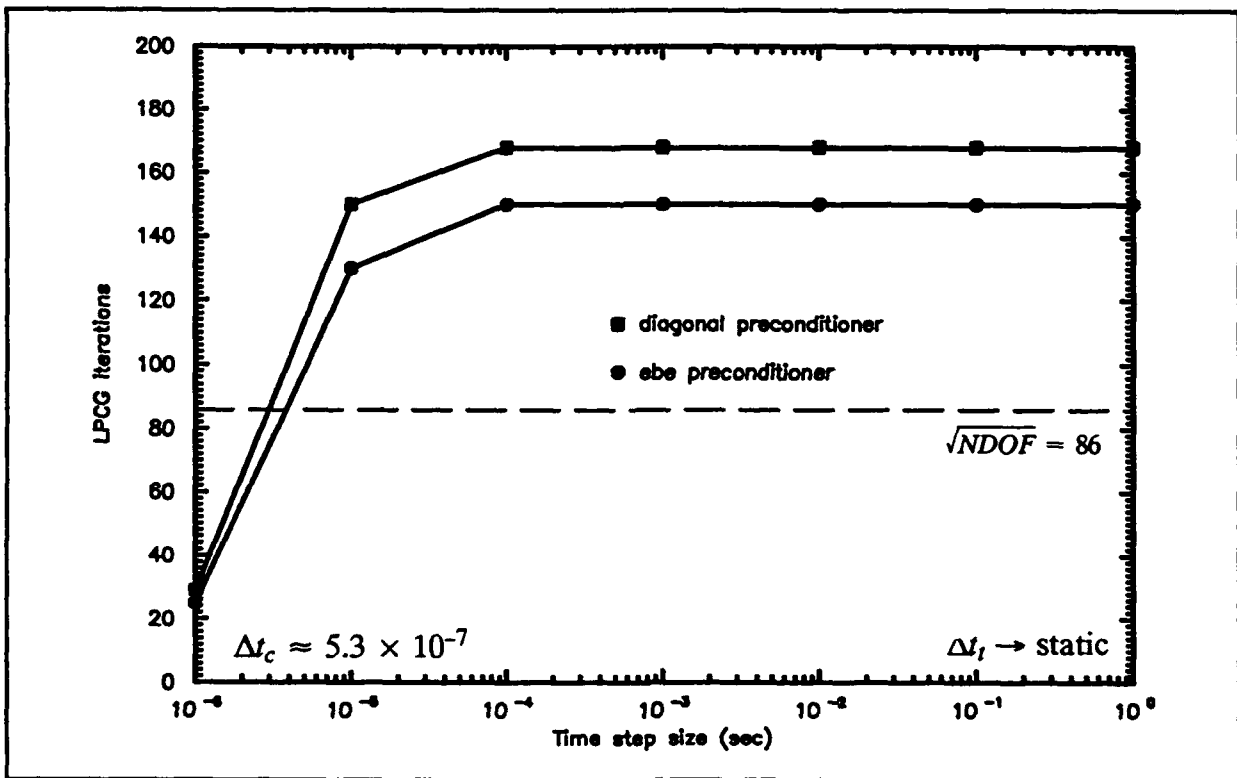


Fig. (4.2-15), LPCG iterations vs. time step size, 3DQ8

ebe in 3DBB could have a definite advantage over explicit dynamics. It seems that the current LPCG preconditioners implemented in NLDfEP can deal successfully with relatively ill-conditioned meshes composed of linear isoparametric elements, but not as successfully with those composed of quadratic isoparametric elements.

Comparing the 3DCS plot to that of 3DBB in Fig. 4.2-9, the condition number of 3DCS is seen to be comparable in magnitude at larger time step sizes and much more sensitive to decreasing the time step size. The 3DCS condition number is nearly constant down only to a time step size of 10^{-2} seconds, and then drops linearly as the time step size continues to decrease. The cause of the ill-conditioning and sensitivity of 3DCS, despite the three dimension aspect of its mesh, is probably the mixture of membrane and bending shell behaviour which leads to wide discrepancies in the eigenvalues of the tangent stiffness matrix.

In Fig. 4.2-13 the convergence rates of both preconditioners are seen to be extremely poor when compared to the $\sqrt{\text{ndof}}$ for time step sizes greater than 10^{-4} seconds. Only for time step sizes less than or equal to 10^{-4} seconds do the convergence rates approach or drop below the $\sqrt{\text{ndof}}$. Further, the diagonal preconditioner consistently converges faster than the ebe preconditioner, leading to its superiority in the 3DCS LPCG timings considered previously. Thus, as in 2DSB, time step sizes in the vicinity of 10^{-4} seconds and less must be required for simulation accuracy if the LPCG w/ diag execution times are to be competitive with those of the direct solver. 3DCS again points out the inadequacy of the current preconditioners for relatively ill-conditioned meshes composed of quadratic elements. Of course, on the Alliant FX/8 the use of the direct solver is not possible, so for 3DCS LPCG is the only alternative. Additionally, for the time step size of 0.25×10^{-4} seconds used in the 3DCS engineering analysis, LPCG w/ diag is nearly 5 times faster than the direct solver in execution time for similar numbers of system factorizations and system back substitutions. Consequently, memory considerations can determine the choice of linear solver even if some efficiency in execution time is sacrificed.

At 10^{-4} seconds, the implicit time step size is about 15 times greater than the critical time step for second central difference. Fig. 4.2-13 indicates that at this time step size the diagonal preconditioner should require approximately 240 linear iterations to converge, about 4 times more than averaged in the 3DCS problem slice. Thus, LPCG w/ diag should remain competitive with the direct solver in execution time, and it may be that implicit dynamics using LPCG would be faster than an explicit solution at this time step. However, the direct solver would still have the advantage over LPCG because much larger time step sizes could be used with no loss of performance.

The 3DL12 condition number is nearly independent of the time step size, as observed in the appropriate plot of Fig. 4.2-9. This plot shows that the condition number remains constant as the time step size decreases from 1.0 to 10^{-4} seconds, and nearly constant to 10^{-5} seconds. This is virtually the entire range of usable time step sizes. Not until the time step is measured in microseconds does the preconditioning effect of the mass matrix assert itself and drive the condition number toward unity. Further, the magnitude of the condition number along the plateau never exceeds 575, meaning that from static problems to impact problems with very small time steps 3DL12 is relatively well-conditioned. This fact enables 3DL12 to be analyzed in the pseudo static manner outlined earlier while maintaining the superior performance of LPCG. Fig. 4.2-9 does not account for influence of nonlinearity on the conditioning of 3DL12, but does suggest that the purely three dimensional nature of the mesh provides a fruitful environment for LPCG, a suggestion supported by the comparison timings for the linear solvers.

The indifference to time step size shown by the 3DL12 condition number is reflected in Fig. 4.2-14. Plateaus exist for both preconditioners that are similar to that seen for the condition number in Fig. 4.2-9. The convergence rates that correspond to these plateaus are 36 linear iterations for the ebe and 79 for the diagonal preconditioner, both of which are less than or equal to the $\sqrt{\text{ndof}}$. Thus, for the linear elastic problem, the performance of both preconditioners is excellent for all time step sizes. While the nonlinearity of the problem degrades this performance, as quantified in Fig. 4.2-17, the linear solver comparison data above shows that even considering the nonlinearity LPCG is faster in execution time than the direct solver for similar numbers of system factorizations and system back substitutions. Further, this performance can only improve as the influence of the mass matrix is included for a truly dynamic analysis. Considering that the critical time step for second central difference is about 3.5×10^{-7} seconds, implicit dynamics using LPCG could have a marked advantage over explicit dynamics. Consequently, for the perfectly three dimensional mesh of 3DL12, LPCG is the dominant linear solver for any time step size.

Like 3DL12, the 3DQ8 condition number is highly insensitive to the time step size. In Fig. 4.2-9, this condition number remains at a virtually constant plateau, equal to approximately 4700.0 at its largest, all the way down to 10^{-5} seconds. At 10^{-6} seconds, the condition number drops, but not as much as seen in previous example problems, reaching only about 570.0. Thus, in contrast with 3DCS, 3DQ8 maintains constant condition numbers over a wider range of time step sizes, its constant value is much smaller, and it is less affected by the preconditioning effect of the mass matrix at very low time step sizes measured in microseconds. This leads to the conclusion that 3DQ8 is far more well-conditioned and stable with

respect to time step size than is 3DCS, and as the latter possesses a definite three dimensional aspect, its sensitivity and ill-conditioning is likely attributable to its mixed shell behaviour, as stated above. Fig. 4.2.-9 shows that the cubic mesh of 3DQ8 provides a favorable environment for LPCG.

That 3DQ8 provides this favorable environment is further supported by Fig. 4.2-15. The plots for both preconditioners show a constant plateau down to 10^{-4} seconds, a gentle downswing to 10^{-5} seconds, and a plunge to 10^{-6} seconds. The convergence rates along the plateau are 168 linear iterations for the diagonal preconditioner and 150 for the ebe preconditioner, both within twice the $\sqrt{\text{ndof}}$. Thus, the convergence rates are in the range of efficient LPCG performance for all time step sizes. The proximity of the curves for the diagonal and ebe preconditioners suggests that LPCG w/ diag should have the advantage, and this is indeed observed in Fig. 4.2-5. Again, the effects of the nonlinearity characteristic of the finite extension problem are not included, but Fig. 4.2-15 provides insight into the behaviour of the problem under varying time step size, and the linear solver results above indicate that considering the nonlinearity in a static analysis, LPCG w/ diag remains faster in execution time than the direct solver for similar numbers of system factorizations and system back substitutions. As with 3DL12, the constant plateau of Fig. 4.2-15 combined with the critical time step size of second central difference, in this case approximately 5.0×10^{-7} seconds, makes it possible that implicit dynamics employing LPCG could require much less computational effort than explicit dynamics. In comparison with 2DSB and 3DCS, it seems that if the conditioning of the problem is not degraded by reduced dimensionality or the problem geometry, the current LPCG preconditioners (primarily the diagonal preconditioner) are able to adequately precondition the problem.

4.2.3.2 EFFECT OF NONLINEARITY AND ELEMENT COMPUTATION ALGORITHM

The performance of LPCG is also dependent on the nonlinearity of the problem and the applied element computation algorithm. Bar charts indicating the average number of LPCG iterations performed per equilibrium iteration in each example problem for both preconditioners and all element computation algorithms are presented in Figs. 4.2-16 and 4.2-17. From these charts, it is observed that the performance of the diagonal preconditioner in all example problems is unaffected by element computation algorithm, but that of the ebe preconditioner often is. This phenomenon and the effect of nonlinearity are addressed in the following discussion.

Only in the case of 3D finite extension does the nonlinear behaviour of the example problems seriously degrade the LPCG convergence rates characteristic of the linear elas-

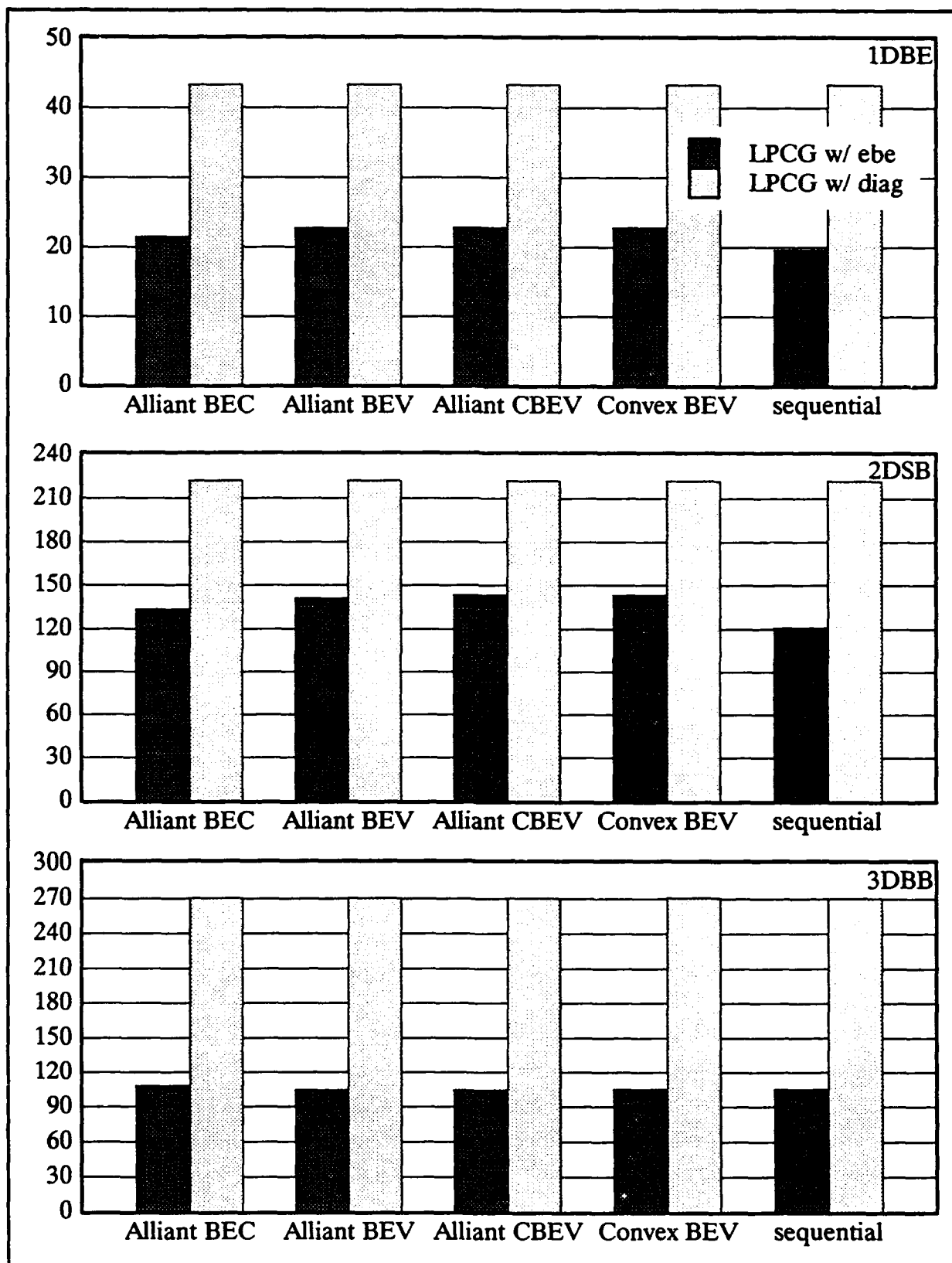


Fig. (4.2-16), average LPCG iterations per equilibrium iteration

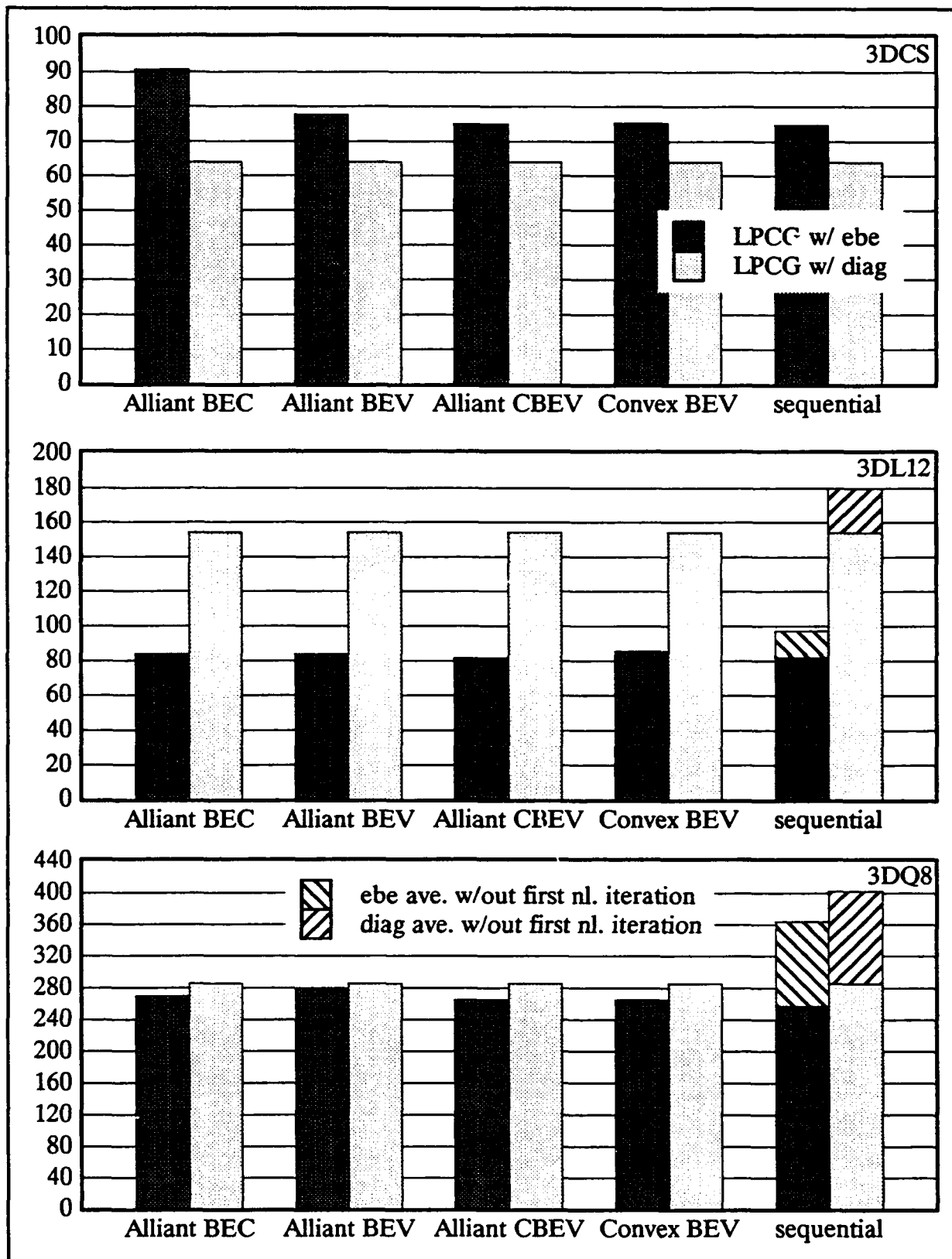


Fig. (4.2-17), average LPCG iterations per equilibrium iteration, cont.

tic dynamic tangent stiffness. For the sequential application of 3DL12, the average convergence rates of the diagonal and ebe preconditioners inferred from Fig. 4.2-17 (with the first linear elastic equilibrium iteration removed) degrade from those of the linear elastic dynamic tangent stiffness observed in Fig. 4.2-14 by factors of approximately 2.2 and 2.6, respectively. Similar behaviour is repeated for 3DL18, which is not pictured in Fig. 4.2-17. Likewise, for 3DQ8 and 3DQ10 the deteriorated convergence rates of both preconditioners are nearly 2.4 times slower. By further progressing the solution of the 3D finite extension problems (as is done in Section 4.4 to compare the nonlinear solution algorithms) the LPCG convergence rates continue to degrade, reaching levels some 4-5 times slower than the linear elastic convergence rates. In the other example problems exhibiting nonlinear behaviour - 1DBE, 2DSB, and 3DBB - the actual LPCG convergence rates are nearly equal to or even faster than those corresponding to the linear elastic dynamic stiffness. The difference seems to be that these problems are dominated by material nonlinearity and show either little or localized geometric nonlinearity, while the 3D finite extension problems combine both in an inherently unstable deformation, as the problems would probably neck given some geometric flaw. In the absence of such instability, the LPCG convergence rates remain comparable to the linear elastic convergence rates.

Even though the deterioration of the LPCG convergence rates is comparable for both the linear and quadratic finite extension problems, the convergence rates of 3DL12 and 3DL18 are much faster than those of 3DQ8 and 3DQ10. Consequently, the deteriorated convergence rates of the linear problems in Fig. 4.2-17 are still comparable to the $\sqrt{\text{ndof}}$ while those of the quadratic problems become quite poor. This is another indication that the current preconditioners perform well for linear elements, but inadequately for quadratic elements.

Figs. 4.2-16 and 4.2-17 indicate that the linear convergence rate associated with the ebe preconditioner is affected by the element computation algorithm, but correlation to the type or extent of the nonlinear behaviour or the type of element employed. For 2DSB, 3DCS, and 3DQ8 the effect of element computation algorithm optimization is relatively pronounced, with the convergence rates for the sequential application being exceeded by as many as 16-22 linear iterations per equilibrium iteration. Such a difference can mean a noticeable decrease in both the speedups and execution time of the LPCG element computations using the element computation algorithms. For 1DBE, 3DBB, and 3DL12 this effect is much reduced, being on the order of 2-3 extra linear iterations. Based on these observations, one would be justified to conclude that the effect is more pronounced for quadratic rather than linear isoparametric elements. However, for the linear mesh of 3DL18 the Convex BEV

degradation from the sequential convergence rate is 16 extra linear iterations (as opposed to just 3 for 3DL12), while for the quadratic mesh of 3DQ10 the Convex BEV convergence rate is actually 3 linear iterations faster (as opposed to 9 slower for 3DQ8). Thus, one might say that quadratic meshes are more likely to be affected, but not in every instance.

There also appears to be no particular element computation algorithm which is responsible for the worst performances of the ebe preconditioner. For instance, it is Alliant CBEV which is worst for 2DSB, Alliant BEC which is worst for 3DCS, Alliant BEV which is worst for 3DQ8, and Convex BEV which performs poorly for 3DL18. It seems to be a more or less random phenomenon as to which element computation algorithm performs the worst for a given problem.

4.2.4 SUMMARY

As the topology of a mesh composed of linear isoparametric elements becomes more three dimensional, such as from the one dimensional mesh of 1DBE to the fully three dimensional cubic meshes of 3DLFE, the conditioning of the dynamic tangent stiffness improves dramatically. In the three dimensional mesh, the linear dynamic tangent stiffness is relatively well-conditioned even at large time step sizes, and as the time step size drops this conditioning is virtually constant until very small time steps sizes are reached. Accordingly, both the ebe and diagonal preconditioners provide for rapid convergence, as compared to the $\sqrt{\text{ndof}}$, for the full range of possible time steps sizes. Therefore, for as large a time step size as can possibly be used for the accuracy of the simulation, LPCG employing either preconditioner will furnish rapid convergence.

Even for a linear mesh with a significant three dimensional topology and a flaw that seriously degrades the overall conditioning of the dynamic tangent stiffness, such as in 3DBB, the situation is much the same. Although the condition number of the 3DBB linear dynamic tangent stiffness is more than 500 times worse than in 3DL12 at large time step sizes and it is dependent on the time step size below 10^{-3} seconds, the preconditioners, primarily ebe, are well able to handle the increased 3DBB ill-conditioning and sensitivity. Even at large time step sizes, an excellent convergence rate, close to the $\sqrt{\text{ndof}}$, is provided. This convergence rate again remains nearly constant with time step, and even the large deformation and extensive yielding about the crack tip exhibited in the engineering analysis does not degrade the convergence.

As the topology of a mesh containing a given number of linear elements becomes more three dimensional, the memory requirements of the direct solver tend to outdistance those of LPCG. The increasing half band width of the dynamic tangent stiffness not only in-

creases memory requirements but represents a greater computational effort in factorization for the direct solver. The direct solver memory requirements as well as the memory discrepancy escalate rapidly with increasing mesh size, so that the memory capacity of a given machine can readily be approached. This leads to considerable paging to and from the swap space during the direct solver factorization, further hampering its performance, while the LPCG system back substitution is relatively free of such costs.

Thus, as the linear mesh topology grows increasingly three dimensional, for similar numbers of system factorizations and system back substitutions LPCG tends to develop an increasing advantage in execution time over the direct solver, even for the extreme case of 3DLFE where the combined material and geometric nonlinearities seriously degrade the convergence rate of LPCG employing either preconditioner in a static analysis. This advantage exists for a wide variety of time step sizes so that a time step can be chosen which allows the implicit dynamics solution to compete very favorably with explicit dynamics. As the mesh size increases, this advantage is accentuated by the relative memory costs of the direct solver and LPCG. The memory discrepancy between the direct solver and LPCG can be so large that the analysis of many problems not possible with the direct solver could be readily accomplished using LPCG.

Both the ebe and diagonal preconditioners show the ability to successfully precondition linear meshes with a significant three dimensional topology. For 3DBB and 3DLFE, which contain such meshes, the solution employing the ebe preconditioner is noticeably less than or approximately equal to that employing the diagonal preconditioner in execution time. In memory usage and in the stability of its solution with regard to the element computation algorithms, the diagonal preconditioner has the advantage.

As in the case of linear elements, for quadratic elements a more three dimensional mesh topology signals an improvement in the conditioning of the dynamic tangent stiffness. From the two dimensional plane strain mesh of 2DSB to the fully three dimensional cubic mesh of 3DQ8, the condition number shows a definite decrease and an increasing insensitivity to time step size. The distribution of the 3DQ8 condition number closely resembles that of 3DL12, although in magnitude it is increased by a factor of 10. However, while the convergence rates attributable to the two preconditioners again follow the same pattern, compared to the $\sqrt{\text{ndof}}$ these preconditioners are not as effective as in the linear problem. Still, the time step size can again be raised as high as desired while maintaining the same rate of convergence.

If the quadratic mesh has a significant three dimensional topology and an attribute which induces a deterioration of the conditioning, such as the mixed bending and membrane behaviour of 3DCS, then the situation worsens. Although the large time step size condition numbers of 3DCS are close to those of 3DBB, the convergence rates of both preconditioners are much worse and highly dependent on time step size. This indicates the inability of the preconditioners to deal with the increased ill-conditioning, and limits the range of time step sizes for which LPCG employing either preconditioner exhibits good performance.

As with linear elements, with an increasing dimensionality of the mesh topology comes increasing half band widths and memory requirements and greater direct solver factorization costs. The magnitude of the half band width for three dimensional quadratic meshes is greater than for linear meshes, so that the discrepancy between the direct solver and LPCG memory requirements and the direct solver factorization costs are also greater than for similarly sized linear meshes. Also, the memory capacity of a given machine can be approached even more quickly with increasing mesh sizes, leading to the same consequences discussed for linear meshes.

Thus, as a mesh topology composed of quadratic elements becomes more three dimensional, without an attribute which causes increased ill-conditioning the conclusion is much the same as it was for linear elements. Even though the performance of the preconditioners is worse for quadratic meshes, it continues to be independent of the time step size, and the memory and factorization costs of the direct solver are heavier. Thus, the memory and execution time advantages of LPCG and the suitability of implicit dynamics remain, even in the similarly extreme case of 3DQFE.

However, for three dimensional ill-conditioned meshes the execution time advantage of LPCG is tied to the time step size. In this case, it may be possible to specify a time step size, within the range of time step sizes desired for reasons of simulation accuracy, small enough to maintain the execution time advantage of LPCG and large enough to permit implicit dynamics to compare favorably with explicit.

The example problems show that for quadratic elements neither the diagonal nor the ebe preconditioner performs particularly well, and often very poorly unless the time step size is reduced to gain a significant preconditioning effect from the mass matrix. In general, the performance of the diagonal preconditioner is predominant due to a similar or superior convergence rate leading to less computational cost and its indifference to varying the element computation algorithm.

4.3 ELEMENT COMPUTATION ALGORITHMS

In this section, the performance of the element computation algorithms in each of the example problems is assessed. The blocking and memory requirements of the algorithms and the speedups of the algorithms versus both the sequential application and compiler optimization are presented and discussed. A summary is provided at the end of the section.

4.3.1 ELEMENT BLOCKING AND MEMORY REQUIREMENTS

One important facet of the performance of each element computation algorithm is the extent to which it can be efficiently blocked and the amount of memory required to execute the necessary element computations. In the following each of these considerations are explored.

4.3.1.1 ELEMENT BLOCKING

The details of the element blocking for each of the element computation algorithms and example problems is shown in Fig. 4.3-1. Optimal blocking for all of the element computation algorithms occurs in the sense that all of the vector processors and vector registers are used to the fullest extent for which the algorithm was designed. For instance, the number of available processors and the length of the vector registers on the Alliant FX/8 are 8 and 32, respectively. Thus optimal blocking for Alliant BEC would consist of 8 similar, nonconflicting elements in every block. Likewise, optimal blocking for Alliant BEV would involve 32 elements in each block and Alliant CBEV blocks of 256 elements in sub-blocks of 32 elements executed concurrently 8 at a time. As the vector length of the Convex C240 is 128, optimal blocking for Convex BEV would amount to 128 elements in each block.

The blocking for BEC in Fig. 4.3-1 is observed to be optimal or nearly optimal for all of the example problems considered. This results from the relatively small block length of eight required by BEC. Further, outside of the loss of some concurrency, the penalty for incomplete blocking – such as the blocks of 2 elements seen in 3DBB, 3DL12, and 3DCS – is not overly severe as the calculations for a given element are quite similar to the sequential calculations and not a great deal of extra overhead is experienced. Thus, with respect to the element blocking, BEC generally exhibits excellent behaviour for meshes containing relatively many or few elements..

The blocking for Alliant BEV is less optimal than that of BEC but remains satisfactory. The larger block length of 32 leads to greater numbers of incomplete blocks, especially for 3DBB and 3DL12 which contain relatively large numbers of elements, but the number of

prblm eca	1DBE	3DBB	3DL12	2DSB	3DCS	3DQ8
Alliant BEC	64x8	304x8 + 6 + 2	214x8 + 2x7 + 2x2	62x8 + 2x6	94x8 + 2x6 + 2x2	64x8
Alliant BEV	16x32	74x32 + 24 + 20 + 15 + 11 + 1 + 1	50x32 + 2x28 + 2x18 + 2x18 + 2x14 + 2x4	12x32 + 4x31	20x32 + 4x30 + 4x2	16x32
Alliant CBEV	2x[8x32]	4x[8x32] + [6x32 + 2x31] + [4x32 + 4x31] + 2x[2x30 + 6x29] + 2x[7x15 + 14] + 2x[4x13 + 4x12] + [2x1] + [2x1]	8x[8x27]	4x[7x16 + 15]	8x[8x12]	8x[8x8]
Convex BEV	4x128	12x128 + 119 + 118 + 117 + 115 + 114 + 106 + 103 + 102 + 10	3DL12: 8x128 + 4x89 + 4x87 3DL18: 40x128 + 8x89	4x127	8x96	3DQ8: 8x64 3DQ10: 8x125

Fig. (4.3-1), element blocking for all example problems

optimal or nearly optimal blocks is still dominant. This dominance should increase as the number of elements increases, as it does from 3DL12 to 3DBB. The penalty for incomplete blocking is more significant for BEV as for small blocks both concurrency and vectorization is squandered and, as the calculations are rearranged so that the element block customarily comprises the inner loop of (artificially) nested DO loop structures, loop overhead is much increased. Consequently, Alliant BEV is generally well-blocked for a wide variety of mesh sizes, although some effects of incomplete blocking for smaller meshes could be seen.

CBEV is the most poorly blocked element computation algorithm in Fig. 4.3-1. The blocking for the three meshes employing quadratic elements - 2DSB, 3DCS, and 3DQ8 - is particularly poor, as only 25-50% of the Alliant vector registers are utilized for all blocks. This results from the much larger CBEV block length of 256, and the fact that these blocks must be further sub-divided into blocks suitable for concurrent execution. Coupling this larger block length with the fewer number of elements characteristic of quadratic meshes leads to reduced blocking, especially as the dimensionality of the mesh topology increases. For the linear meshes, the situation is not quite as bad, but there are still significant numbers of incomplete blocks in 3DBB and the vector registers are not fully utilized in 3DL12. As with all element computation algorithms employing designed block vectorization, incomplete blocking carries a relatively stiff penalty as it leads not only to wasted opportunities for vectorization but to increased loop overhead as well. Fortunately, the blocking algorithm is designed to favor concurrency so that at least the processors are in general not sitting idle. For meshes with a significant three dimensional topology, CBEV can only be well-blocked for very large numbers of elements, so that for a great number of applications the CBEV blocking is adequate at best and debilitating at worst.

In spite of a block length of 128, half that of CBEV, Fig. 4.3-1 shows Convex BEV to be typically well-blocked. This is the consequence of not having to sub-divide the blocks for concurrent processing. The cases of worst blocking occur for 3DCS, where the vector registers are 75% utilized, and 3DQ8, where they are only 50% utilized. Still, vector lengths of 96 and 64 are not small. Further, as the size of the mesh increases, as from 3DQ8 to 3DQ10, the blocking correspondingly improves. The progression of block length from 8 (BEC) to 32 (Alliant BEV) to 128 in Fig. 4.3-1 shows that as long as there is no need to further sub-divide the blocks, good blocking can be achieved for most applications for a variety of block lengths. However, if the blocks are sub-divided, it is difficult to maintain an adequate vector length if designed block vectorization is a concern.

4.3.1.2 ELEMENT MEMORY REQUIREMENTS

The sizes of the element common areas which contain the element block data structures are shown in Fig. 4.3-2 for the element computation algorithms studied. Of course, the entire common area is not in use at any given time, but Fig. 4.3-2 provides an estimate of the amount of memory commanded by the various element computation algorithms. The size of the element common area for a given example problem is dependent on the nature of the element blocking.

From Fig. 4.3-2, BEC is seen to require the least amount of memory, while on the Alliant FX/8 CBEV requires by far the most. This is because BEC block data structures depend on the number of processors available, while CBEV structures hinge on both the number of available processors and the vector length of the processors. As the vector length easily exceeds the number of processors for the machines considered in this research, the element computation algorithms relying on designed vectorization suffer with respect to memory. BEC block data structures have the advantage that they are small enough to utilize the cache memory more effectively and this advantage is apparent in the results that follow. For the bar extension problem, CBEV data structures rival the size of system solution memory required and the manipulation of this data can be cumbersome and cause a decrease in efficiency due to heavy bus traffic, gather/scatter overhead, and frequent cache misses. This problem is not as severe for BEV on the Alliant FX/8 because its element block data structures rely solely on the size of the vector length. However, Alliant BEV still requires over four times the element common memory of BEC.

Problems caused by the large size of the CBEV element common area are manifest in 3DBB. This size, combined with the large number of elements, leads to difficulty on the Alliant FX/8. The solution of CBEV version LPCG w/ ebe runs can become problematical in the sense that the number of nonlinear equilibrium iterations required for convergence of a given time step will vary randomly from run to run. Sometimes, the number of equilibrium iterations will conform to the results of all the other element computation algorithms, which is assumed to be the correct solution. Other times, arbitrarily, the resolution of a time step, any time step, will require many more equilibrium iterations than usual. One can only conclude that data is being lost somewhere along the bus, causing the solution to be thrown off track. This phenomenon occurs only for 3DBB, and only for CBEV. The inference is that the large, sustained, amount of bus traffic demanded by the size of the CBEV element data structures combined with the large number of elements at times overloads the bus. This is a significant strike against the feasibility of CBEV.

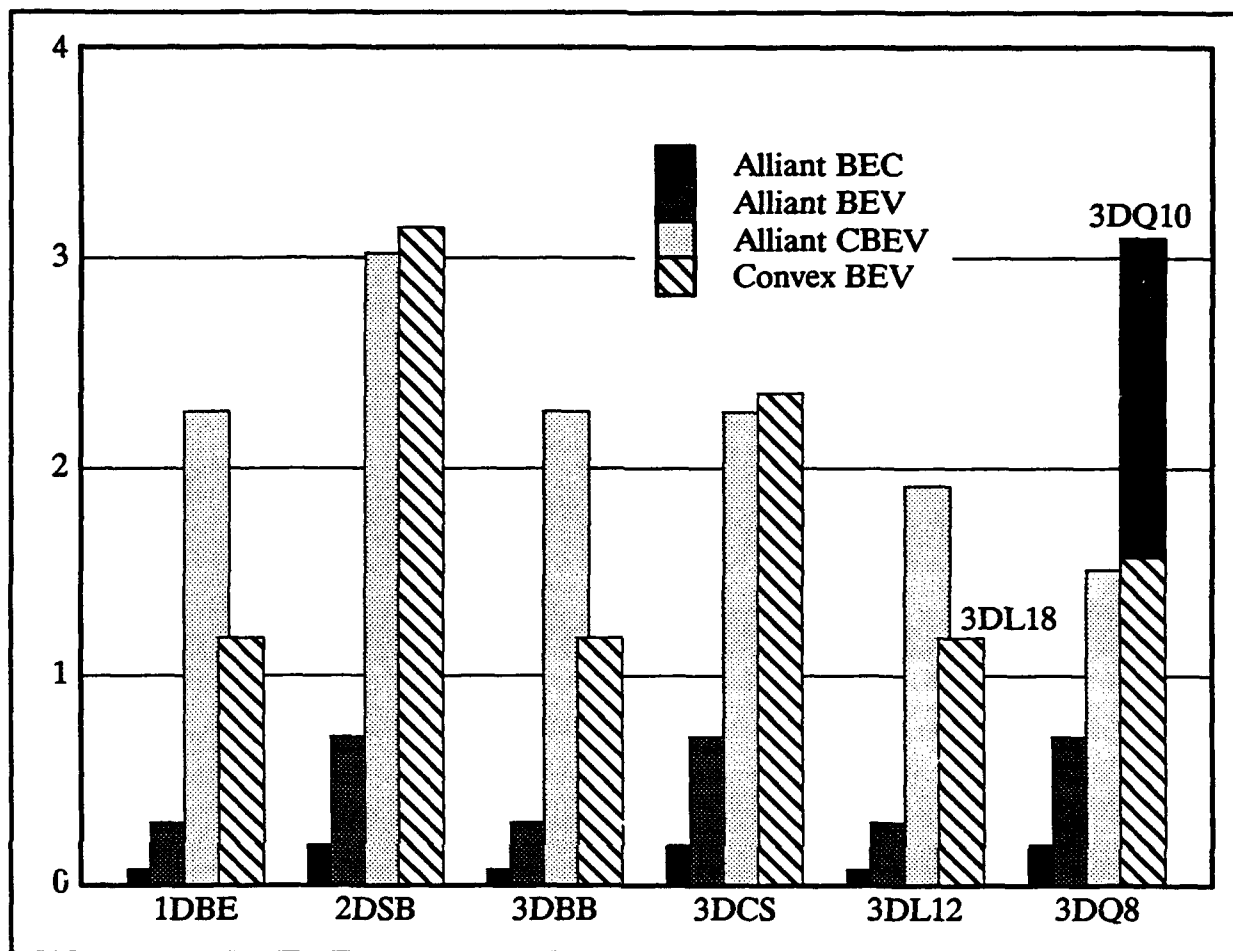


Fig. (4.3-2), size of element common area for example problems, in megabytes

The element common memory requirements for Convex BEV rival those of CBEV because of the large Convex vector length, and like CBEV Convex BEV may be handicapped by the overhead caused by its stringent memory requirements. In fact, for the example problems employing quadratic elements, Convex BEV exceeds CBEV in element common memory. This is because Fig. 4.3-1 illustrates that Convex BEV is more fully blocked than CBEV for these problems and thus its element common memory requirements are greater. For all of the example problems, BEC and Alliant BEV contain at least one full block, so that moving in Fig. 4.3-2 across the example problems containing linear elements and across those containing quadratic elements, the element common memory requirements of BEC and Alliant BEV are constant. Notice that, for equal blocking, the requirements of quadratic element meshes exceed those of linear elements due to the greater number of degrees of freedom.

The reduced Convex BEV blocking in 3DCS and/or 3DQ8 can be a benefit if the vector lengths provided by the element blocking are large enough so that the decrease in the required element common memory, and thus the block granularity of the element computations, alleviates overhead costs while the vector calculations are still efficient. This trade-off is examined in the following discussion.

4.3.2 SPEEDUPS

The comparison of the element computation algorithms is divided into two areas: basic element computations and LPCG element computations. The basic element computations include the calculation of the internal force vector, tangent stiffness matrix, and lumped tangent mass matrix calculations, and the stress recovery. LPCG element computations for the ebe preconditioner consist of the ebe factorization process (Crout factorization and Winget regularization), the solution of the ebe preconditioning step (element forward reduction and back substitution), and the step length calculation. For the diagonal preconditioner, only the step length calculation is performed using the element computation algorithms.

The speedups concerning the basic element computations are presented in Figs. 4.3-3 to 4.3-10, and those concerning the LPCG element computations are given in Figs. 4.3-11 to 4.3-16. Both speedups against the sequential application and the compiler optimized sequential application are shown. For the latter, the sequential code was manipulated so that the compiler could optimize it as fully as possible, within the limits of the sequential structure. A comparison against compiler optimized sequential code is intended to highlight the need for optimized element computation algorithms. The speedups are obtained from direct solver, LPCG w/ ebe, and LPCG w/ diag runs. They are averaged across the runs for a given linear solver and, for computations that are identical for all linear solvers, from these the greatest speedup is chosen. Differences in computations for different linear solvers are noted in context.

4.3.2.1 BASIC ELEMENT COMPUTATIONS

Figs. 4.3-3 and 4.3-4 contain the speedups for the internal force vector calculation. On the Alliant FX/8, BEC generally exhibits the greatest sequential speedups for linear elements, in the range of 11-12, although BEV is close. This occurs because BEC operates with the relatively inefficient incidental vector length of 24 associated with eight node isoparametric elements, while BEV uses this value as the size of the outer loop in COVI mode execution. As 24 is an even multiple of 8, the number of vector processors, COVI mode execution is fairly efficient. This smaller incidental vector length is also reflected by the excellent

speedups versus compiler optimization, in the range of 4–5 for BEC and BEV, as compiler optimization relies primarily on vector-concurrent loops of length 24.

The CBEV internal force vector speedups for linear elements are much smaller than seen in BEC or BEV. Apparently, there is simply too much data to be manipulated, which leads to prohibitive gather/scatter costs and probably more cache misses. The slight effect of incomplete blocking can probably be seen in the decrease in CBEV speedups from 1DBE to 3DBB and 3DL12.

For quadratic elements, BEC by far displays the greatest internal force vector sequential speedups, on the order of 13. The BEC sequential speedups increase from those associated with linear elements because of the increased incidental vector length, which rises from 24 for linear elements to 60 for quadratic elements. The increased incidental vector length also leads to decreased speedups versus compiler optimization, on the order of 3, as the compiler is more effective working with the larger vector length. BEV is worse for quadratic elements because the outer concurrent loop in COVI mode execution also rises to 60, which is not an even multiple of 8 and thus decreases the COVI mode efficiency.

The effect of incomplete blocking on the CBEV internal force vector speedups for quadratic elements is clearly seen in Fig. 4.3–4. The reduction in speed of the CBEV calculations is so pronounced that CBEV is not much better than compiler optimization.

The internal force vector sequential speedups are approximately the same for linear and quadratic elements, about 5, using Convex BEV. Here, the increased size of the outer loop resulting from the use of quadratic elements that hampers Alliant BEV has little or no effect because concurrency is not employed. The increased incidental vector length of the sequential application associated with quadratic elements is reflected in the compiler optimized speedups, which decrease from over 3 for linear elements to about 2.5 for quadratic elements. The effect of incomplete blocking for 3DCS and 3DQ8 appears to be negligible for the internal force calculation, with the trade-off between reduced vector efficiency and reduced block granularity canceling out.

Figs. 4.3–5 and 4.3–6 display the speedups for the tangent stiffness matrix calculation. Again, for linear elements, on the Alliant FX/8 the BEC and BEV speedups are quite comparable, and for basically the same reasons enumerated above for the internal force vector calculation. The speedups against compiler optimization for linear elements, which range from 2.5–3.0, are reduced from internal force vector levels by the ability of the compiler to perform the sequential triple matrix product B^TDB necessary to compute the tangent stiffness matrix effectively. Correspondingly, the BEC sequential speedups for linear elements are

also generally increased from internal force vector levels, reaching a high of about 13. CBEV sequential speedups are again lesser than those of BEC and BEV, probably due to its greater block granularity and memory requirements.

An interesting development can be observed in Fig. 4.3-5. For 1DBE, which contains 512 elements, the sequential speedups for BEC, Alliant BEV, and CBEV are approximately 13, 13, and 11, respectively. For 3DBB, which contains 2440 elements, those values drop to about 10, 10, and 9. Finally, for 3DL12, which contains 1728 elements, the BEC sequential speedup returns to about 13 while those of Alliant BEV and CBEV remain depressed. Further, the sequential speedups of Convex BEV remain stable at approximately 4. Thus, the key factors are seen to be the number of elements, the size of the element block, and the memory capacity of the machine. Alliant BEV and CBEV also operate with a greater degree of incomplete blocking for 3DBB and 3DL12, but this effect is probably minimal. The cause of the depressed speedups is probably increased overhead and system paging costs during the gather/scatter operations on the Alliant FX/8, which are probably aggravated by larger numbers of elements combined with the block granularities and memory requirements of the various element computation algorithms for the tangent stiffness matrix calculation.

For quadratic elements, BEC again owns the largest tangent stiffness matrix sequential speedups, which consistently exceed 15. BEV sequential speedups drop, and the compiler optimized speedups also drop to a range of about 1.5-2.0. All of this is expected due to the increased incidental vector length for BEC and the compiler optimized sequential application and the decreased outer loop efficiency of Alliant BEV COVI execution. Again, the incomplete blocking of CBEV for the example problems employing quadratic elements manifests itself in very poor performance, to the point that compiler optimization is actually faster.

The Convex BEV tangent stiffness matrix sequential speedups are very similar for both linear and quadratic elements, although they are slightly reduced for quadratic elements. This is probably due to the increased block granularity of the quadratic calculations and possibly increased outer loop overhead in the triple matrix product. The compiler optimized speedups drop as expected, again going from 2.5-3.0 for linear elements to 1.5-2.0 for quadratic elements. The incomplete blocking for 3DCS and 3DQ8 seems to have an insignificant effect on the tangent stiffness calculation, with the trade-off again canceling out.

A general trend observed in Figs. 4.3-7 and 4.3-8 is that the lumped tangent mass matrix sequential speedups for the direct solver runs are consistently greater than those of the LPCG solver runs. In these figures, the sequential and compiler optimized speedups refer to

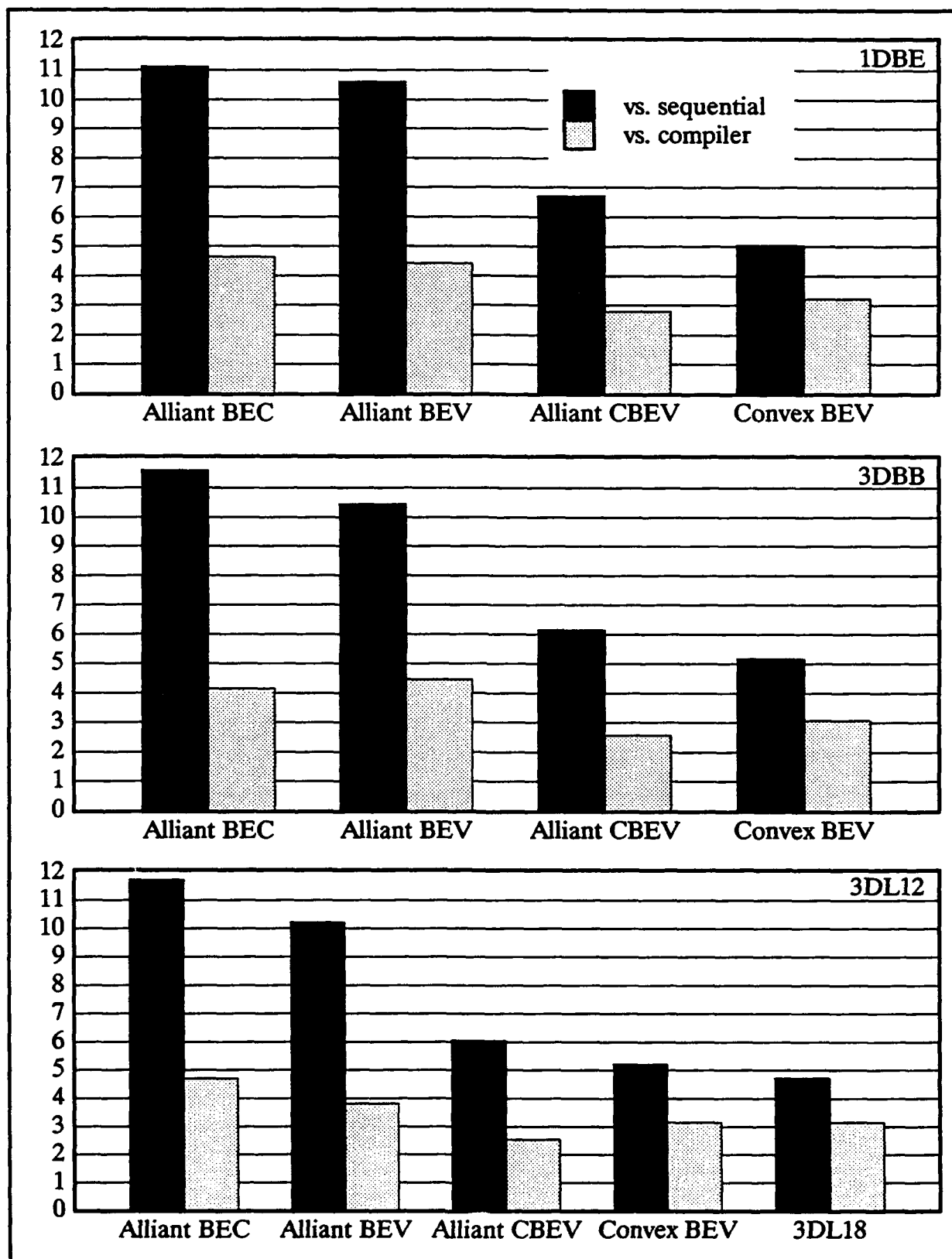


Fig. (4.3-3), speedups for internal force vector, problems with linear elements

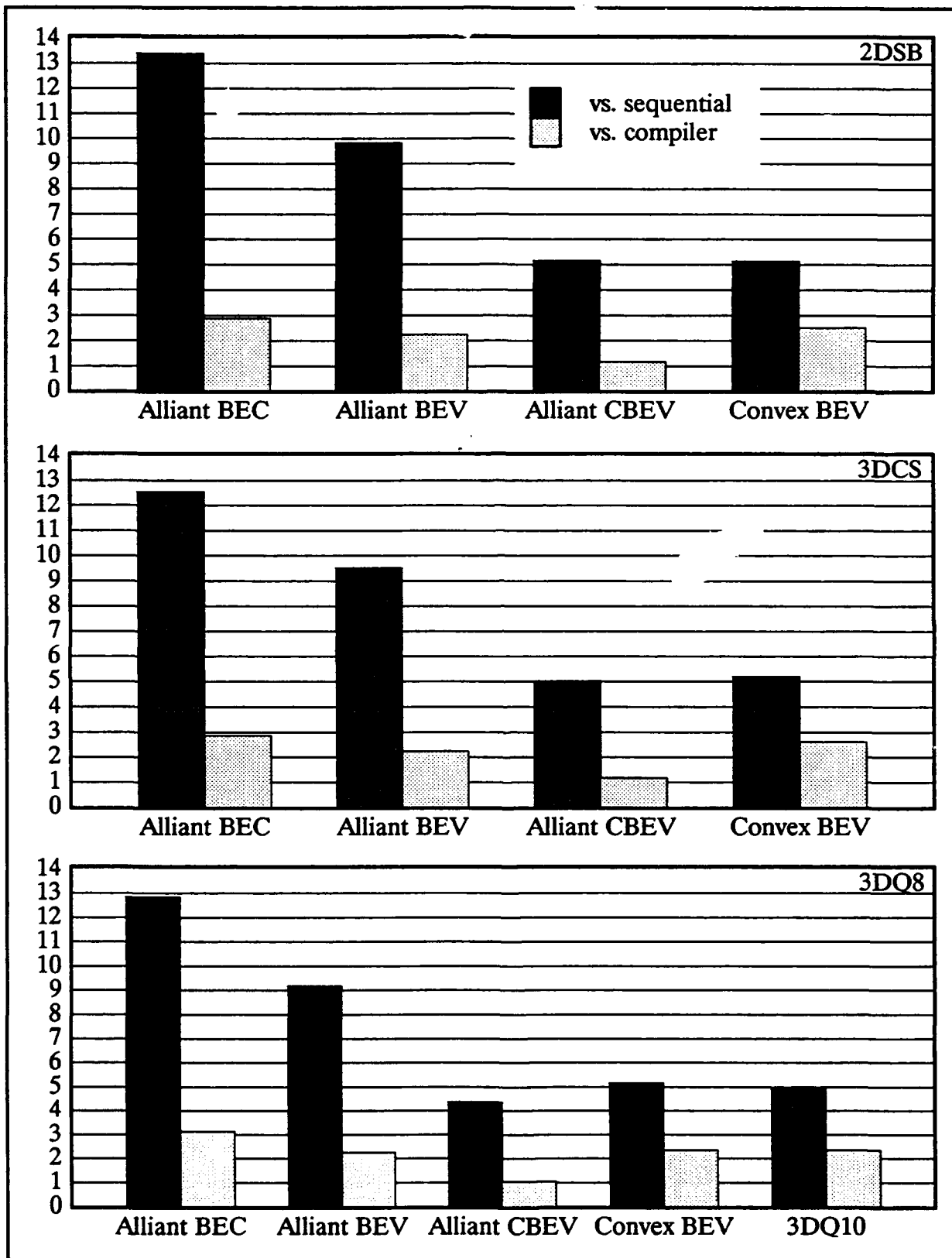


Fig. (4.3-4), speedups for internal force vector, problems with quadratic elements

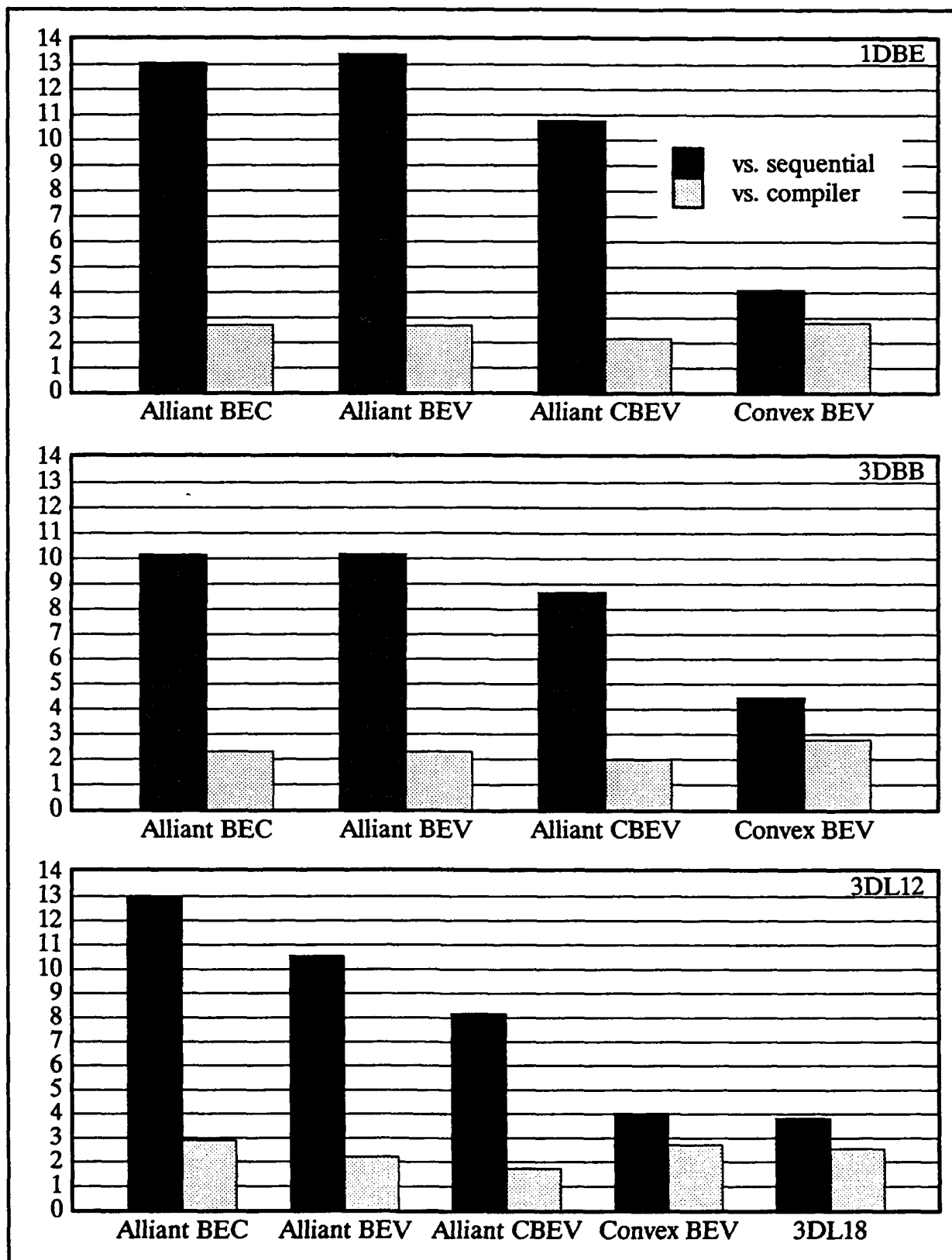


Fig. (4.3-5), speedups for tangent stiffness matrix, problems with linear elements

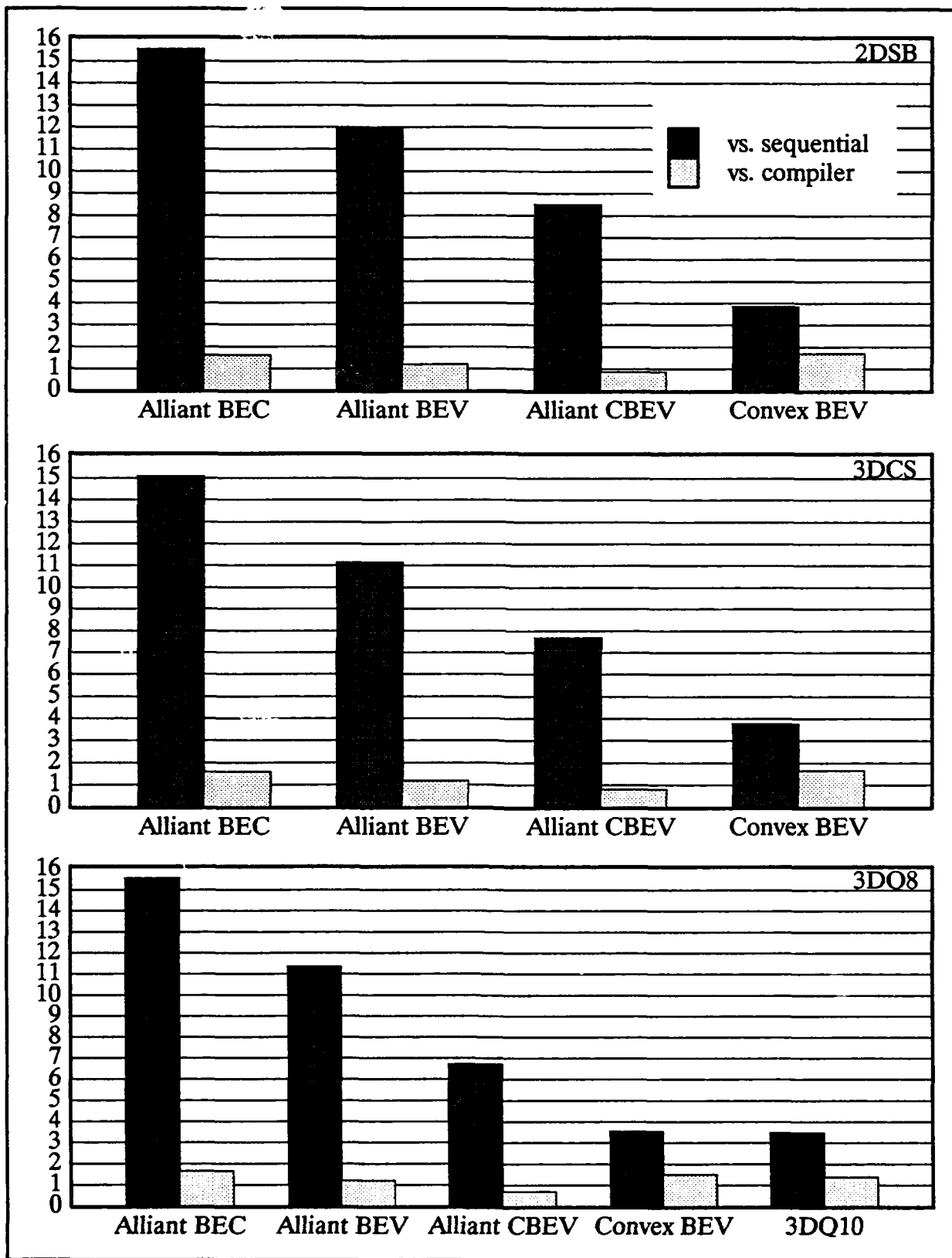


Fig. (4.3-6), speedups for tangent stiffness matrix, problems with quadratic elements

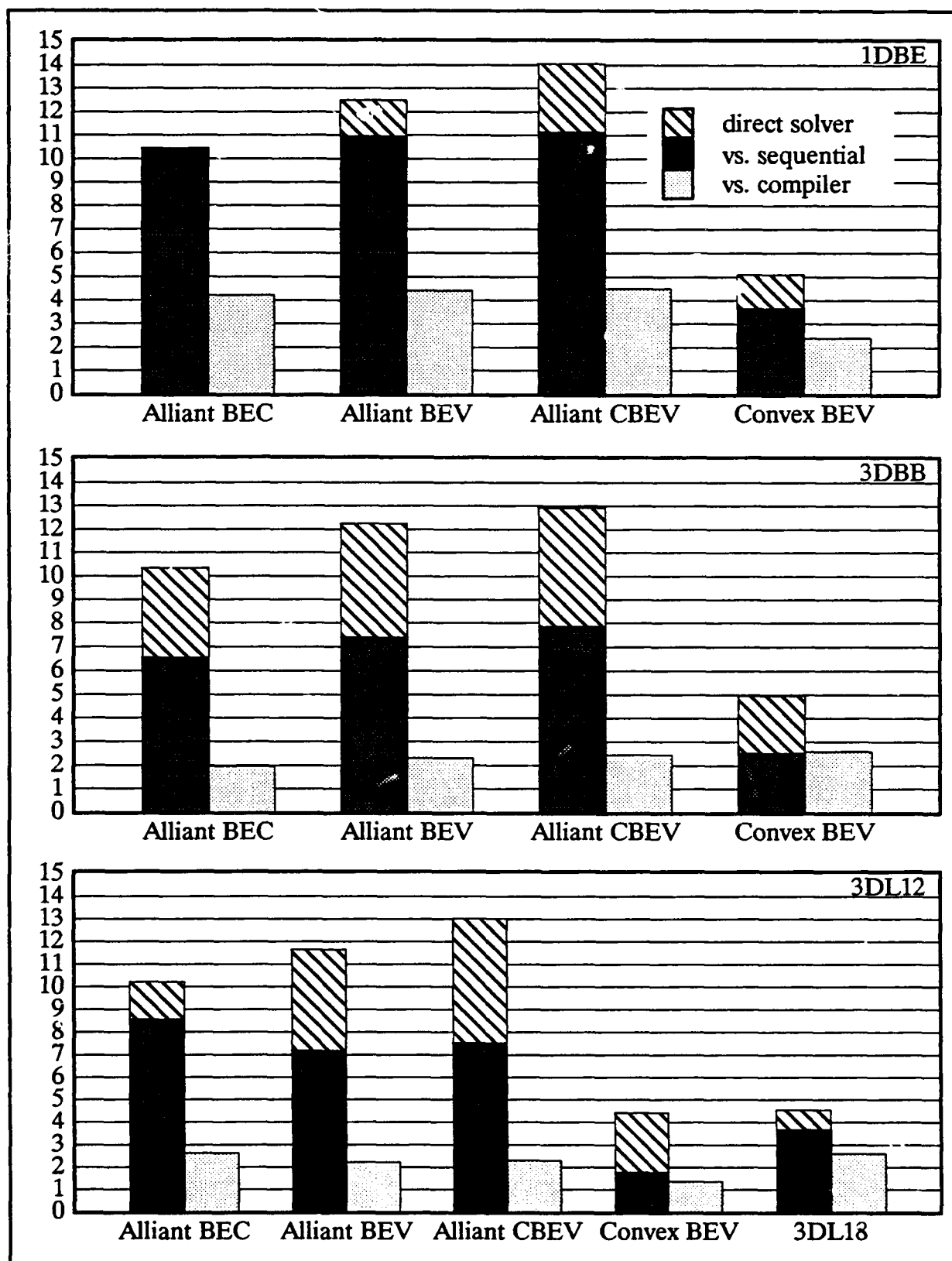


Fig. (4.3-7), speedups for lumped tangent mass matrix, problems with linear elements

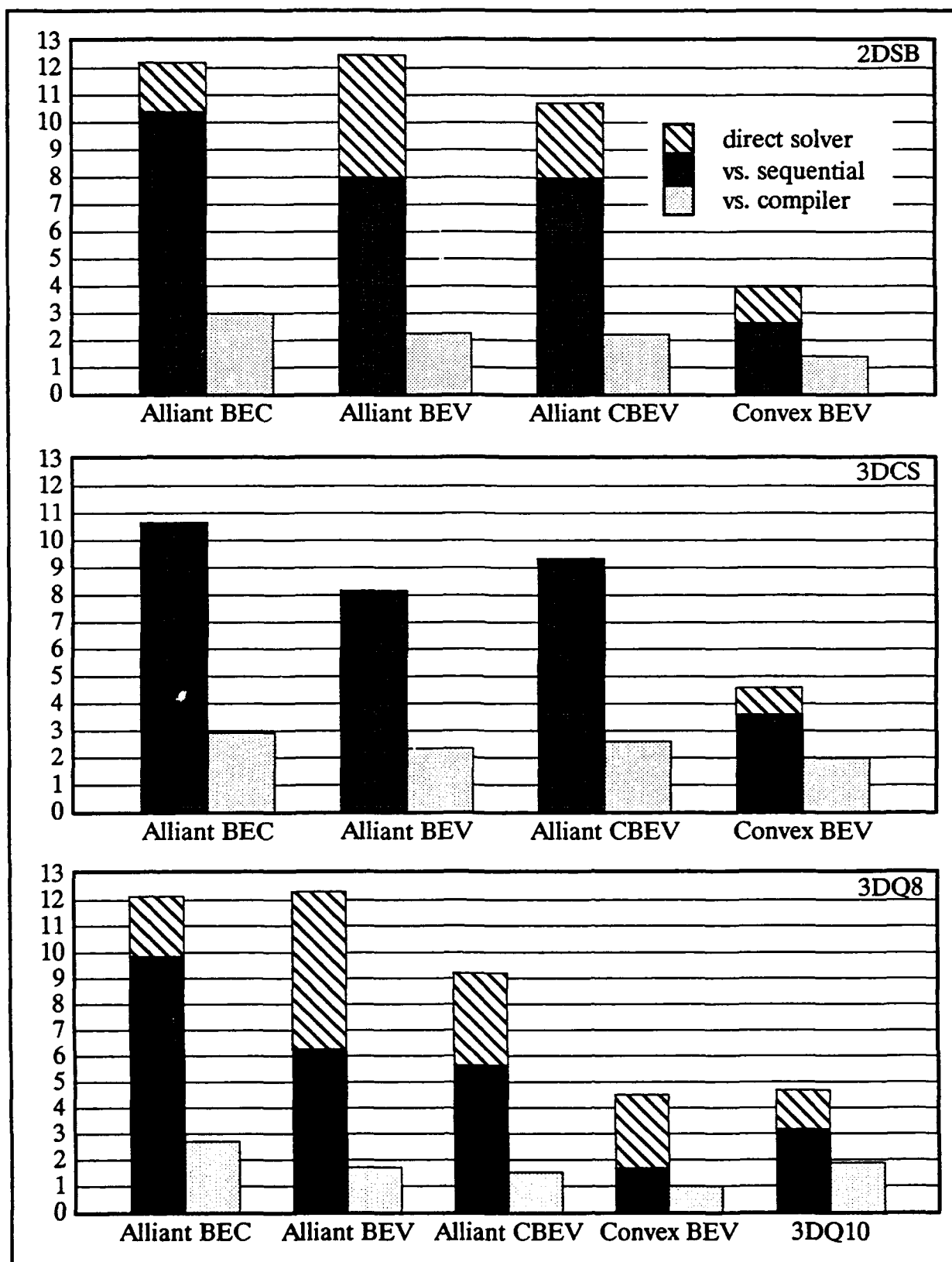


Fig. (4.3-8), speedups for lumped tangent mass matrix, problems with quad. elements

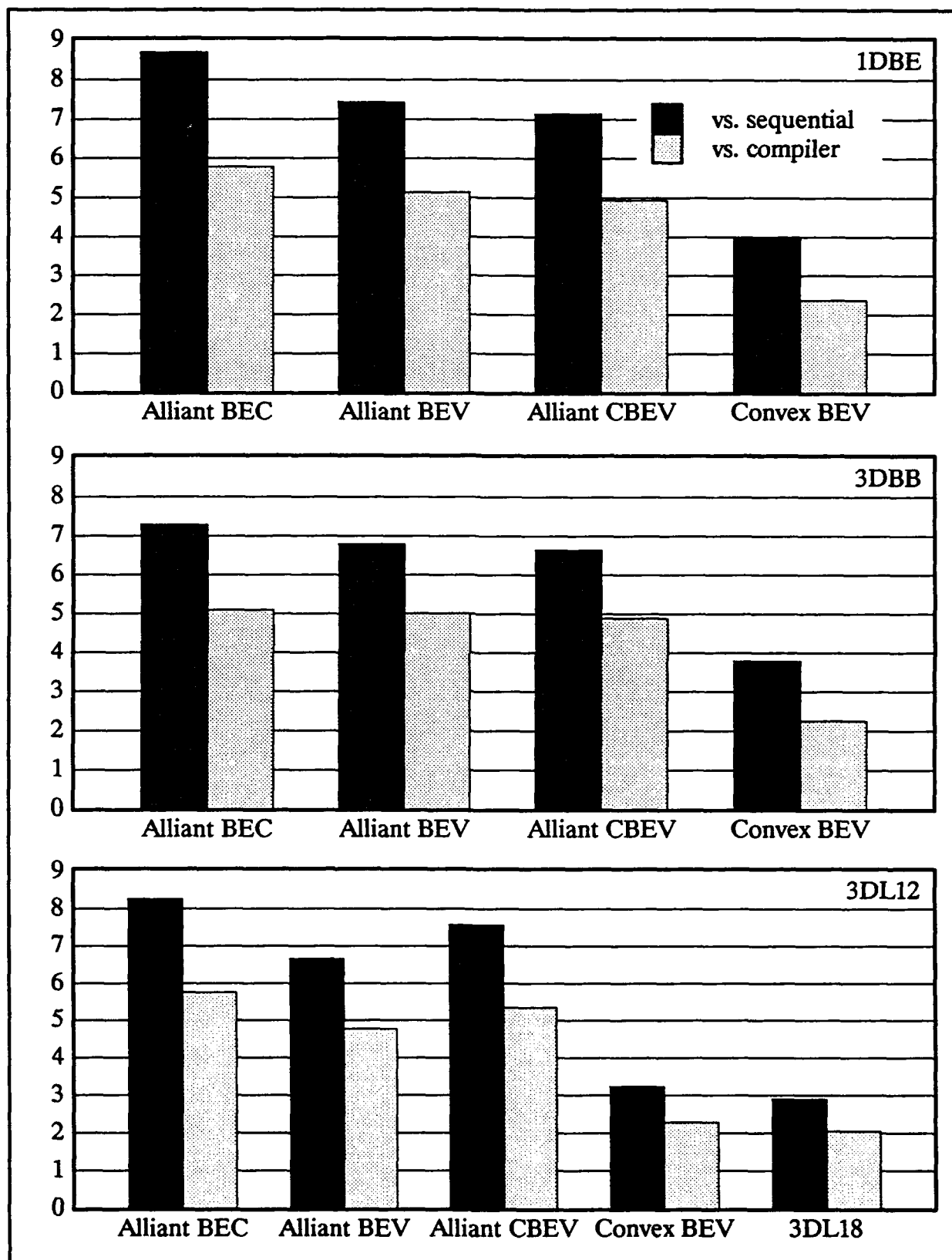


Fig. (4.3-9), speedups for stress recovery, problems with linear elements

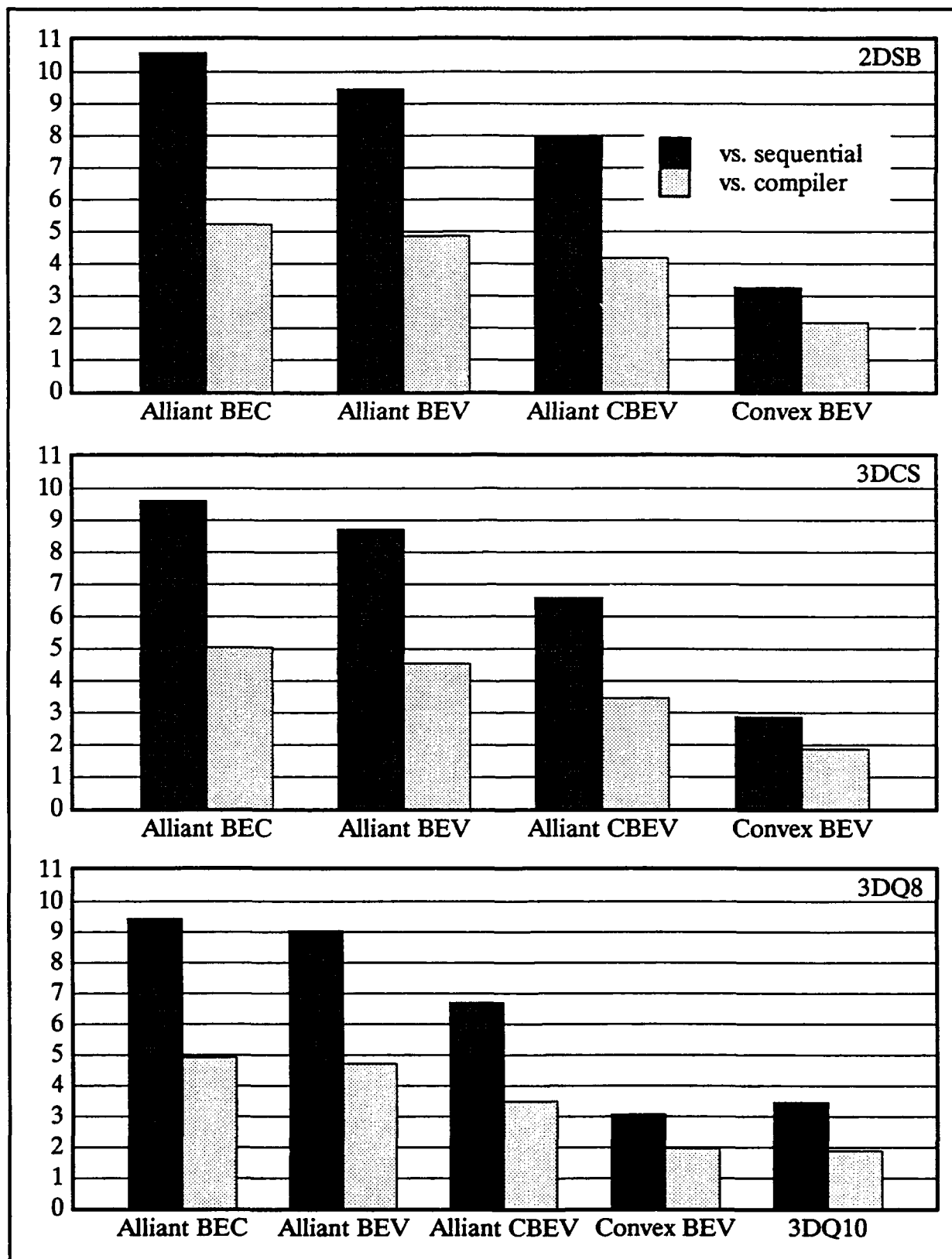


Fig. (4.3-10), speedups for stress recovery, problems with quadratic elements

the LPCG runs, with sequential speedups superimposed on those of the direct solver runs. The greater direct solver speedups are due to an extra scatter operation when computing the lumped mass for LPCG versions of NLD FEP. The element diagonal mass vectors are scattered to the diagonal elements of the element mass matrices in upper triangular vector form. This greatly facilitates the gather operations for the matrix-vector product imbedded in the step length calculation, a very key calculation for the LPCG solvers. A decrease in the efficiency of the lumped mass calculation, whose cost is very little, to achieve an increase in the efficiency of the step length calculation, whose cost is very great, is well worth the effort. Another trend is that the extra scatter costs are greater for those algorithms relying on block vectorization, due to the greater block granularity of these algorithms and a corresponding greater effort required to perform the scatter.

On the Alliant, for linear elements CBEV shows the highest sequential speedups for the direct solver runs, on the order of 13-14, and the greatest effect of the extra scatter operation. However, for the LPCG runs BEC is much more competitive. This indicates that the block granularity and memory requirements of CBEV for the lumped tangent mass matrix calculation are small enough so that designed block vectorization is more effective than incidental vectorization, and that the extra scatter operation all but negates this effectiveness.

For quadratic elements, the incomplete blocking of CBEV renders it noncompetitive and the incidental vector length of BEC is increased, so that on the Alliant the BEC and BEV lumped tangent mass sequential speedups are nearly equal for the direct solver runs (~ 12) and BEC regains its customary predominance for the LPCG runs. The BEV sequential speedups are approximately equal for the direct solver runs of all the example problems.

The Convex BEV lumped tangent mass sequential speedups for the direct solver runs are about 5 for linear elements and about 4-5 for quadratic elements.

Of all the basic element computations, Figs. 4.3-9 and 4.3-10 show the stress recovery to have the smallest sequential speedups. For BEC, this is due to the almost nonexistent incidental vectorization characteristic of this calculation. As the key vector length is the number of stresses (6), and given that the loops are often conditional, the sequential speedups are due almost entirely to concurrency. For the element computation algorithms based on designed block vectorization, the reduced sequential speedups result basically from the conditional loops inherent to the stress recovery, which obstruct the block vectorization. On the other hand, the speedups versus compiler optimization are impressive, often reaching levels comparable with the sequential speedups. This is again the consequence of the meager inci-

dental vector length, which leaves the compiler very little with which to work, and the conditional loops.

For linear elements on the Alliant, BEC shows the greatest stress recovery sequential speedups, with values varying between approximately 7.5 and 8.5. The compiler optimized speedups vary from 5–6, on the order of 60% of the sequential speedups. The CBEV sequential speedups rival or exceed those of BEV, possibly indicating that the designed concurrency of CBEV, by avoiding the conditional loops, is more suitable to the stress recovery than the incidental concurrency of BEV.

For quadratic elements on the Alliant, BEC again claims the largest sequential speedups for the stress recovery. These speedups range from 8.5 to 9.5, somewhat elevated from the speedups for linear elements. This elevation is again due to the increase in the number of element degrees of freedom from 24 to 60, which affects the gather operations and auxiliary calculations. The compiler optimized speedups are about 5, over 50% of the sequential speedups. CBEV is no longer competitive with BEV, as the incomplete CBEV blocking hinders the efficiency of the algorithm.

The Convex BEV stress recovery sequential speedups vary from 3–4 for linear elements and hover about 3 for quadratic elements. The compiler optimized speedups are about 2 for all of the example problems, again on the order of 50–60% of the sequential speedups.

4.3.2.2 LPCG ELEMENT COMPUTATIONS

On the Alliant FX/8, BEC is by far the fastest element computation algorithm for the LPCG element computations using both linear and quadratic elements. Sequential speedups for linear elements range from 15–17 and compiler optimized speedups vary from 8–10. Continuing the trend seen in the basic element computations, as the key incidental vector length rises from 24 to 60 with the use of quadratic elements, the BEC sequential speedups rise to about 16–18 while the compiler optimized speedups drop significantly to about 4–6. The greater decrease in compiler optimized speedups than increase in sequential speedups is due to the nature of the element matrix storage. As these matrices are stored in upper triangular vector form, the LPCG element computations are hampered by indirect addressing and variable vector lengths. For linear elements, with a maximum incidental vector length of 24, the vector-concurrent execution employed by the compiler is relatively inefficient because a significant number of the vector lengths are quite small. As the maximum incidental vector length increases to 60 for quadratic elements, a greater number of the vector lengths are long enough to be executed reasonably well in vector-concurrent mode. Meanwhile, much of the

BEC computational speed is attributable to concurrency, which is unaffected by the increased maximum incidental vector length.

Alliant BEV and CBEV do an adequate and sometimes excellent job on the ebe factorization, but for ebe preconditioning and the step length calculation – which comprise the bulk of the cost for the LPCG linear solvers – they are not competitive with BEC. Apparently, these algorithms do not utilize the cache memory of the Alliant well as the consequence of their greater block granularity and memory requirements. BEC is efficient because it utilizes the cache memory of the Alliant much more efficiently. In fact, the BEC matrix-vector multiply involved in the step length calculation is performed in the fashion of the sub-matrix partitioning technique that has proven to be the most efficient for the hierarchical memory architecture of the Alliant FX/8 [43].

The lone exception to the rule of BEC supremacy on the Alliant is the ebe factorization for 3DBB. The BEC speedups for this calculation are uniquely poor. This may be due to the relatively large number of elements combined with the following oversight in the gather/scatter operations for the ebe factorization. Instead of gathering the appropriate element data to temporary storage, computing, and then scattering back to global storage, the ebe factorization was performed on the global storage itself. As observed in the tangent stiffness calculation, the number of elements aggravates the overhead in gather/scatter, possibly due to system paging costs, and not using temporary storage causes this overhead to interrupt the calculations themselves.

The LPCG element computation speedups for Convex BEV are about the same for linear and quadratic elements. For the ebe factorization, the sequential speedups are in the range of 3–4, for the ebe preconditioning about 4–5, and for the step length calculation about 5–6. These speedups are respectable but somewhat disappointing, although they are comparable to those of the basic element computations. As stated in Section 4.2, the sequential speedups of the LPCG solvers lag behind that of the direct solver, and the LPCG element computations are primarily responsible. One problem may be the larger block granularity and memory requirements of Convex BEV, which may inhibit efficient vectorization with prohibitive overhead. Another may be the upper triangular vector form of the element matrices. Because of this form, there is a rather complicated outer loop structure about the inner element block vector loops. The compiler may not be able to create efficient vector code for this calculation. Unrolling the outer loops would be a messy and cumbersome chore, but it may be

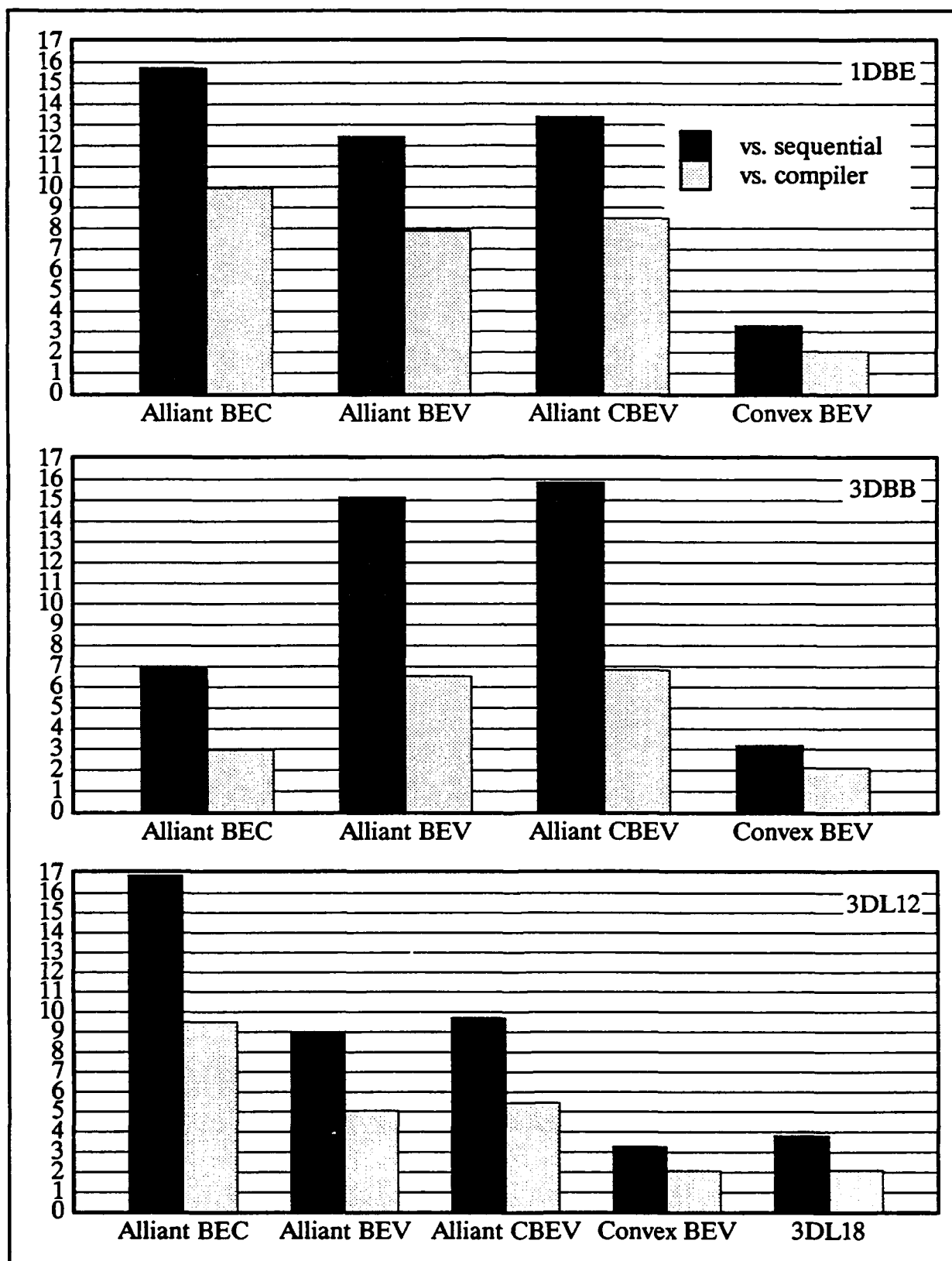


Fig. (4.3-11), speedups for ebe factorization, problems with linear elements

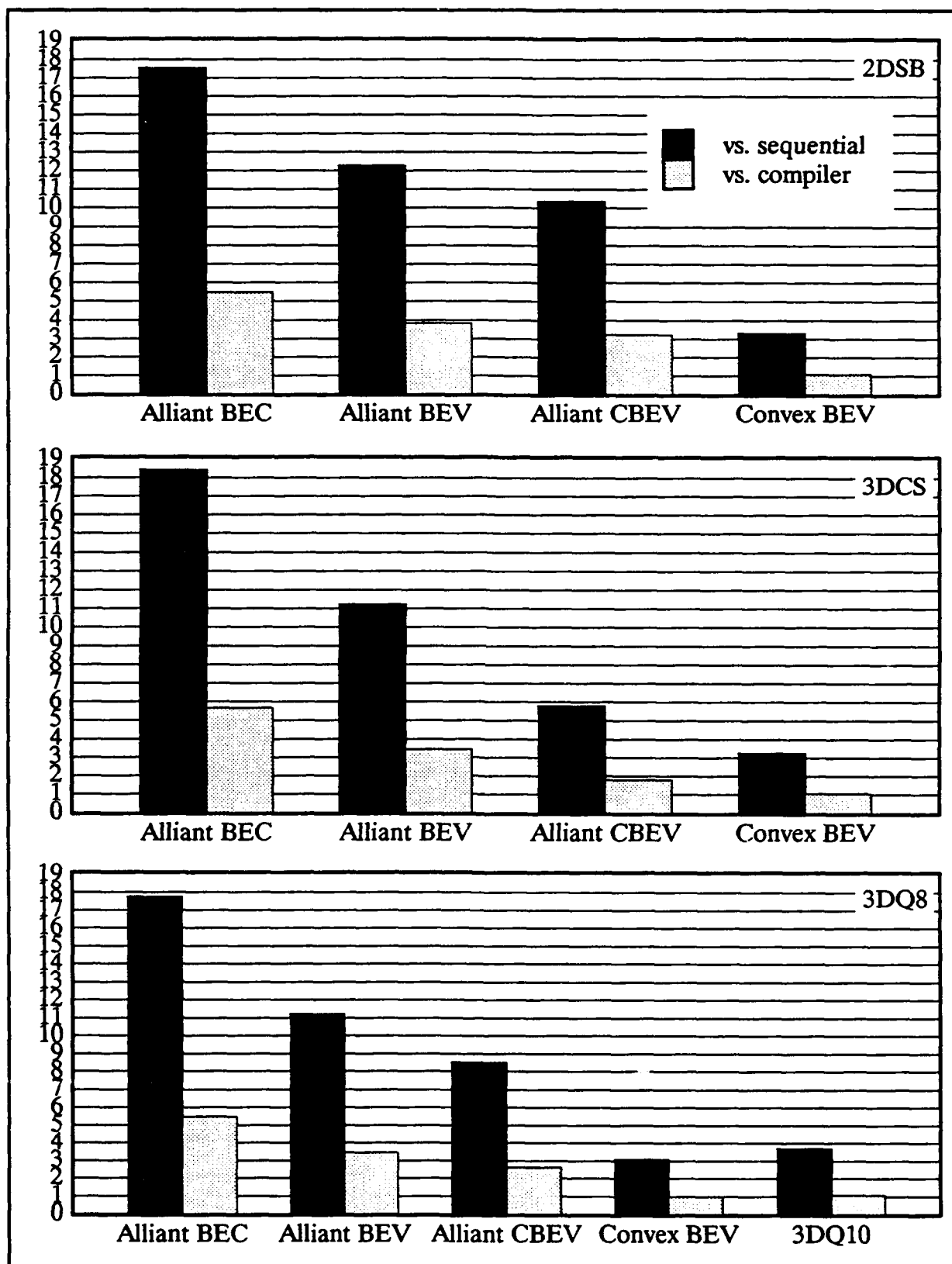


Fig. (4.3-12), speedups for ebe factorization, problems with quadratic elements

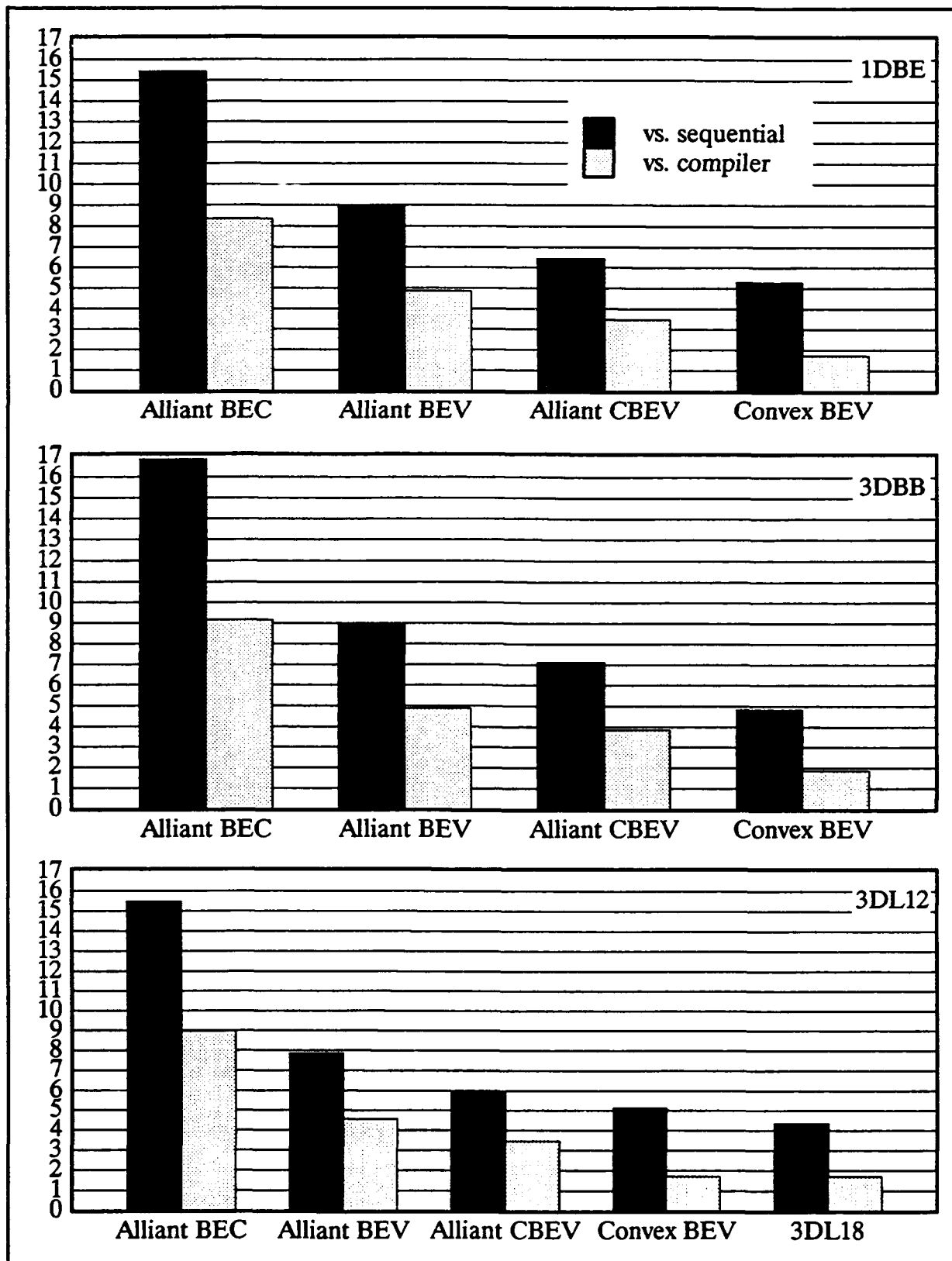


Fig. (4.3-13), speedups for ebe preconditioning, problems with linear elements

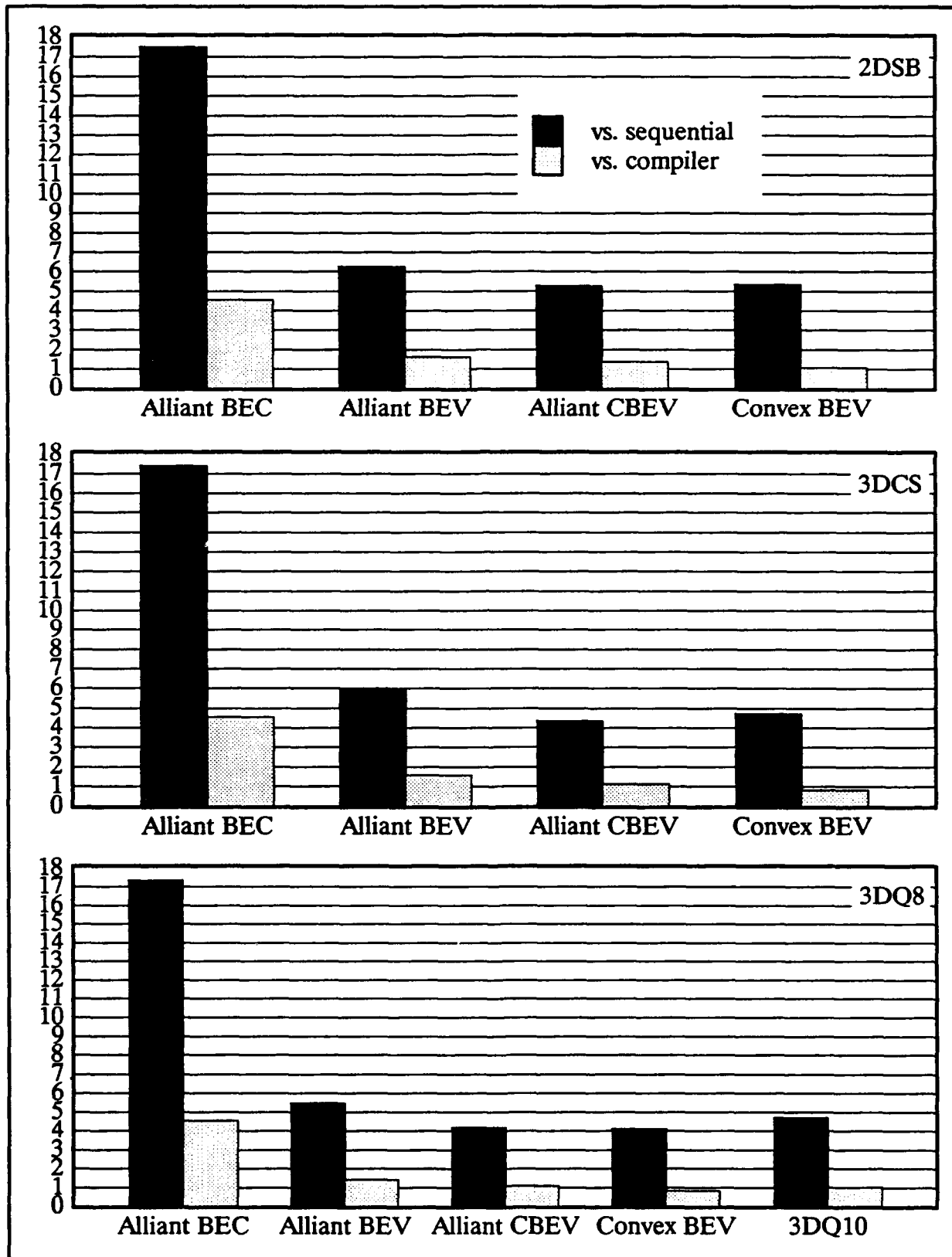


Fig. (4.3-14), speedups for ebe preconditioning, problems with quadratic elements

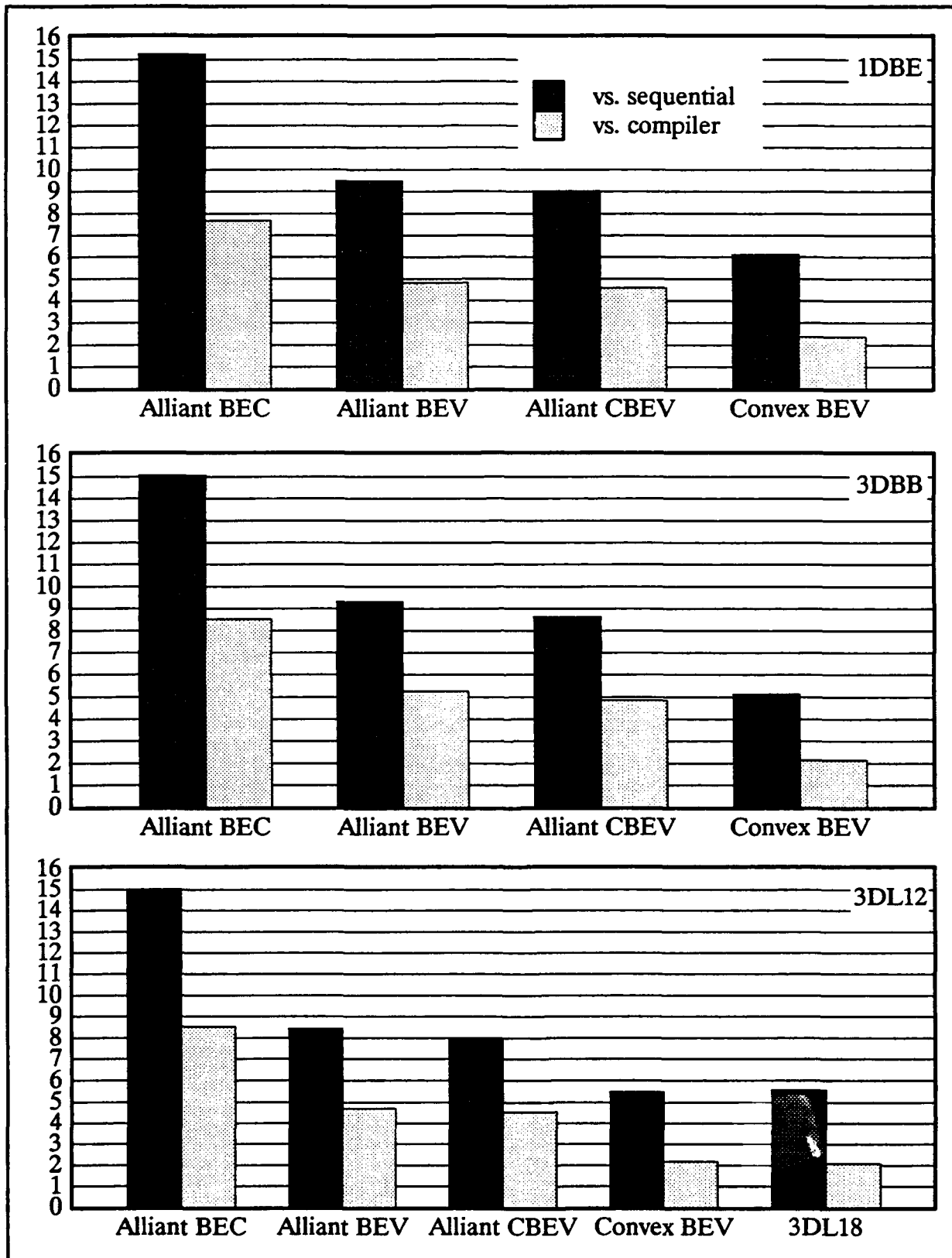


Fig. (4.3-15), speedups for step length calculation, problems with linear elements

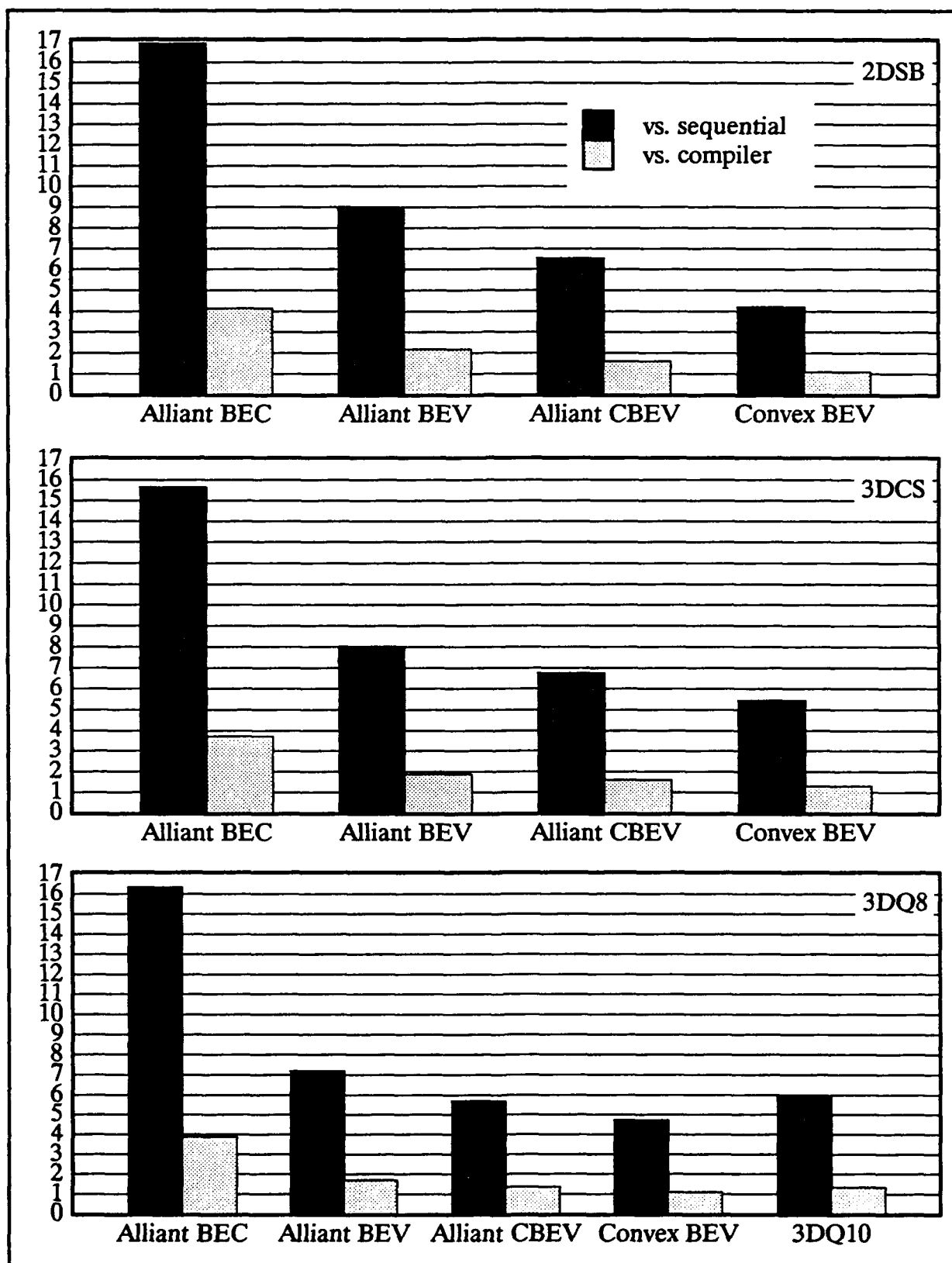


Fig. (4.3-16), speedups for step length calculation, problems with quadratic elements

necessary to improve the performance of the LPCG element computations, and perhaps those of the basic element computations as well.

For linear elements, the Convex BEV compiler optimized speedups for the LPCG element computations are on the order of 2, so that Convex BEV is seen to be a meaningful improvement over compiler optimization. For quadratic elements, however, the compiler optimized speedups are on the order of 1. The step length calculation is slightly better in this respect than the others, with compiler optimized speedups approaching 1.5, but otherwise there is little difference between Convex BEV and compiler optimization. A major reason for the disappointing performance of the ebe factorization and preconditioning is that BEV requires the preconditioning matrices for the element block to be gathered before processing for these calculations while the sequential application does not. As much as 30–40% of the BEV execution time for these calculations is spent in the gather operations. The gather operations also inhibit the sequential speedups for these calculations. Another reason for the poor compiler optimized speedups of quadratic elements is the increased maximum incidental vector length of the quadratic elements, to which the compiler can apply the more powerful vector capability of the Convex to good effect.

The necessity of gather operations exist for the Alliant element computation algorithms as well. These gather costs affect BEV and CBEV more than BEC and degrade their performance relative to BEC. However, the gather costs are more significant for Convex BEV because concurrency helps to alleviate these costs.

Another factor inhibits the sequential speedups for the step length calculation using Convex BEV. From Fig. 4.3–16 the Convex BEV sequential speedup for the 2DSB step length calculation is about 4, which is noticeably less than its counterparts for the other example problems. One possible explanation is related to the large number of constrained degrees of freedom in 2DSB, due to its plane strain boundary conditions. In order to avoid the cost in memory of storing the constrained element dynamic tangent stiffness matrices, for all of the element computation algorithms (including the sequential application) these matrices are assembled and constrained before the step length matrix–vector product is performed. For the Alliant element computation algorithms, the constraint process can be performed very effectively using concurrency. In fact, the BEC and Alliant BEV step length sequential speedups are greater for 2DSB than the other example problems employing quadratic elements. Unfortunately, this is not an option for Convex BEV, so that the constraint process suffers. With so many constrained degrees of freedom spread over so many elements in 2DSB, the con-

straint process can seriously disrupt the efficiency of the matrix-vector product. Thus, the speedups would be lowered.

4.3.3 SUMMARY

On the Alliant FX/8, BEC proves itself to be the best element computation algorithm in nearly every aspect. One reason for this superiority is reason is the smaller block granularity of BEC. As a result, the BEC gather/scatter operations are more streamlined, with less elements to process and those elements likely to be more closely bunched in numbering so that there is less possibility of virtual memory paging to access the block elements. It takes a large number of elements, as in 3DBB, for the BEC gather/scatter operations to be adversely affected, while for the lesser but still significant number of elements in 3DL12 BEC is unaffected and BEV and CBEV remain hindered. The block granularity advantage of BEC is even more telling for quadratic elements, given the greater number of element degrees of freedom and the relatively few number of elements in a quadratic mesh. The smaller block granularity of BEC results from the fact that there are fewer vector processors, 8, than the length of the vector registers, 32, so that when optimally blocked BEC requires four times less element block data structure memory than BEV. CBEV relies on both the number of vector processors and the length of the vector registers and thus requires 32 times the element block data structure memory of BEC.

A second, more significant reason for the superiority of BEC is its smaller, designed parallelizable granularity. One important ramification of this is fewer cache misses for BEC computations. Given an aggregate Alliant FX/8 cache memory of 128 kilobytes, a large part of the entire element block data structure of BEC for even quadratic elements can fit in the cache, and the entire structure is not required for a given calculation. Thus, BEC data is more likely to be in cache when needed by a vector processor. Conversely, the fully blocked CBEV element block data structure far exceeds the capacity of the cache and calculations are hampered by the transfer of data between the cache and main memory. BEV data also has difficulty fitting in cache, although BEV performance for the basic element computations of most of the example problems employing meshes composed of linear elements nearly equaled that of BEC. However, for quadratic elements and for the LPCG element computations such as determination of the step length and the ebe preconditioning BEV is noticeably inferior, probably in large part due to ineffective use of the cache.

A second ramification of the smaller, designed parallelizable granularity of BEC is a reduction in bus traffic, both in volume and frequency. In volume, for instance, so much data needs to be manipulated for CBEV that for some problems, namely 3DBB, some of the data

is lost and the solution is compromised. BEC never approaches this problem, while at one time BEV did. Early on, the element diagonal scaling step of the ebe preconditioning was performed on an element-by-element basis as an alternative to performing it on the global level. The element-by-element scheme was abandoned due to its inefficiency compared to the global scaling for block vectorized methods. However, while it was in use, BEV had the same problem for 3DBB as does CBEV. Removing the element-by-element scaling reduced the amount of data manipulation necessary and BEV was no longer afflicted.

In frequency, the incidental concurrency of BEV requires more communication between processors than the designed concurrency of BEC and overhead is increased. Overhead is also increased by another feature of incidental concurrency: its reliance on the outer loops of calculations to determine the parallelism of COVI mode execution. Where the designed concurrency of BEC provides for equal load of all eight vector processors, sometimes the sizes of the outer loops may not be an even multiple of eight. This can be seen in the decrease in efficiency from linear to quadratic elements, as the number of degrees of freedom and often the outer loop size goes from 24, an even multiple, to 60, which is not. Also, in the matrix-vector product of the BEV step length calculation, the outer loop sizes are variable, which also decreases COVI overhead and efficiency.

A third reason for the excellent performance of BEC is the adequate to good incidental vectorization found in the 3D isoparametric elements. Often, the incidental vector lengths equal the number of degrees of freedom, which at 24 and 60 for linear and quadratic elements fairly utilize the size of the vector registers, especially for the quadratic elements. The latter can be seen in the improvement of speedups from linear to quadratic elements in the example problems. Even when the upper triangular vector storage of the element matrices causes interrupted vector calculations and indirect addressing, as in the matrix-vector product of the step length calculation, the vector performance within BEC is reasonably effective, again especially for the quadratic elements.

A final reason for the pre-eminence of BEC is the consistently good element blocking obtained for all sizes of problems. Although a main concern of this research is large problems, and for large problems will probably be enough elements to adequately block each element computation algorithm, there is a relatively small limit to the size of problem that can be run on the Alliant FX/8 and many times the number of elements for such a limited problem size is insufficient to satisfactorily block CBEV and even BEV. Further, as the arrangement of the elements becomes more three dimensional – the desirable situation – the interdependency of elements becomes more prevalent and it is more difficult to create sufficiently large blocks of nonconflicting elements for a given number of elements. For Alliant BEC, however,

only eight nonconflicting elements are needed for a block, and it is generally true that effective blocking can be provided for BEC regardless of the problem size.

On the Convex C240, the performance of BEV is adequate but not as effective as one might expect. The efficiency of LPCG element computations is particularly disappointing, especially the speedups versus compiler optimization using quadratic elements. One reason for this could be the large block granularity of Convex BEV. This is the consequence of the larger length of the Convex vector registers, 128, and the attempt to create a commensurate vectorizable granularity. The same problems with gather/scatter operations and cache memory utilization outlined above can occur here, with the added problem that the Convex runs were made during multiuser time and may compete with other memory intensive jobs, so that manipulating the large amount of data required by BEV may lead to more operating system overhead time.

Gather operations are a second reason for the lukewarm Convex BEV performance in the LPCG element computations. These operations represent a significant portion of the computation costs and since the sequential application does not require them the speedups of Convex BEV are damaged. The lack of concurrency seems to exacerbate this problem as the gather costs of Alliant BEV and CBEV are not as damaging.

A third aspect of Convex BEV that may be harmful to its efficiency is that, since there is no concurrency, the outer loops are not optimized and in some cases the compiler may not be able to create efficient vector object code. Specifically, the LPCG element computations, due to the upper triangular vector storage of the element matrices, contain relatively complicated outer loop structures that are very difficult to unroll and that the compiler may not be able to handle effectively. To unroll these loops would require very long-winded and detailed code, and the attempt was not made. Unfortunately, without unrolling, the compiler may be doing far too many fetches and stores compared to vector operations. However, even if such loops are unrolled there are indications that the Convex does not perform particularly well, either.

There are some difficulties in providing an effective element blocking for BEV given the large size of the Convex vector registers, an example of which is 3DQ8 where the vectorizable granularity provided is only half the size of the vector registers. However, in 3DQ10 where the number of elements is merely doubled the blocking is nearly optimal. Consequently, in general the element blocking is not much of a problem for Convex BEV, although for smaller to moderate size problems with a more three dimensional mesh and quadratic elements, like 3DQ8 and perhaps 3DCS, pose relatively minor problems.

4.4 NONLINEAR SOLUTION ALGORITHMS

In this section, the results of the nonlinear solution algorithm comparisons are presented and discussed. Of the example problems, results are presented only for the 3DBB and 3D finite extension problems, as described in Section 4.1. A summary is provided at the end of the section.

4.4.1 3DBB

Figs. 4.4-1 to 4.4-3 contain the timing data for the 3DBB runs updating the tangent stiffness before every (nonlinear) equilibrium iteration in the range of significant nonlinear behaviour. Figs. 4.4-4 to 4.4-6 contain the timing data for the 3DBB runs updating the tangent stiffness before just the first two equilibrium iterations in this range. In all figures, data is presented for three separate cases. The first case corresponds to the engineering analysis of Section 3.3 and employs 10 equal time steps of 0.03457712 milliseconds to reach the loading flat top. In the first 5 time steps, the tangent stiffness is updated before the first equilibrium iteration only, while the second 5 time steps encompass the range of significant nonlinear behaviour. The LPCG convergence tolerance is set to 0.01%. The second case is identical to the first except the LPCG convergence tolerance is relaxed to 1.0%. In the third case, the loading flat top is reached in 1 time step of 0.3457712 milliseconds. The LPCG convergence tolerance applied in the third case is again 0.01%.

The line search tolerance employed for all NLPCG runs is 1.0%. By varying the line search tolerance between 0.01% and 10% for the first case above and for both tangent stiffness updating strategies, it was observed that the relatively relaxed 1.0% tolerance generally lead to the either the fastest or nearly the fastest NLPCG solution in terms of total execution time. Eliminating the line search altogether – by assigning a step length of 1.0 – significantly degraded the NLPCG nonlinear convergence rates and the corresponding increase in linear iterations and system solution costs more than offset the savings in basic element computations. This indicates that a line search is necessary for adequate NLPCG performance.

For the 0.01% LPCG convergence tolerance and both tangent stiffness update strategies, NLPCG is noticeably slower than secant-Newton even though it requires an identical number of equilibrium iterations. When the tangent stiffness is updated before every equilibrium iteration MNR shows the same nonlinear convergence rate as the accelerated algorithms, so that one would not expect NLPCG to have an advantage in equilibrium iterations over secant-Newton. However, reducing the number of tangent stiffness updates leads

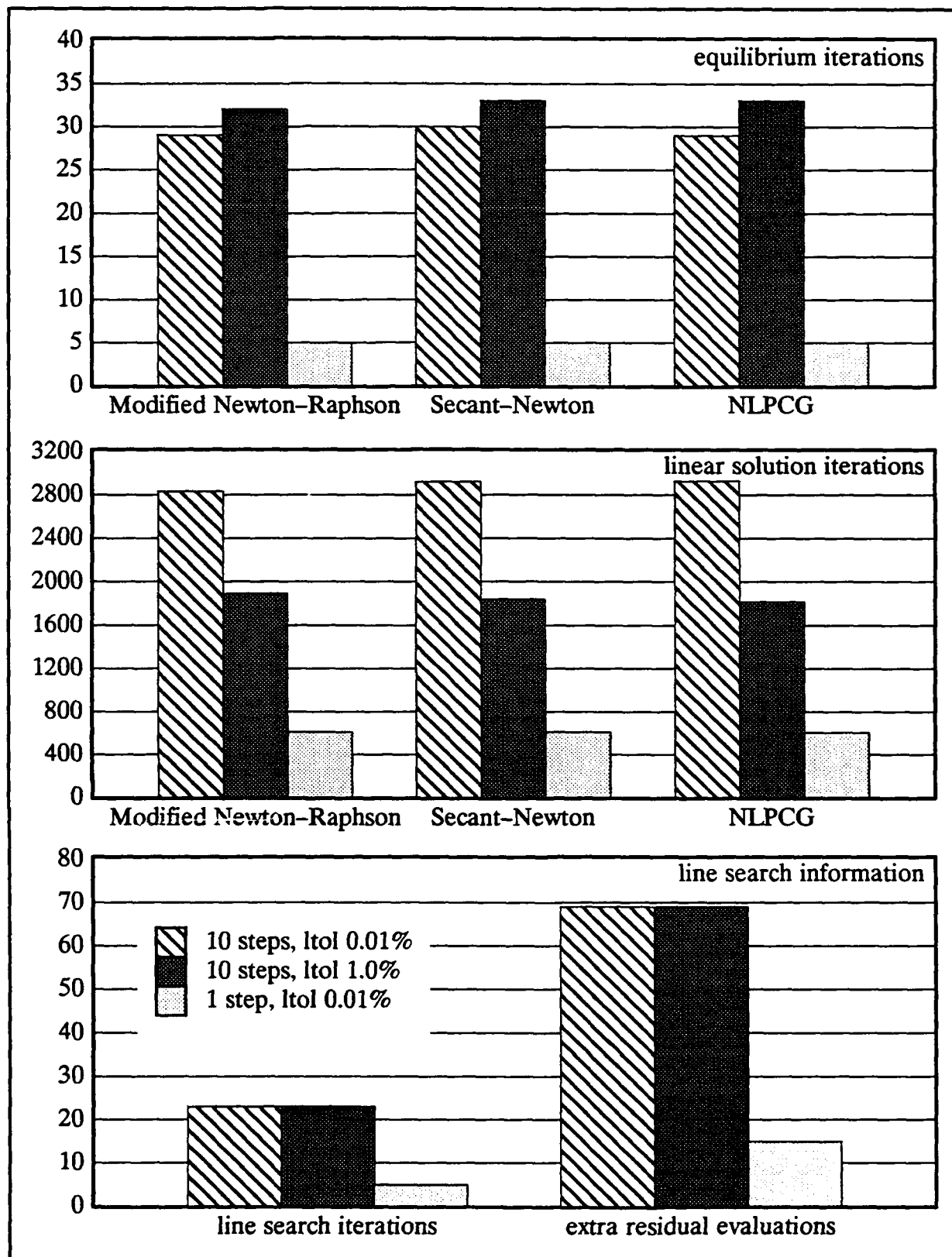


Fig. (4.4-1), 3DBB timings – stiffness updates all equilibrium its.

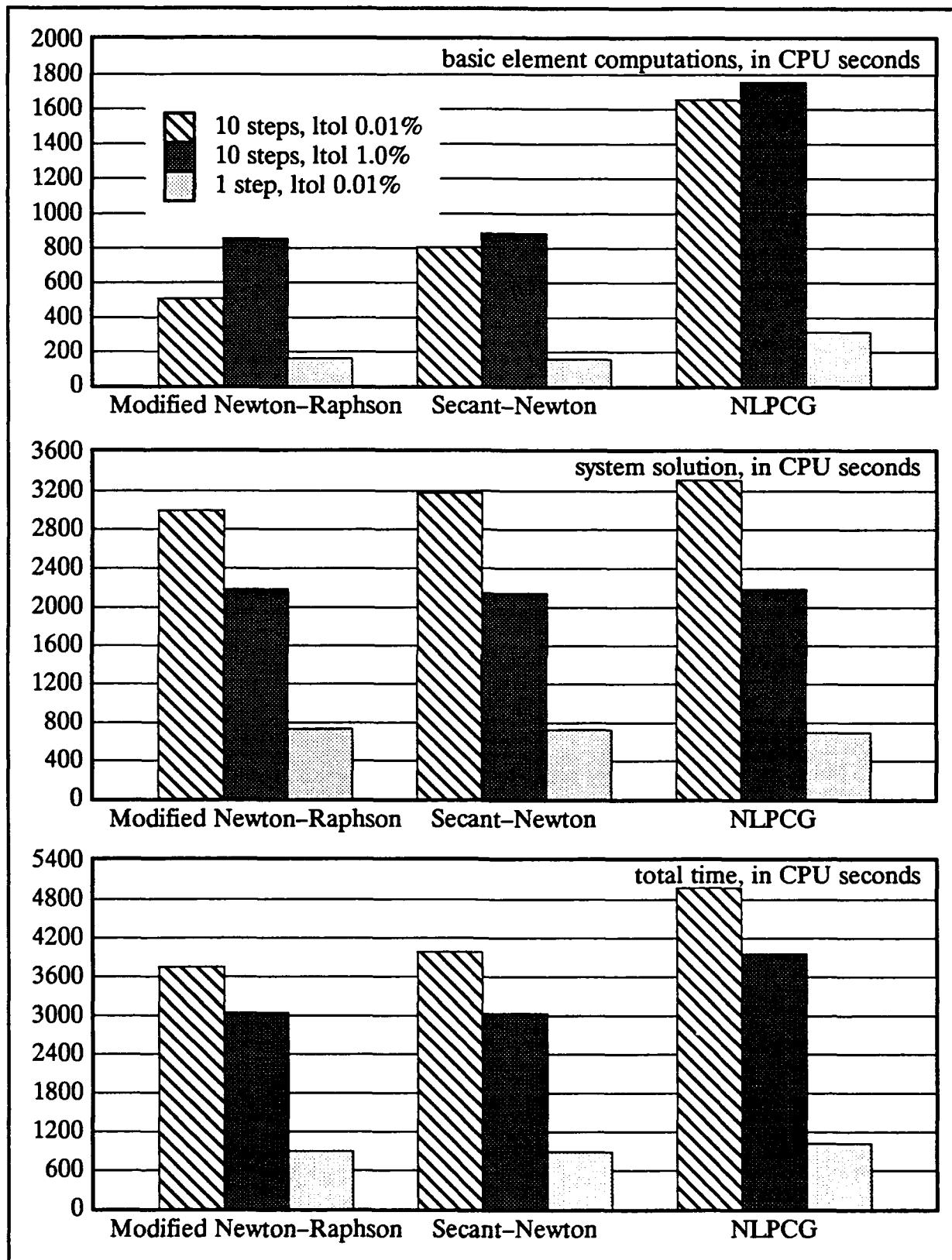


Fig. (4.4-2), 3DBB timings – stiffness updates all equilibrium its., cont.

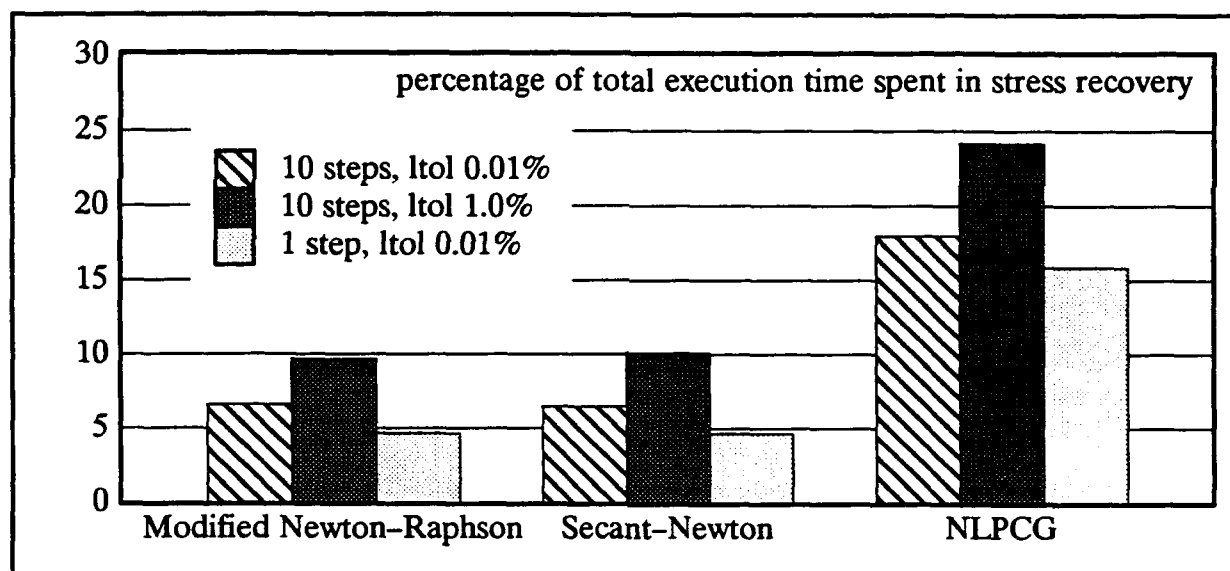


Fig. (4.4-3), 3DBB timings – stiffness updates all equilibrium its., cont.

to a measurably slower MNR nonlinear convergence rate relative to secant-Newton, and even in this case NLPCG converges no faster than secant-Newton. Even though the NLPCG line search for both tangent stiffness update strategies requires less than 3 extra residual evaluations per equilibrium iteration, without an advantage in equilibrium iterations the necessary line search cost simply represents additional overhead and NLPCG is at a disadvantage.

The results concerning the 1% LPCG convergence tolerance illustrate that relaxing the LPCG convergence tolerance can lead to improved performance for both tangent stiffness updating strategies. For all three nonlinear solution algorithms, the number of performed linear iterations are greatly reduced while the number of equilibrium iterations is increased by a lesser amount. The corresponding decrease in system solution costs more than compensate for the increased basic element computation times and the total execution times are significantly decreased, although the relaxed tolerance does not noticeably alter the relative speed of the secant-Newton and NLPCG. This indicates that as long as the analyst is satisfied with the accuracy of the stress analysis, to achieve optimal LPCG performance for a given problem and nonlinear solution algorithm one should relax the LPCG tolerance and compensate with increased equilibrium iterations until the trade-off ceases to result in greater computational speed.

The third case defined above utilizes a time step size 10 times greater than that specified in the first two cases. For this case, all three algorithms employing tangent stiffness updates before every iteration execute the same number of equilibrium iterations, so that MNR and secant-Newton are practically equal in total execution time while NLPCG is again

slower due to the line search. However, reducing the number of tangent stiffness updates leads to a very poor nonlinear convergence rate for MNR so that secant-Newton and NLPCG require significantly fewer equilibrium iterations, with NLPCG requiring the fewest. Because of its slightly improved nonlinear convergence rate, and because the greater time step size causes LPCG to execute about 20% more linear iterations per equilibrium iteration than in the first case, secant-Newton and NLPCG finish in a virtual dead heat. The significance of the increased LPCG convergence rate is that the cost of the line search is de-emphasized, which can be seen in a reduced percentage of the total execution time spent in the stress recovery.

Obviously, executing fewer time steps in an implicit dynamic analysis – like solving fewer load steps in a static analysis – often leads to faster total execution times. The question remains: how does increasing the time step size affect the comparison of the nonlinear solution algorithms? For 3DBB, the answer seems to be that NLPCG benefits the most. Considering the third cases of both tangent stiffness update strategies, the larger time step size leads to a greater system solution cost per equilibrium iteration that devalues the debilitating effect of the NLPCG line search, and NLPCG is better able to handle the degradation of the nonlinear convergence rate that can result from a larger time step size. This is not to say that NLPCG is the fastest algorithm in both third cases; secant-Newton is faster. NLPCG simply improves the most. This implies that for a given problem and set of convergence criteria one might be able to adopt different time step sizes – all of which allow for good LPCG performance and acceptable simulation accuracy – which produce the best performance for each algorithm and which results in NLPCG being the fastest algorithm overall.

Figs. 4.4–3 and 4.4–6 indicate that the fastest 3DBB run in each case is that of MNR or secant-Newton with tangent stiffness updates before every equilibrium iteration in the range of nonlinear behaviour. Considering that all of the timings presented in this section employ LPCG as the linear solver, this suggests that when using LPCG one would want to minimize the number of total equilibrium iterations by performing frequent tangent stiffness updates.

The cost of the line search is dependent upon the number of extra residual evaluations performed. To evaluate the residual, both a stress recovery and an internal force vector update are required. For 3DBB, the stress recovery is approximately 3 times more costly than the internal force calculation. This relative cost seems to result primarily from three factors. First, when considering linear elements, there is a greater amount of computation necessary for the stress recovery based on incremental J_2 flow theory than there is for the internal force

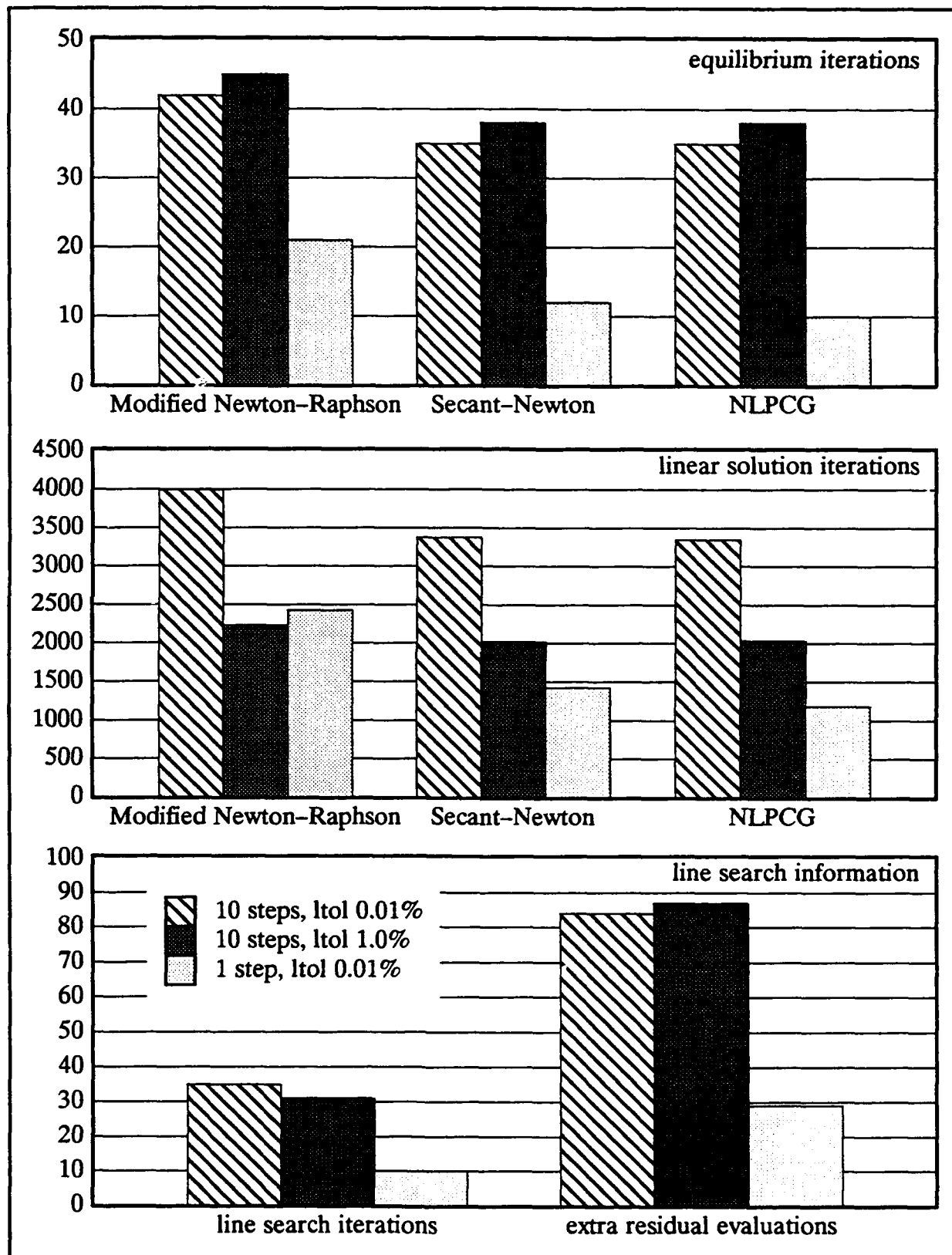


Fig. (4.4-4), 3DBB timings - stiffness updates first two equilibrium its.

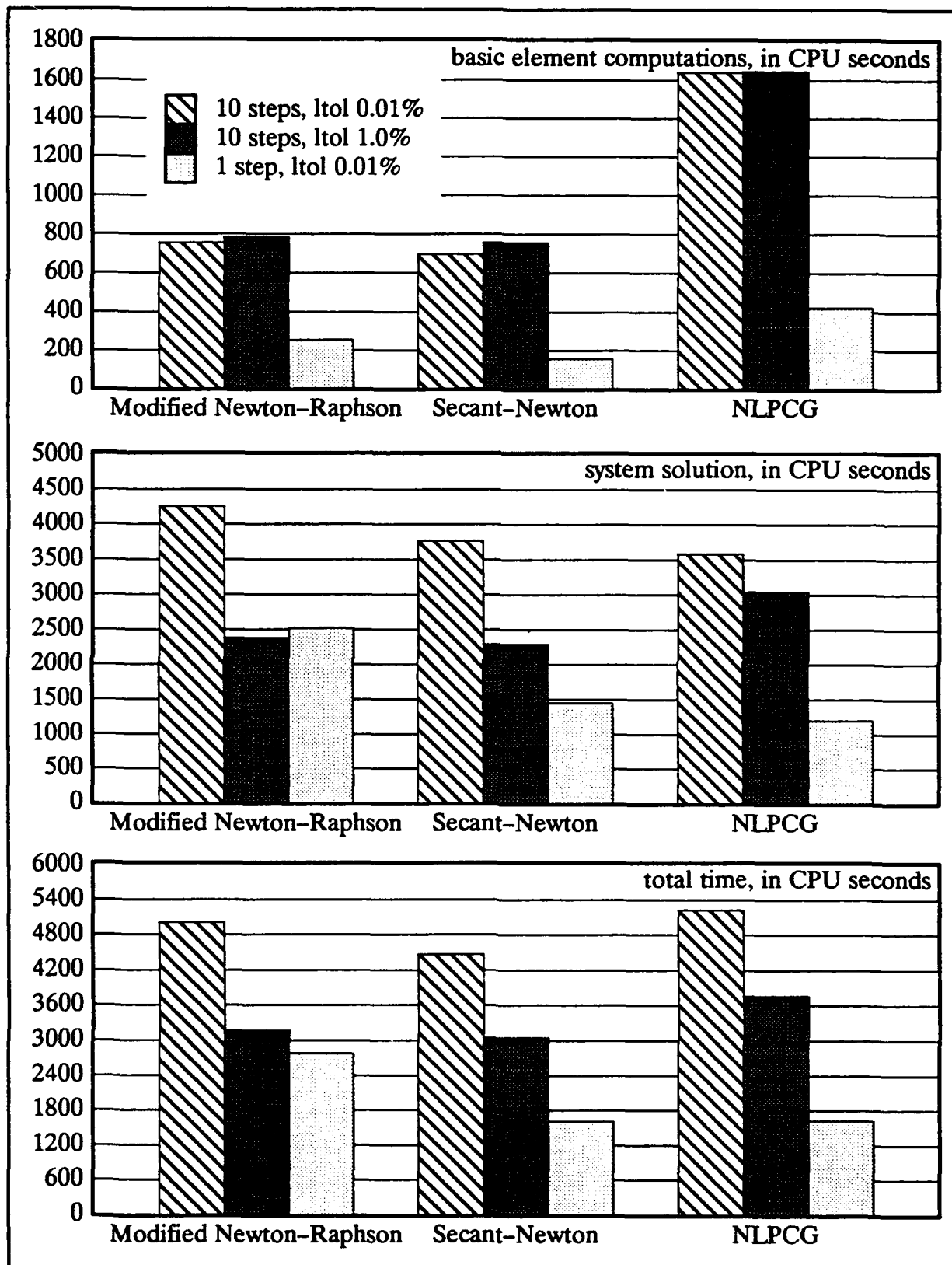


Fig. (4.4-5), 3DBB timings - stiffness updates first two equilibrium its., cont

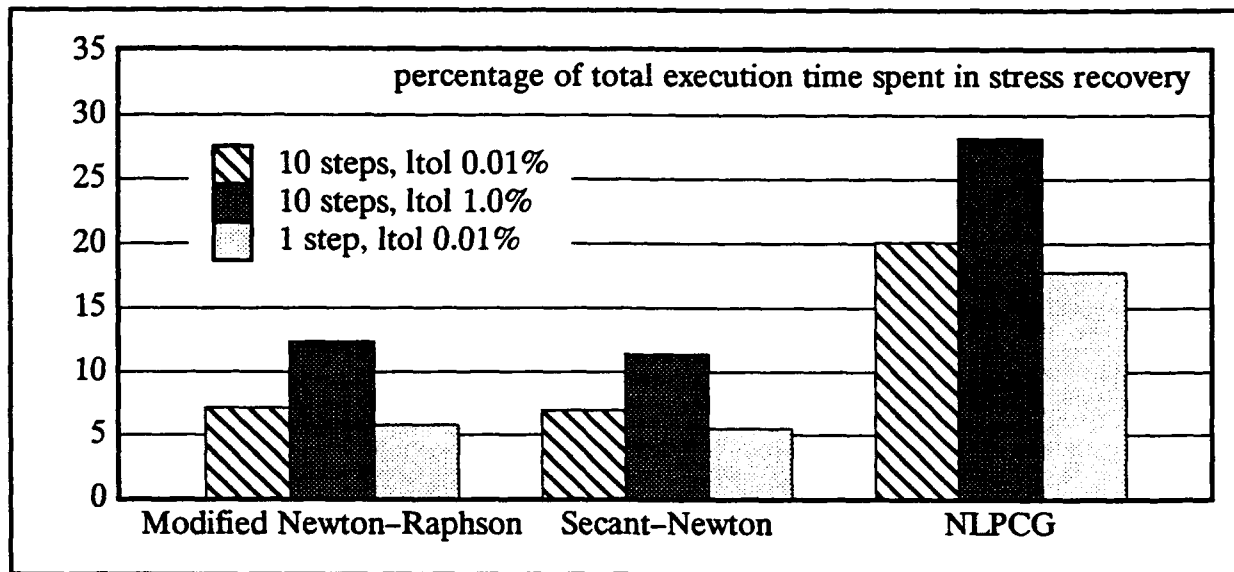


Fig. (4.4-6), 3DBB timings - stiffness updates first two equilibrium its., cont.

calculation. Second, employing the unrotated formulation with its required polar decompositions in the stress recovery leads to even more computations. Third, in Section 4.3 it was observed that the element computation algorithms are least effective for the stress recovery. Thus, the internal force calculation can be performed somewhat more efficiently. Based on the relative cost of the stress recovery and the internal force calculation, the cost of the line search for the linear elements of 3DBB appears to hinge primarily upon the stress recovery.

For the three cases solved with either tangent stiffness update strategy, NLPCG - which employs a line search - spends far more of the total execution time in the stress recovery than does MNR and secant-Newton. By employing the Truesdell formulation, the cost of the stress recovery for 3DBB is reduced by 55-60% on the Convex C240. This represents a fairly small savings in total execution time for those runs not employing a line search, but as much as an 18% savings for those that do. This is a significant number, and suggests that the Truesdell formulation is better suited to NLPCG than is the unrotated formulation. However, Section 3.5 indicates that the Truesdell formulation is not suitable for the kinematic hardening specified in 3DBB, so that improvement is necessary for the Truesdell formulation before it could be realistically applied to a problem such as 3DBB.

On the Alliant FX/8, the Truesdell formulation is only about 20% faster than the unrotated formulation, which is reasonable considering the cost of the polar decomposition. The difference between machines is that the use of concurrency significantly enhances the speed of the eigenvalue extraction during the polar decomposition, while attempts to vectorize the Jacobi iterations are less successful. Thus, the cost of the unrotated formulation on

the Convex is due to the failure of effort (to date) to optimize the computations. If future attempts at optimizing the polar decomposition on the Convex are more successful, then the relative performance of NLPCG in the analyses considered in this section should show some improvement.

4.4.2 3D FINITE EXTENSION PROBLEMS

The timing data for the 3DL12 nonlinear solution algorithm comparisons is displayed in Figs. 4.4–7 to 4.4–9. Data is presented for both a LPCG convergence tolerance of 0.01% for both of the load step arrangements discussed in Section 4.1 and one of 1% for the smaller load step arrangement.

The 0.01% smaller load step data shows NLPCG to be over 2.5 times faster than MNR. This results from a great advantage in equilibrium iterations for NLPCG. Enough equilibrium iterations are performed by MNR to engender basic element computation costs that exceed the cost of the NLPCG line search. More importantly, the equilibrium iteration advantage translates into thousands less linear iterations, which creates a large NLPCG advantage in the system solution expense that dominates the total execution time. Even if zero time was spent in basic element computations for MNR, NLPCG would still be far superior in total execution time. In this instance, the cost of the line search is relatively insignificant.

For the 0.01% tolerance and smaller load steps, NLPCG also shows an advantage in equilibrium iterations versus Secant–Newton. This advantage is far less than that which exists versus MNR. As such, the comparison of Secant–Newton and NLPCG boils down to a trade-off between the lesser basic element computation costs of Secant–Newton and the lesser system solution time of NLPCG. The total execution times show this conflict resolved in favor of NLPCG, but only by a slight margin. Considering that the data presented pertains to a line search tolerance of 1%, and that a relaxation of the NLPCG line search tolerance from 0.01% to 1% resulted in a significant improvement in the line search cost (about 30% fewer residual evaluations and approximately a 9% decrease in the total execution time) without a loss in the nonlinear convergence rate, it is certainly possible that further improvement could be generated by continuing to increase the line search tolerance so that NLPCG would gain versus Secant–Newton. However, such a gain would likely be small in relation to the overall execution time and the result of the comparison of the two algorithms would probably be largely unchanged.

The relaxation of the LPCG convergence tolerance to 1% decreases the execution time advantage of NLPCG relative to MNR (only twice as fast), and allows secant–Newton to overtake NLPCG as the fastest algorithm. The relaxed tolerance decreases the number of

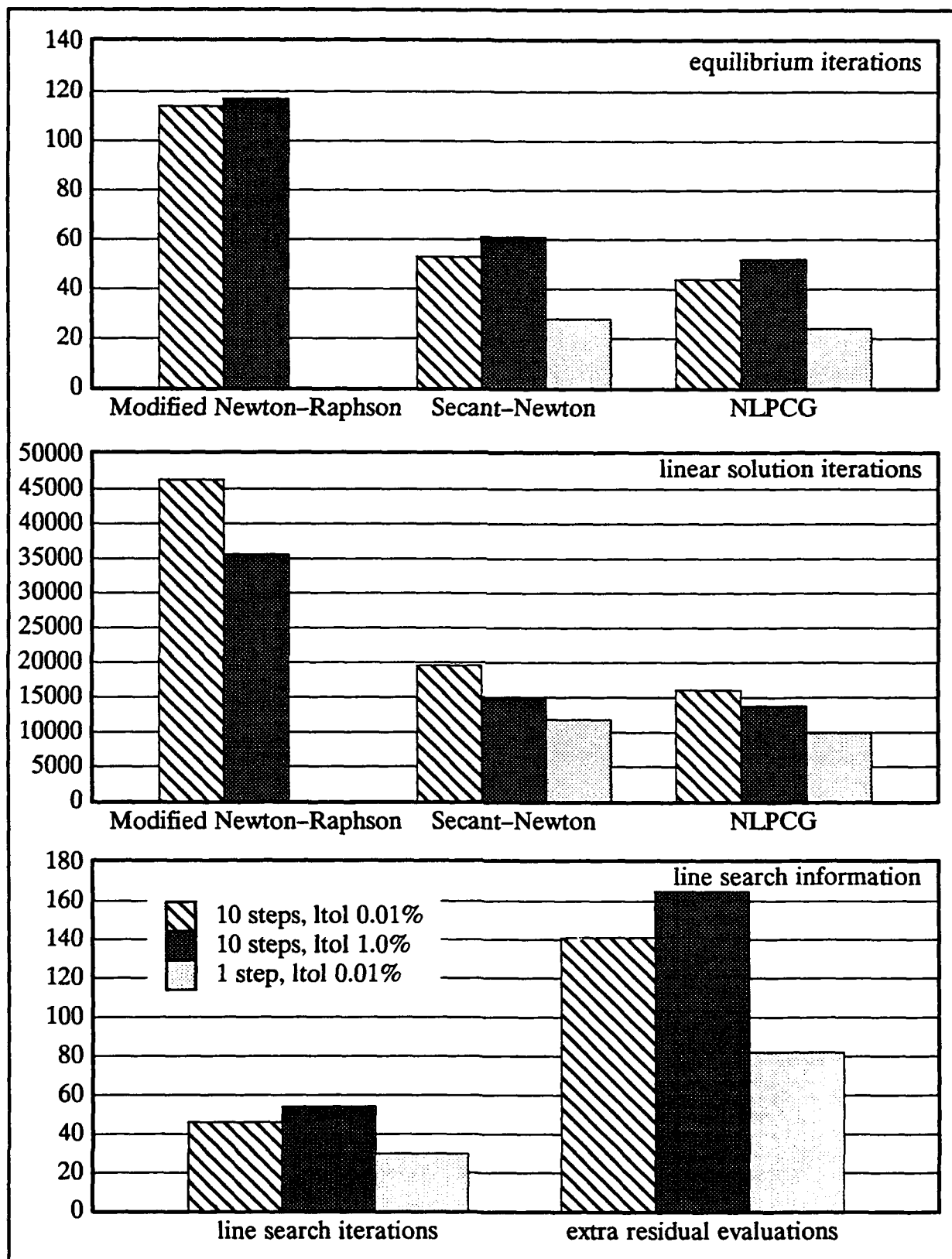


Fig. (4.4-7), 3DL12 timings

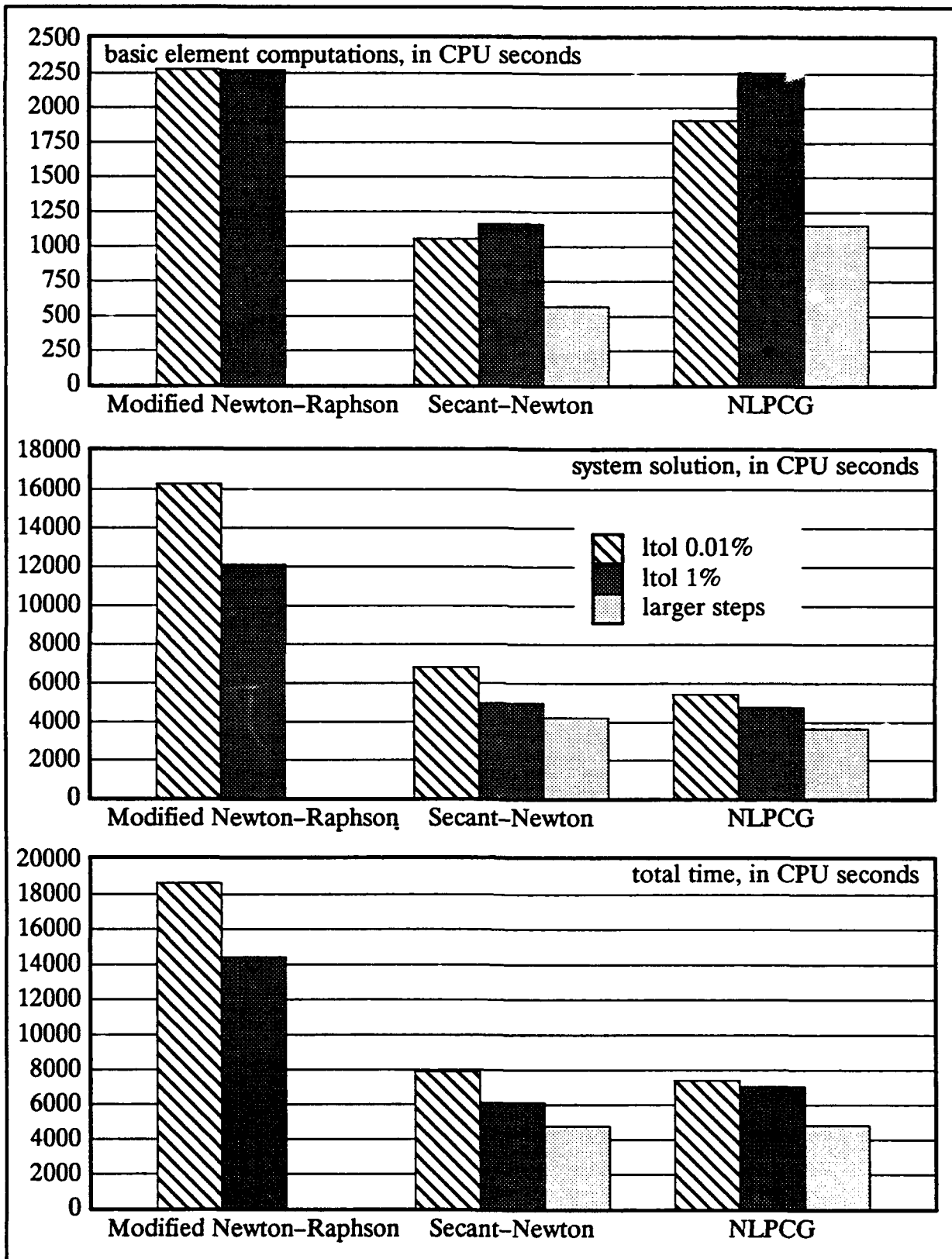


Fig. (4.4-8), 3DL12 timings, cont.

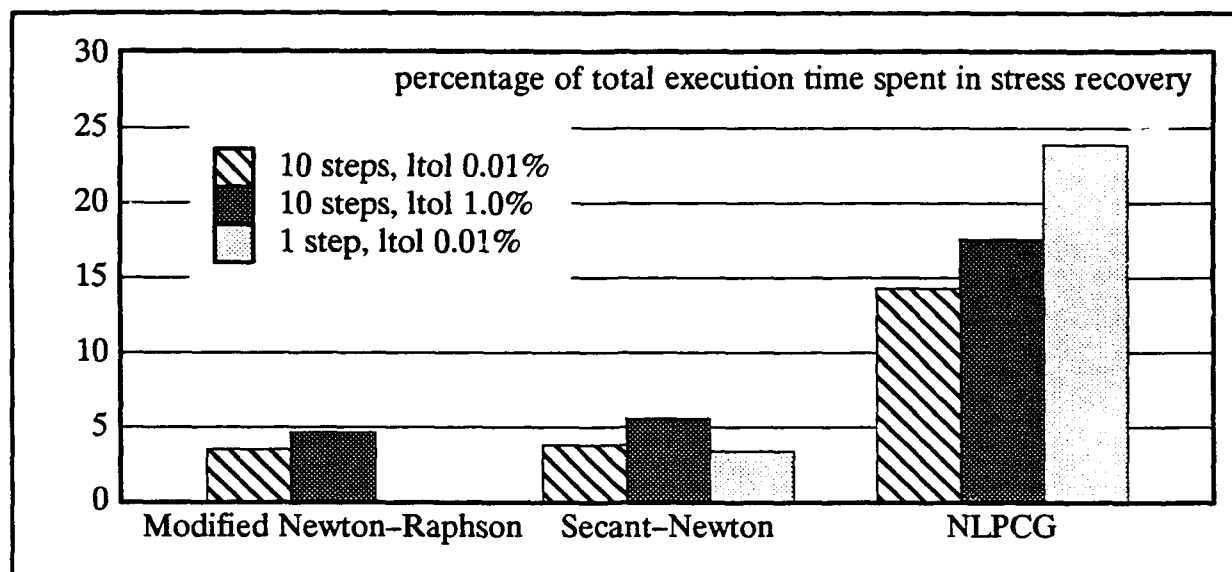


Fig. (4.4-9), 3DL12 timings, cont.

performed linear iterations and thus the system solution costs per equilibrium iteration, which lessens the significance of the NLPCG advantage in equilibrium iterations. This also places a greater emphasis on the NLPCG line search, which because of the reduced accuracy of the 1% tolerance also shows a 17% increase in the number of residual evaluations. The 1% tolerance also degrades the NLPCG and secant-Newton nonlinear convergence rates relative to that of MNR.

Increasing the size of the load steps in Fig. 4.4-7 to 4.4-9 seems to benefit secant-Newton. In contrast to results for the reduced tangent stiffness update strategy of 3DBB, the NLPCG equilibrium iteration advantage over secant-Newton remains in the same proportion as seen for the smaller load steps. In addition, the line search becomes more expensive, requiring 11% more residual evaluations per equilibrium iteration. The consequence is that secant-Newton pulls even with NLPCG in total execution time, as opposed to the slight advantage of NLPCG for the smaller load steps. As increasing the size of the load step size for static 3DL12 does not have the same effect in terms of the LPCG convergence rate as it does for dynamic 3DBB, in the absence of an increased equilibrium advantage the cost of the line search becomes the determining factor in the algorithm comparison.

The amounts of the total execution time attributable to the stress recovery for 3DL12 are similar to those of 3DBB with tangent stiffness updates before each equilibrium iteration, although they are generally somewhat reduced. The larger load step case spends the largest percentage in the stress recovery, indicating the increased importance of the line search. Again, the stress recovery costs for the linear elements of 3DL12 are about 3 times

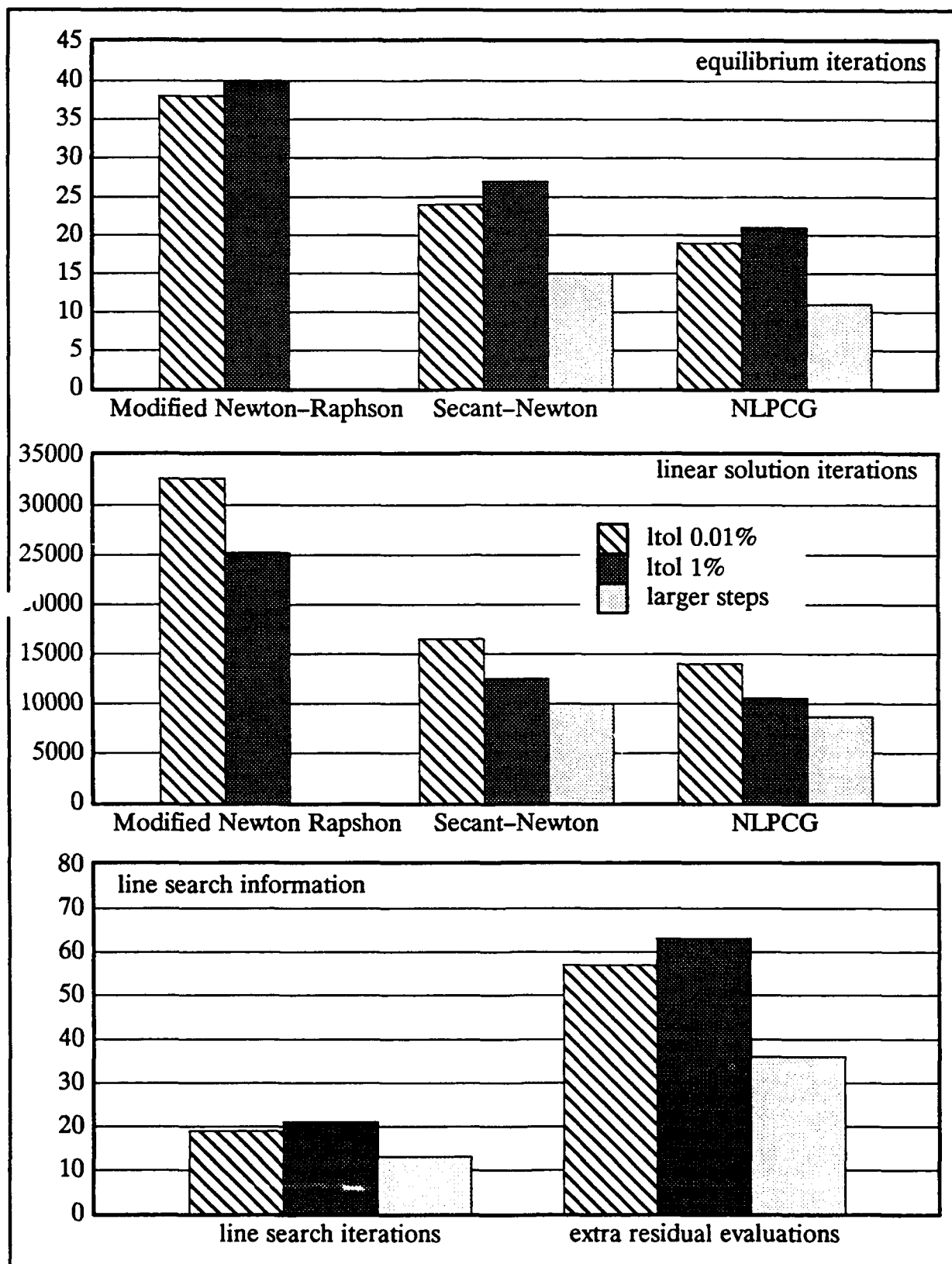


Fig. (4.4-10), 3DQ8 timings

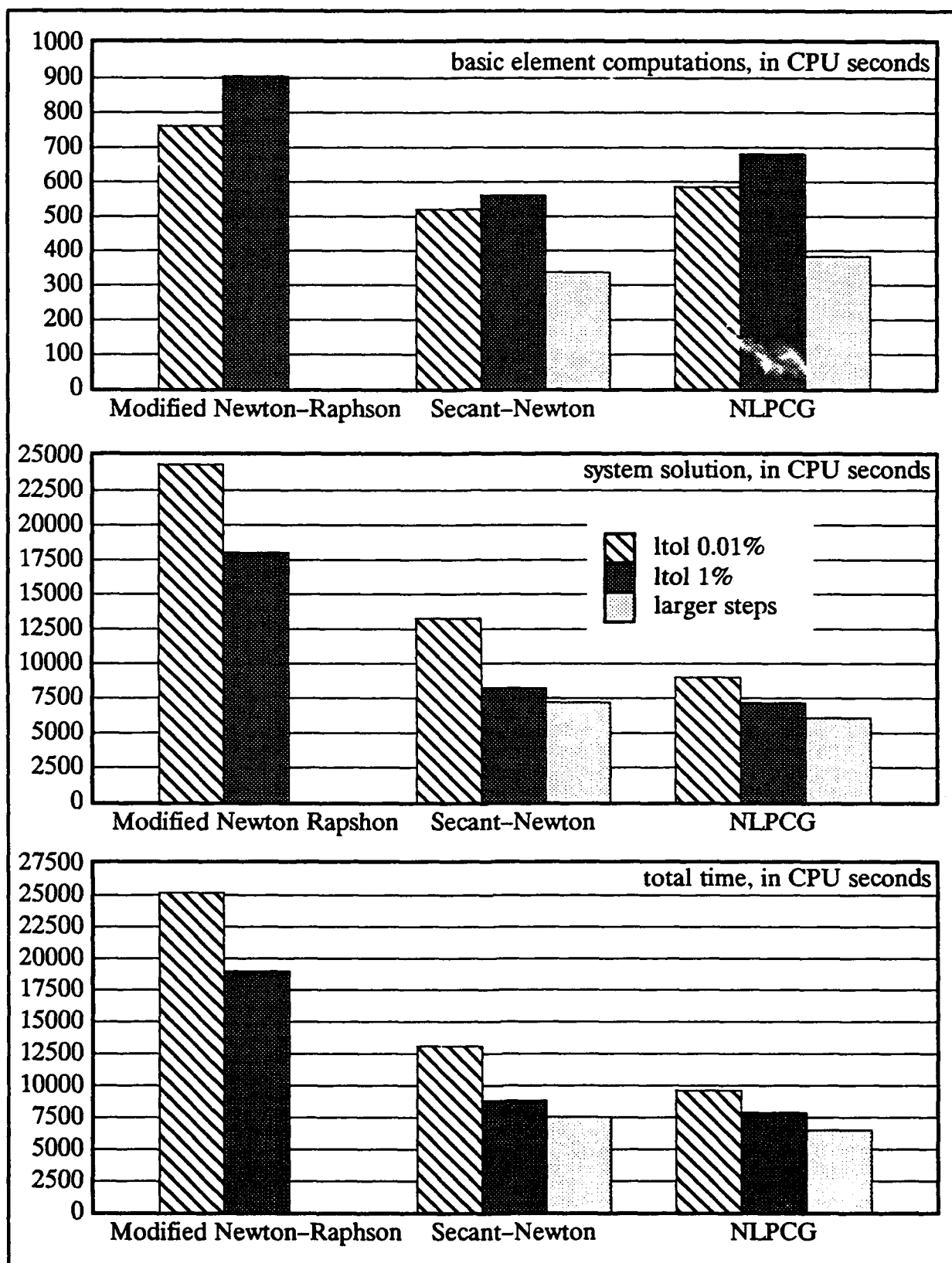


Fig. (4.4-11), 3DQ8 timings, cont.

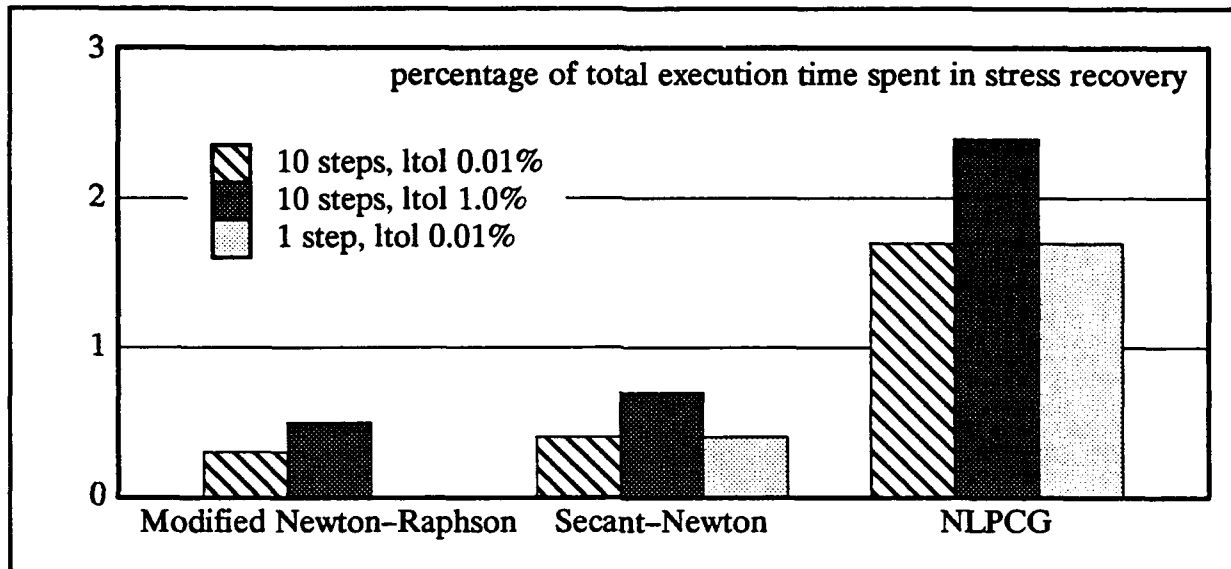


Fig. (4.4-12), 3DQ8 timings, cont.

greater than those of the internal force calculation and thus dominate the line search. Use of the Truesdell formulation again reduces the stress recovery costs by approximately 60%, so that for the NLPCG runs a decrease in the total execution time of as much as 14% could be realized. Again, use of the Alliant FX/8 would mitigate this savings and increase the performance of NLPCG using the unrotated formulation in the stress recovery.

The timing data for the quadratic mesh of 3DQ8 is displayed in Figs. 4.4-7 to 4.4-8. Data is presented for the same cases outlined above for 3DL12. The 0.01% smaller load step data is very similar to that of 3DL12, except that smaller discrepancies in equilibrium iterations lead to the similar advantages in total execution time. This results from the poor LPCG convergence rate for quadratic elements discussed in Section 4.2. Accordingly, many more linear iterations are performed per equilibrium iteration, creating a greater NLPCG linear iteration superiority for a given equilibrium iteration discrepancy. Further, the cost of a linear iteration escalates with the switch to quadratic elements. All told, the NLPCG advantage in system execution time, and thus total execution time, remains at least as great as before for a smaller equilibrium iteration disparity.

As with 3DL12, relaxing the LPCG convergence tolerance decreases the relative speedup of NLPCG with respect to both MNR (2.6 to 2.4) and secant-Newton (1.36 to 1.12) because the system solution cost per equilibrium iteration are reduced. However, for 3DQ8 the cost of the NLPCG line search is insignificant compared to the system solution costs, so that NLPCG remains faster than secant-Newton. Also, all three algorithms show a similar increase in the number of equilibrium iterations so that MNR does not gain as much on

NLPCG and secant-Newton as it did for 3DL12. Considering the runs employing a 1% LPCG convergence tolerance for 3DBB, 3DL12, and 3DQ8, it appears that relaxing the LPCG convergence tolerance tends to favor secant-Newton and MNR over NLPCG.

The consequence of increasing the load step size for 3DQ8 is similar to that for 3DL12, except that again, as discussed below, the cost of the NLPCG line search is not significant, allowing NLPCG to remain faster than secant-Newton.

For the quadratic elements of 3DQ8, the costs of the stress recovery relative to the total execution time are much reduced from 3DBB or 3DL12. In fact, even for NLPCG which requires a line search, the stress recovery is almost nonexistent compared to the total execution time. Further, the expense of the stress recovery is only about 1.6 times that of the internal force calculation. Apparently, the increased amount of computations required by quadratic elements affects the internal force calculation more than the stress recovery, and also increases the system solution time relative to the stress recovery. The poor LPCG convergence rate also contributes to the small percentages of the total execution time exhibited by the stress recovery and the internal force calculation; however, the solution of the boundary value small scale yield problem in Section 3.5 shows that the stress recovery costs comprise less than 1% of the total execution time, and that solution combines quadratic elements and the direct solver.

4.4.3 SUMMARY

3DBB is an example of an analysis dominated by material nonlinearity without undergoing a large scale yielding which would globally activate geometric nonlinearity. Although significant localized geometric nonlinearity develops in the vicinity of the crack tip – with the effective strains there reaching over 30 times the yield strain – the nature of the problem produces moderate but not severe convergence problems for MNR. For this problem, the accelerated nonlinear solution algorithms of secant-Newton and NLPCG in a variety of analyses show very similar nonlinear convergence rates. Because the implementation of secant-Newton is inexpensive, even when its convergence rate is comparable to that of MNR it is competitive. However, the line search required by NLPCG renders it noncompetitive unless it has a faster convergence rate, and even then secant-Newton is at least as fast.

For the highly nonlinear problems of 3DL12 and 3DQ8 where MNR experiences severe convergence problems, NLPCG fares much better. These problems are examples of cases where significant material and geometric nonlinearities combine. This occurs due to the gross yielding of the mesh and the consequent large deformation which introduces considerable geometric effects. For these problems, NLPCG has such an overwhelming advantage

over MNR in its nonlinear convergence rate that the savings in system solution time dominates the cost of the line search and NLPCG is over 2.5 times faster than MNR. The NLPCG convergence rate is also distinctly faster than that of secant-Newton, but for the linear mesh of 3DL12 the line search all but negates this advantage. For the quadratic mesh of 3DQ8, the line search cost is inconsequential, but by relaxing the LPCG convergence tolerance secant-Newton is again very competitive with NLPCG.

Considering Section 3.5, if the 3D finite extension comparisons were carried further into the solution, NLPCG would gain a continually greater advantage over MNR and secant-Newton in the nonlinear convergence rate and thus probably become increasingly dominant in total execution time, regardless of the LPCG convergence tolerance and the line search cost. However, it is possible that if a line search were incorporated into secant-Newton that its nonlinear convergence rate would rival that of NLPCG. In this case, secant-Newton – which is obtained from MNR by a few simple vector operations – or MNR would likely be preferable to NLPCG in all of the comparisons, considering that secant-Newton does not generally require a line search and is generally faster than or approximately equal to NLPCG in total execution time.

The fact that NLPCG requires a line search is a serious drawback, as the line search relies on repeated performance of the stress recovery. Generally, as the material behaviour becomes more complicated, the stress recovery becomes more costly and more difficult to optimize. Such is the case with finite strain plasticity based on the theoretically rigorous unrotated formulation and J_2 flow theory. Although the stress recovery employing Truesdell formulation is less costly, its expense is not inconsiderable. Considering that the plasticity in this thesis employs a constant tangent modulus, generalizing to allow a non-constant modulus can only increase the computational expense. While it may be possible to adopt an approach other than J_2 flow theory so as to minimize the stress recovery cost, for general nonlinear material models the stress recovery will continue to be a line search bottleneck.

5 CONCLUSION

In this chapter, the conclusions concerning the four main focuses of this research are presented. These focuses consist of the comparisons of the competing element computation algorithms, linear solution algorithms, nonlinear solution algorithms, and finite strain plasticity material models, and their suitability to supercomputer applications.

5.1 ELEMENT COMPUTATION ALGORITHMS

The element computation algorithms are the cornerstones of the various versions of NLDPEP. It is important to minimize the cost of the basic element computations so that the majority of the execution time is spent in the nonlinear and linear solution algorithms, and the LPCG element computations are vital to the success of the LPCG solution algorithms. In the remainder of this section, the conclusions pertaining to the two machines used in this research, the Alliant FX/8 and the Convex C240, are presented.

5.1.1 ALLIANT FX/8

In almost every respect, the data of the example problems shows that the best and most efficient element computation algorithm on the Alliant FX/8 is BEC. The performance of BEC is generally excellent, and BEC is particularly effective for the LPCG element computations that form the backbone of the LPCG algorithms. The reasons for this superiority are

- 1) The smaller block granularity of BEC, which facilitates gather/scatter operations and results in less memory occupied by the element block data structure and less virtual memory (swap space) paging.
- 2) The smaller, designed parallelizable granularity of BEC, leading to less overhead and fewer cache misses.
- 3) The generally good incidental vector lengths found in the 3D isoparametric elements – often 24 for linear elements and 60 for quadratic elements – as compared to 32, the length of the Alliant vector registers.
- 4) The consistently good BEC element blocking obtained for all sizes of problems.

Three dimensional isoparametric elements, especially quadratic elements, require a great deal of data and memory to describe and analyze their behaviour. This built-in granularity is not conducive to block vectorization methods because these methods attempt to create a vectorizable granularity on top of the natural granularity of the calculations. BEC works with the natural granularity and exploits it with vectorization while keeping the block

and parallelizable granularity at levels suitable to the architecture of the Alliant FX/8, a parallel-vector machine that emphasizes concurrency, with moderately powerful vector processors and a limited cache and main memory. Consequently, BEC is without question the best element computation algorithm for the Alliant FX/8. It does a good job of minimizing the cost of the basic element computations, and an excellent job of performing the LPCG element computations and allowing LPCG to compete with and in some cases defeat the direct solver for similar numbers of system factorizations and back substitutions.

5.1.2 CONVEX C240

On the Convex C240, BEV is the only implemented element computation algorithm due to the difficulties with the implementation and effective execution of the concurrent algorithms. The performance of BEV in the example problems is respectable; the sequential speedups are reasonable. However, these speedups do not approach the speedups associated with the direct solver operations, primarily the Cholesky factorization. On the Alliant FX/8, the performance of BEC generally exceeds that of the direct solver operations. In addition, the compiler optimized speedups of the LPCG element computations using quadratic elements indicate that BEV is not significantly faster than compiler optimization for these calculations. Some reasons for the disappointing performance of Convex BEV are

- 1) The large block and vectorizable granularity of Convex BEV which creates greater gather/scatter costs – especially for the LPCG element computations where the penalty for gather operations is significant – greater element block data structure memory requirements, and overhead.
- 2) The possible inability of the compiler to generally create efficient vector code – in the sense that the observed vector performance is not commensurate with that seen for benchmark matrix operations – for the vector operations across element blocks required by the element computations.

BEV appears to work fairly well with the block granularity of the linear isoparametric elements, defeating the incidental vectorization of the compiler optimization and rendering reasonable sequential speedups for the pertinent example problems. The performances of both the Convex BEV basic and LPCG element computations are respectable. For quadratic elements, however, the larger natural granularity and consequent larger incidental vectorizable granularity available for compiler optimization – combined with the necessity of gather operations and the fact that this increased natural granularity does nothing but increase the block granularity and overhead of the Convex BEV calculations – cause BEV and the compiler optimization to be nearly equal for the crucial LPCG element computa-

tions. BEV remains superior to compiler optimization for the quadratic basic element computations, and does not perform poorly for the LPCG element computations, simply disappointingly. Modifications should be made to Convex BEV to improve both the performance of the LPCG element computations and the overall performance relative to the direct solver factorization.

5.1.3 GENERAL

On the Alliant FX/8, a significant portion of the BEC speedups is due to the incidental vectorization exploited at the element level. The speedup due solely to concurrency is in the range of 5–7 (on 8 processors) with the remainder due to vectorization (which can be on the order of 3–4). The implication is that BEC using 3D isoparametric elements would not be as effective for massively parallel architectures, which contain very large numbers of sequential processors. It would be necessary to transfer large amounts of data between the processors – generating a great deal of bus traffic and overhead – and vectorization would not exist. BEC using 3D isoparametric elements is most suitable for parallel–vector architectures like the Alliant FX/8.

For simpler elements containing fewer nodes and dof and requiring smaller amounts of data to be manipulated, it is probable that BEC would not be nearly as effective on parallel–vector architectures. The lesser natural granularity of such elements would not afford much of an opportunity for incidental vectorization. BEV or CBEV would likely be much more effective as they would create opportunities for vectorization and the lesser amounts of data would lead to more favorable block and parallelizable granularities and block data structure memory requirements than do the 3D isoparametrics. On the other hand, BEC would be more suitable for massively parallel architectures.

5.2 LINEAR SOLUTION ALGORITHMS

A main focus of this research is the comparison of the LPCG linear solution algorithms, using both ebe and diagonal preconditioners, and a representative direct solver on the machines used in this research, the Alliant FX/8 and the Convex C240. This comparison is broken down by the type of three dimensional isoparametric element employed, and conclusions are presented.

5.2.1 LINEAR ISOPARAMETRIC ELEMENTS

For implicit dynamic applications employing linear isoparametric elements and containing a significant three dimensional mesh topology, LPCG employing its current pre–

conditioners should provide an efficient and economical exploitation of the resources of a supercomputer for a wide variety of applications. This conclusion is reached because

- 1) From the one dimensional mesh of 1DBE to the cubic mesh of 3DL12, the conditioning of the linear dynamic tangent stiffness improves dramatically and is well conditioned even for large time steps sizes. Accordingly, the ebe and diagonal preconditioners provide for nearly constant rapid convergence – as measured by the $\sqrt{\text{ndof}}$ – for the full range of possible time step sizes,
- 2) These properties hold even for the mesh of 3DBB that has a significant three dimensional topology and a flaw that seriously degrades the overall conditioning of the linear dynamic tangent stiffness. Further, only in the extreme case of the 3D finite extension problem does the presence of nonlinearity significantly deteriorate the LPCG convergence rate,
- 3) As the topology of a mesh containing a given number of elements grows more three dimensional, the memory requirements of the direct solver tend to significantly outdistance those of LPCG, to the extent that the analysis of many problems not possible with the direct solver could be readily accomplished using LPCG,
- 4) And, as the mesh topology grows more three dimensional LPCG tends to develop an increasing advantage in execution time over the direct solver for similar numbers of system factorizations and system back substitutions. This advantage exists in the presence of severe nonlinearity and for a wide variety of time step sizes, so that a size can be chosen which allows the implicit dynamics solution to compete very favorably with explicit dynamics.

Both the ebe and diagonal preconditioners perform well for linear elements. Neither preconditioner is generally superior, though the ebe preconditioner seems to be preferable for problems such as 3DBB dominated by material nonlinearity and containing localized geometric nonlinearity. For problems such as 3DL12 combining significant geometric and material nonlinearity, the diagonal preconditioner is a more attractive choice.

5.2.2 QUADRATIC ISOPARAMETRIC ELEMENTS

For a mesh composed of quadratic elements and displaying a significant three dimensional topology without conditions which otherwise cause the dynamic tangent stiffness to be ill-conditioned, the conclusion is much the same as it was for linear meshes. However, for meshes otherwise ill-conditioned the benefits of LPCG can depend on specifying a time step size – within the range desired for the accuracy of the simulation – that is small enough to

maintain the execution time advantage of LPCG yet large enough to permit a favorable comparison between implicit and explicit dynamics. The reasons for this conclusion are

- 1) Moving from the two dimensional mesh of 2DSB to the cubic mesh of 3DQ8 also signals an improvement in the conditioning of the linear dynamic tangent stiffness. However, while the convergence rates attainable with the current preconditioners are nearly constant for a full range of time step sizes, they are slower than for linear meshes,
- 2) For the mesh of 3DCS where the linear dynamic tangent stiffness is ill-conditioned by the mixed bending and membrane behaviour, the convergence rates of both preconditioners are much worse and highly dependent on the time step size,
- 3) The trends concerning the LPCG and direct solver memory requirements observed for linear meshes are accentuated for quadratic elements,
- 4) And, the trend concerning LPCG and direct solver execution times observed for linear meshes are also accentuated for quadratic elements, but for ill-conditioned meshes like 3DCS the LPCG advantage is largely tied to the time step size.

In a choice between the ebe and diagonal preconditioners for quadratic meshes, the use of diagonal preconditioner is recommended. However, as neither preconditioner is truly effective, the search should continue for a more competent preconditioner for quadratic elements.

5.2.3 GENERAL

Since the cost of a solution employing LPCG as the linear solver resides primarily in the system back substitution, then to minimize this cost one would basically want to minimize the number of equilibrium iterations. Within the limits of the cost of a tangent stiffness update, this means updating the tangent stiffness as often as possible. This leads to an interesting observation concerning MNR and LPCG. When using LPCG, the original impetus behind MNR – avoiding direct solver factorization costs by sacrificing the nonlinear convergence rate – is removed. MNR would effectively become Newton-Raphson. In a similar vein, accelerated nonlinear solution algorithms like secant-Newton are quite suitable for LPCG as they are designed to enhance the nonlinear convergence rate.

5.3 NONLINEAR SOLUTION ALGORITHMS

Another focus of this research is the comparison of the nonlinear solution algorithms, using LPCG w/ diag as the linear solver and performed on the Convex C240. Recommendations for selecting a nonlinear solution algorithm are provided.

5.3.1 RECOMMENDATIONS

A version of NLDFEP capable of either MNR or secant-Newton is recommended for general use, with the acceleration provided by secant-Newton being reserved for those analyses with severe convergence problems where MNR is particularly ineffective. This recommendation is made and the use of NLPCG is generally discouraged because

- 1) NLPCG employing a preconditioner other than the full dynamic tangent stiffness converges so slowly on the first time step that its costs are prohibitive.
- 2) NLPCG employing the full dynamic tangent stiffness as the preconditioner is not competitive with either secant-Newton or MNR for moderately nonlinear problems dominated by material nonlinearity. The cost of the line search necessary for NLPCG more than outweighs for any advantage NLPCG may have in the nonlinear convergence rate.
- 3) Even for highly nonlinear problems combining significant geometric and material nonlinearities where NLPCG is decidedly faster than MNR, secant-Newton is competitive with NLPCG.
- 4) The line search required by NLPCG relies heavily on the stress recovery, which for nonlinear analyses may be a costly and complicated calculation resistant to optimization.
- 5) It is a relatively simple matter to obtain secant-Newton from MNR, consisting of a small number of global vector operations well suited to supercomputers.

5.4 MATERIAL MODEL

An important focus of this research has been the search for an effective finite strain plasticity material model for NLDFEP. In the following, recommendations are made concerning the selection of a finite strain plasticity model.

5.4.1 RECOMMENDATIONS

In Section 2.4, the theory and application of the proposed finite strain plasticity models were presented and developed. In Section 3.5, numerical results were presented that

establish the effectiveness of the finite strain plasticity models and compare and contrast the behaviour of the models dependent on the Truesdell and unrotated stress rates. Through these examples, and from the theory supporting the two models presented in Section 2.4, it has been demonstrated that fundamental differences exist between the two formulations. Accordingly, use of the finite strain plasticity model based on the unrotated stress rate is recommended for the following reasons:

- 1) The unrotated formulation is generally more accurate than the Truesdell formulation for larger step sizes. The unrotated formulation is accurate because all required polar decompositions are performed exactly (as opposed to an approximate evolution of the rotation and left or right stretch tensors), while the inaccuracy of the Truesdell formulation results from the use of the state n stresses in the time integration of the Truesdell stress rate.
- 2) The unrotated formulation is more elegant and less complicated to apply. Despite the necessity of polar decompositions, the corresponding extra computation time is generally small compared to the overall solution time. On the Alliant FX/8, the unrotated formulation is only about 20–30% more costly than the Truesdell formulation. This percentage rises significantly on the Convex C240, which simply indicates that the optimization of the polar decomposition on this machine needs improvement.
- 3) The unrotated stress is a measure of true stress so that increments of unrotated stress in the stress recovery process are physically meaningful. Its principal invariants are identical to those of the Cauchy stress, which are therefore stationary for a vanishing unrotated stress rate. The unrotated stress rate accounts for the actual material rotation at a given point and its use implies that the relationship between stress and deformation depends solely upon the material stretching and is independent of the local rotation.
- 4) The Truesdell formulation is deficient in the context of kinematic hardening in that the deviator nature of the back stress is not preserved across the Truesdell rate transformation to the current configuration.

If there is only minor material rotation in a given problem and if isotropic hardening is specified, then the Truesdell formulation is a feasible alternative for those analyses where the added cost of the unrotated formulation is an important concern. Otherwise, the unrotated formulation is generally advocated.

5.5 IMPROVEMENTS AND EXTENSIONS

In this section, possible improvements and extensions to this research are presented and discussed.

5.5.1 LINEAR SOLUTION ALGORITHMS

From consideration of the timing data given in the various example problems, the single calculation most critical to the success of the LPCG algorithms is observed to be the step length calculation. Consequently, any improvement in the efficiency of this calculation would be of great benefit. Currently, the step length calculation of all the element computation algorithms is characterized by certain key features having a direct bearing on its computational efficiency. These are

- 1) Global and local element matrices are stored in an upper triangular vector form within the format of the element data structures of the various element computation algorithms discussed in Section 2.3. By using an upper triangular vector form, the memory requirements of the element matrices are nearly halved.
- 2) Because of the solution requirements of the direct solver and the consistent mass matrix, separate global storage of the element tangent stiffness and mass matrices is provided. However, additional global storage of the unfactorized element dynamic tangent stiffness matrices was considered too costly and is not provided. Consequently, a primary gather operation for the step length calculation is the creation of the local element dynamic stiffness matrices. This gather requires a daxpy operation involving the vector forms of the element tangent stiffness and mass matrices.
- 3) Again to preserve generality, the global element tangent stiffness and tangent mass matrices cannot be constrained, or they would need to be recomputed if only the constraints are changed for a given time step. Thus, the local element dynamic tangent stiffness matrices are constrained as they are created.

The difficulty with the upper triangular vector storage of an element matrix is that it does not lend itself well to the computational efficiency of the matrix-vector product. On the other hand, if a full matrix storage scheme is used, the situation is improved. Testing of the matrix-vector product for BEC on the Alliant FX/8 revealed that while using the upper triangular vector storage, the product produced 14–15 MFLOPS for linear elements and 20–21 MFLOPS for quadratic elements. Using the full matrix storage, these values increased to approximately 21 and 28, respectively. In addition, preliminary testing on the Alliant FX/2800

indicates that full matrix storage is necessary for the efficiency of the matrix-vector product. Apparently, the lack of vector registers in the FX/2800 architecture is responsible for the lack of efficiency of the computations based on the upper triangular vector storage.

Similarly, upon implementation of full matrix storage the Convex BEV matrix-vector product MFLOPS improve from about 11-12 for both linear and quadratic elements to about 15-16 (24 for quadratic elements if the compiler is allowed to interchange loops and vectorize over the number of element degrees of freedom). The most impressive increase occurs for the compiler optimized sequential product on the Convex. The MFLOPS rise from about 4 for linear and 7 for quadratic elements to about 12 and 24. This means that a BEC full matrix implementation on the Convex run in dedicated mode may very well be extremely fast. Unfortunately, all attempts to gather from the global upper triangular vector storage to a local full matrix storage on both machines have been unsatisfactory because the gather - which requires a daxpy operation - is very slow due to indirect addressing and interrupted vector calculations. Thus, one is faced with the computational efficiency of the full matrix storage versus the memory advantage of the upper triangular vector storage.

Both the daxpy operation during the step length gather and the subsequent constraint process - in many cases very costly compared to the matrix-vector product - can be eliminated by replacing global storage for the unconstrained element tangent stiffness and mass matrices with global storage for the constrained element dynamic tangent stiffness matrices. By doing so, the tangent stiffnesses and masses would have to be updated whenever the constraints or time step size is changed as well as whenever a stiffness update is requested. Fortunately, experience with the example problems shows that such conditions would not generally lead to many unnecessary updates of the costly tangent stiffness computation.

Instead of constraining the dynamic tangent stiffness matrix, the linear residual can be constrained after computation during each linear iteration. However, by constraining the dynamic tangent stiffness matrix only when this matrix is updated a considerable savings in execution time can potentially be realized for highly constrained problems. Similarly, the matrix-vector product can be replaced by essentially the recomputation of the element tangent stiffness matrices for each linear iteration. Considering that the presence of geometric non-linearity renders this computation much more costly than the matrix-vector product, it is preferable to sacrifice the savings in memory resulting from the elimination of the element matrices for the computational speed of the matrix-vector product.

Unfortunately, if the consistent mass is specified for use then in order to evaluate the residual the element tangent mass matrices must exist. In this case, either global storage

would be necessary or each element consistent tangent mass matrix would have to be computed every time it was used. However, in all of the example problems solved in this research the lumped mass was used and performed well. If the lumped mass is specified and no global storage for the element mass matrices is provided, then to evaluate the residual only global vector storage for the lumped mass would be required and no recomputation would be necessary (either for the residual evaluation or the situations described above).

Thus it is recommended that the following improvements be implemented:

- 1) Provide global storage of the constrained element dynamic tangent stiffness matrices in lieu of the unconstrained element tangent stiffness and mass matrices.
- 2) Remove the option of employing the consistent mass matrix, leaving only the lumped mass matrix.
- 3) Store element matrices in full matrix form. Combined with 1), full matrix storage would result in less overall memory requirements than currently exist for LPCG versions of NLDFEP using the diagonal preconditioner. For those versions using the ebe preconditioner, the memory requirements attributable to the element matrices would increase by about 33%. If this cost proves too heavy, one could revert to the upper triangular vector storage.

Given the success of the current preconditioners with linear elements, it may be profitable to adopt hierarchical shape functions for quadratic elements. Some attempt should also be made to utilize the previous incremental change of displacements as a starting vector for LPCG.

5.5.2 ELEMENT COMPUTATION ALGORITHMS

One of the biggest bottlenecks to the efficiency of the LPCG element computations is the gather of blocks of nonconflicting element matrices from global to the appropriate local storage. This gather is particularly damaging for Convex BEV, where it can represent up to 40% of the total LPCG element computation time. There appear to be two ways in which this gather operation can be eliminated:

- 1) Renumber the elements in nonconflicting order, so that the global storage of the elements in a block can be operated on directly without sacrificing the properties of unit vector stride or data locality,
- 2) Renumber the nodes for connectivity so that the nodal scatter algorithm discussed

in Section 2.3 can be implemented.

Choosing option 2) would benefit the step length calculation but not the ebe preconditioning and factorization while option 1) would benefit all LPCG element computations. Thus, element renumbering in nonconflicting order is recommended to eliminate the costs of gathering element quantities to local data structures. The ramifications of this action are the necessity of constant rather than unit strides for some vector calculations and the loss of local data structures and possibly data locality.

Recommendations 1) and 2) for the linear solution algorithms and element renumbering were implemented on both the Alliant FX/8 and the Convex C240. The actual element renumbering was performed in the preprocessor for NLDFEP which translates a PATRAN-II neutral file into a NLDFEP input file. These improvements eliminated the gather of element matrices for the LPCG element computations and the creation of the constrained element dynamic stiffness matrices on the fly. The resultant improvement in the computational speed of LPCG is dramatic. Fig. 5.5-1 displays the revised relative speedups of the various linear solvers previously seen in Fig. 4.2-8. LPCG with either preconditioner in Fig 5.5-1 exhibits a speedup of at least 1.5 over the corresponding timings in Fig. 4.2-8, with the speedup sometimes exceeding 2. LPCG w/ diag generally benefits more than LPCG w/ ebe because the improvement in speed of the matrix-vector product imbedded in the step length calculation benefits the most from the improvements. The result of these speedups is that LPCG becomes more attractive relative to the direct solver, and LPCG w/ diag clearly becomes the preferable preconditioner for nearly every example problem.

As a further illustration of the beneficial effects of eliminating the gather of element matrices, Figs. 5.5-2 and 5.5-3 display the revised speedups associated with the ebe preconditioning step with respect to both the sequential and compiler optimized applications. The ebe preconditioning is chosen because it was not necessary to rerun the sequential timings as the sequential ebe preconditioning code is unchanged. While BEC is still the fastest algorithm on the Alliant, BEV and CBEV both improve drastically over the previous speedups seen in Figs. 4.3-13 and 4.3-14. Convex BEV also shows the same level of improvement, and with sequential speedups on the order of 8 it rivals the speedups associated with the highly optimized vendor supplied LINPACK direct solver. It seems that eliminating the gather of element matrices goes a long way towards improving the performance of Convex BEV to a more satisfactory level.

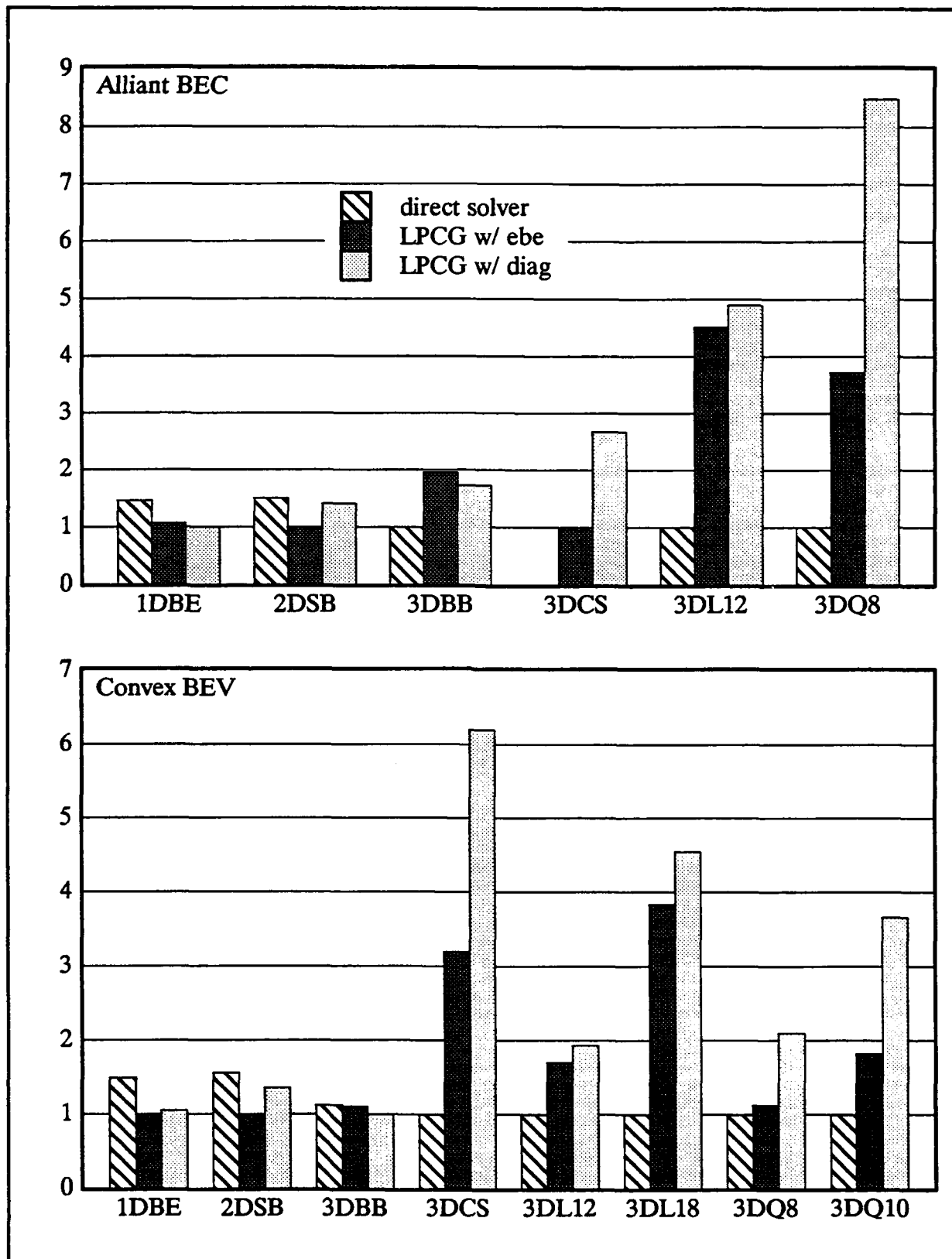


Fig. (5.5-1), relative speedups

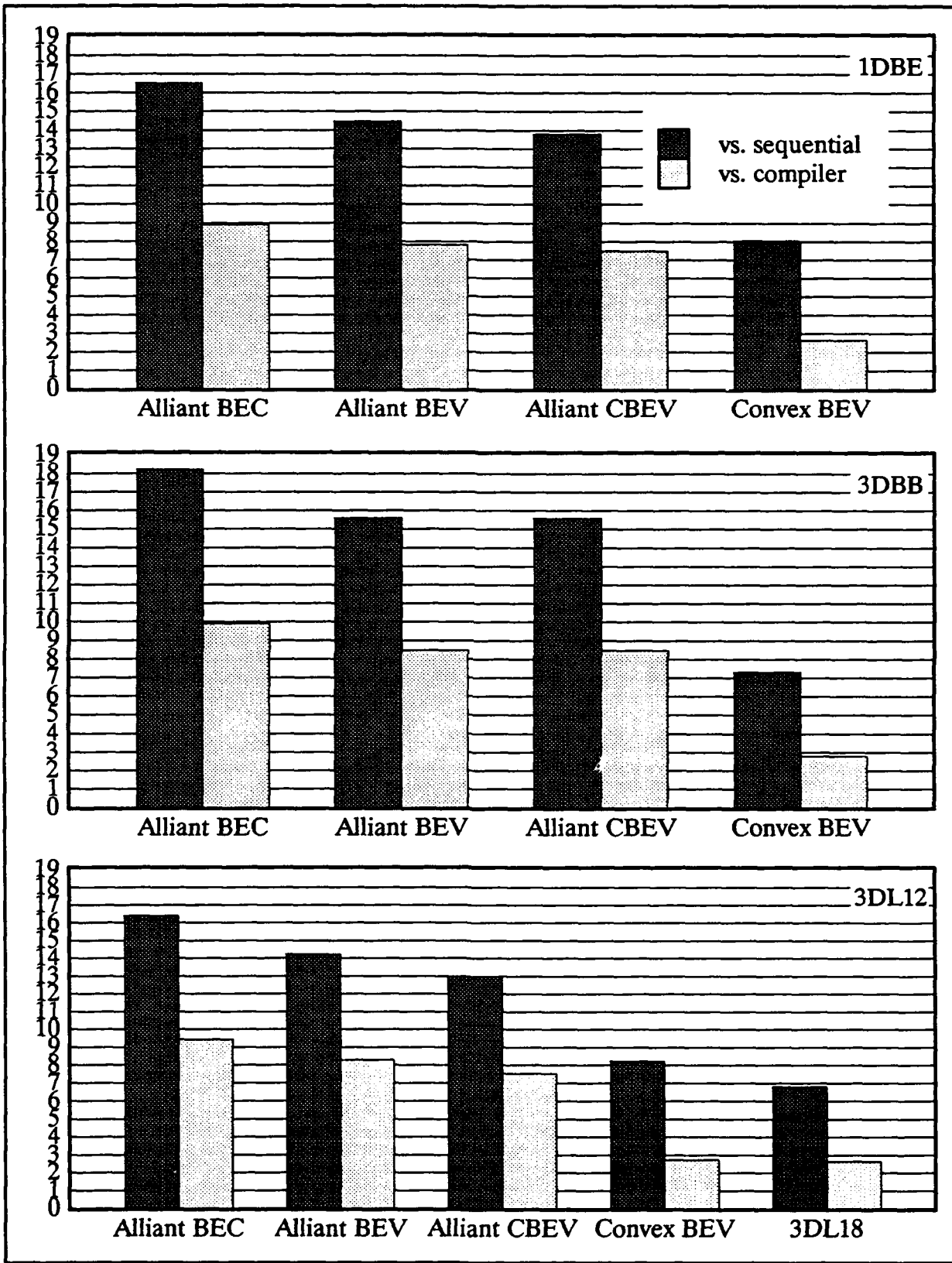


Fig. (5.5-2), speedups for ebe preconditioning, problems with linear elements

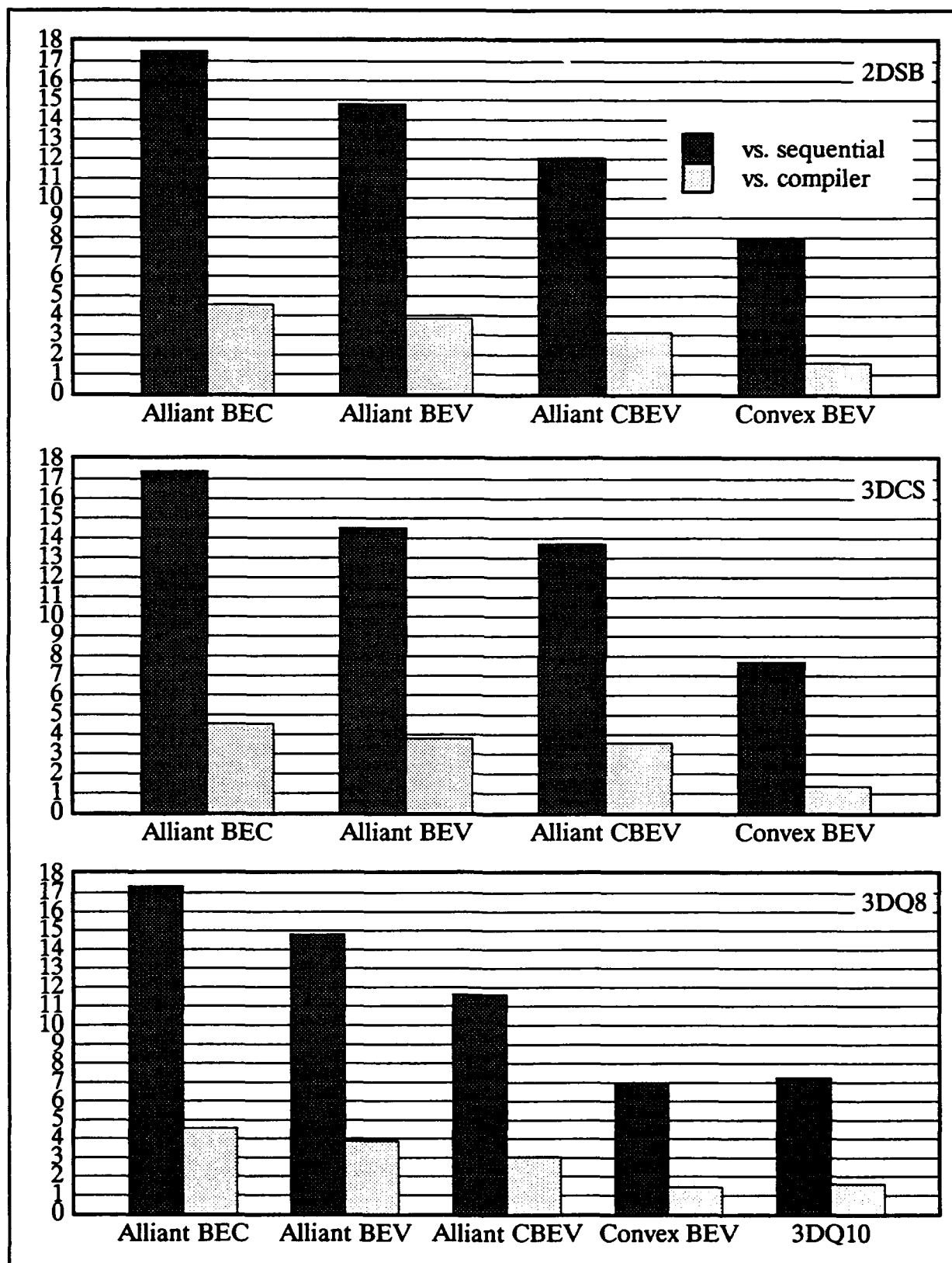


Fig. (5.5-3), speedups for ebe preconditioning, problems with quadratic elements

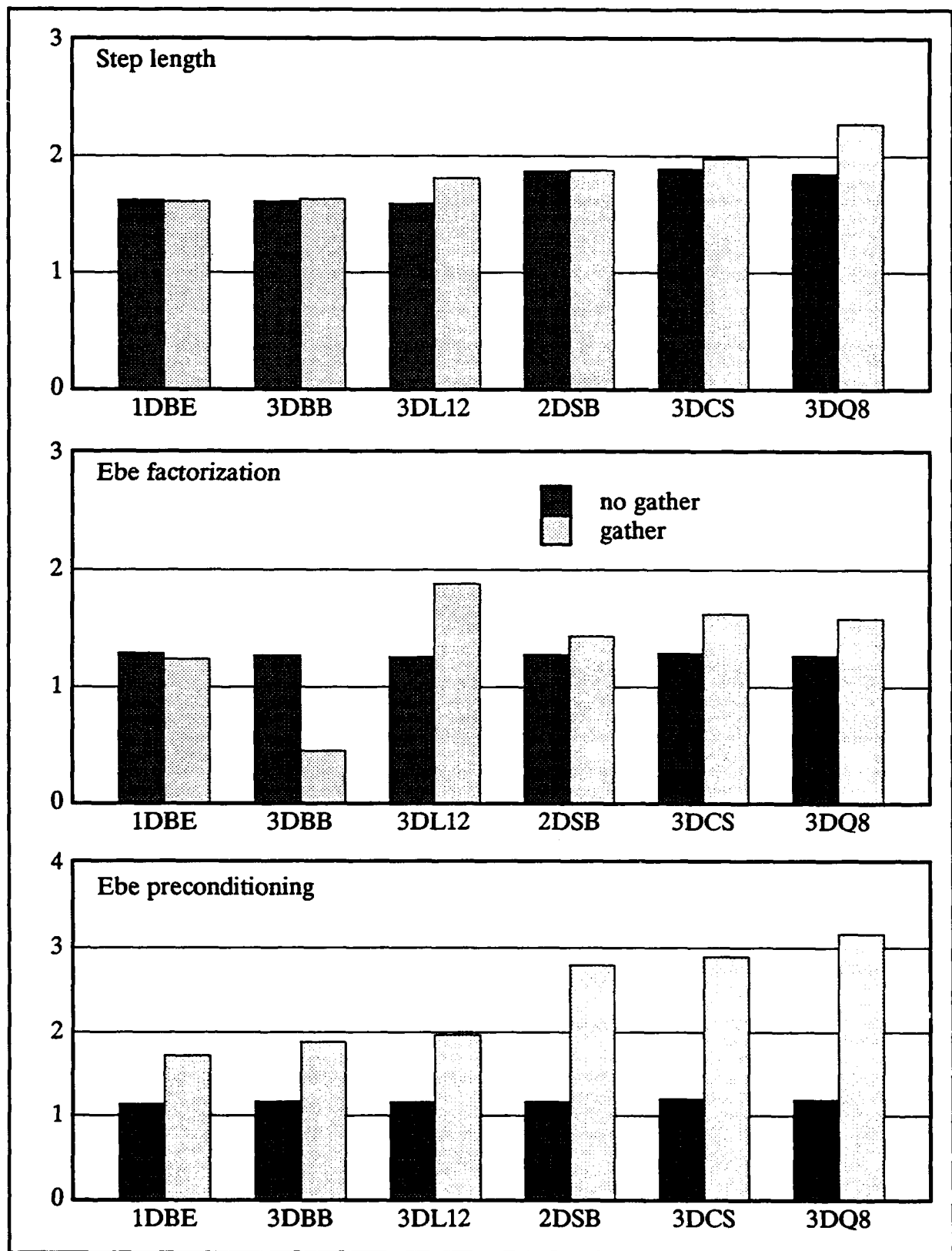


Fig. (5.5-4), relative speedup of Alliant BEC vs. Alliant BEV

Because, before application of the improvements, BEC was already avoiding the gather of element matrices for the ebe factorization and preconditioning by directly passing the address of the appropriate element matrix, it shows the least increase in speed of all of the element computation algorithms for LPCG w/ ebe. The increase in performance of BEV with respect to BEC on the Alliant FX/8 is shown in Fig. 5.5-4. For the step length calculation, all algorithms create the constrained dynamic stiffness matrices on the fly and the relative performances remain fairly constant. As expected, the ebe preconditioning shows the greatest gains for BEV. Still, BEC remains the fastest element computation algorithm on the Alliant, especially for LPCG w/ diag which relies most heavily on the matrix-vector product. It simply is not as superior as it was before.

Given that BEV is the only element computation algorithm implemented on the Convex C240, the possibilities for concurrency using the four vector processors on the Convex have been ignored. In the future, Convex versions of both CBEV and BEC should be implemented. CBEV would be an extension of BEV, with designed concurrency applied on top of block vectorization. BEC would be an extension of the compiler optimized sequential application, with designed concurrency applied on top of the incidental vectorization of the compiler. From the Convex results for BEV and compiler optimization (and considering that large enough problems could be analyzed on the Convex to allow sufficient CBEV element blocking), there are indications that CBEV may very well be the most effective algorithm for linear isoparametric elements while BEC may be preferable for quadratic elements. Although there would have to be a very light machine load – or the runs would have to be made in dedicated or single user mode – to make concurrency a viable option on the Convex, these opportunities for concurrency should be exploited.

For the Alliant implementation of BEV, blocks of 32 elements were created and COVI mode execution was arranged. Given the memory requirements of the direct solver, on the Alliant FX/8 the example problems using quadratic elements contained too few elements to create full blocks (witness CBEV) if vector-concurrent inner loops of 256 elements were to be employed. By abandoning the direct solver and using LPCG exclusively, much larger problems could be analyzed and full blocks of 256 elements could be created. In this case, it may be preferable to arrange for vector-concurrent inner loops for Alliant BEV rather than the COVI loops currently in use.

5.5.3 NONLINEAR SOLUTION ALGORITHMS

The use of NLPCG with a preconditioner other than the full dynamic tangent stiffness was abandoned because the convergence was so poor on the first time step that it was

not competitive. This use of NLPCG could be resurrected if the first time step is assumed linear and, in effect, LPCG was used. Once past the first time step, using the change in displacements for the previous step as a starting point for nonlinear iterations could greatly enhance convergence. In addition, the line search could be replaced by essentially a secant stiffness calculation [4].

An optional line search could be added to the version of NLDfEP capable of either MNR or secant-Newton for those extreme cases where secant-Newton has a deficient nonlinear convergence rate.

5.5.4 MATERIAL MODEL

As mentioned in Section 2.4, the current implementation of the unrotated formulation in NLDfEP does one too many polar decompositions per stress recovery. By eliminating this extra computation, the speed of the unrotated formulation relative to the Truesdell formulation would be significantly enhanced. This modification should be made.

BIBLIOGRAPHY

- [1] ABAQUS User's Manual, Version 4.8, Hibbitt, Karlsson & Sorenson, Inc., Providence RI, 1990.
- [2] Argyris J., "An Excursion Into Large Rotations," *Computer Methods in Applied Mechanics and Engineering*, Vol. 32, 1982, pp. 85-155
- [3] Atluri S.N., "On Constitutive Relations at Finite Strain: Hypo-elasticity and Elastoplasticity with Isotropic and Kinematic Hardening," *Computer Methods in Applied Mechanics and Engineering*, Vol. 43, 1984, pp. 137-171
- [4] Biffle J.H., "Indirect Solution of Static Problems Using Concurrent Vector Processing Computers," *Parallel Computations and their Impact on Mechanics*, ed. by A.K. Noor, American Society of Mechanical Engineers, New York, 1987, pp. 317-330
- [5] Boland W.R., Kamgnia E.R., and Kowalik J.S., "A Conjugate Gradient Optimization Method Invariant to Nonlinear Scaling," *J. Optim. Theory Appl.*, Vol. 27, No. 2, February 1979, pp. 221-230
- [6] Broyden C.G., "The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations," *J. Inst. Maths Applies*, Vol. 6, 1970, pp. 76-90
- [7] Broyden C.G., "The Convergence of a Class of Double-rank Minimization Algorithms 2. The New Algorithm," *J. Inst. Maths Applies*, Vol. 6, 1970, pp. 222-231
- [8] Buckley A.G., "A Combined Conjugate Gradient Quasi-Newton Minimization Algorithm," *Mathematical Programming*, Vol. 15, 1978, pp. 200-210
- [9] Carey G.F., "Parallelism in Finite Element Modelling," *Communications in Applied Numerical Methods*, Vol. 2, No. 3, May-June 1986, pp. 281-287
- [10] Carey G.F., and Barragy E., "Finite Element Analysis Using Advanced Processors," *Proceedings of the Fall Joint Computer Conference*, University of Texas, November 1986, pp. 535-539
- [11] Carey G.F., and Jiang B., "Element-By-Element Linear and Nonlinear Solution Schemes," *Communications in Applied Numerical Methods*, Vol. 2, No. 2, March-April 1986, pp. 145-153
- [12] Carlson D.E., and Hoger A., "On the Derivatives of the Principal Invariants of a Second-Order Tensor," *Journal of Elasticity*, Vol. 16, 1986, pp. 221-224

- [13] Cervera M., Liu Y.C., and Hinton E., "Preconditioned Conjugate Gradient Method for the Nonlinear Finite Element Analysis with Particular Reference to 3D Reinforced Concrete Structures," *Engineering Computations*, Vol. 3, September 1986, pp. 235-242
- [14] Chin H., "The SAS Domain Decomposition Method for Structural Analysis," Doctoral Thesis, Computer Science, University of Illinois at Urbana-Champaign, 1988
- [15] Concus P., Golub G.H., and O'Leary D.P., "A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equation," *Sparse Matrix Computations*, ed. J.R. Bunch and D.J. Rose, Academic Press, New York, 1965, pp. 307-322
- [16] Cook R. D., "Concepts and Applications of Finite Element Analysis," Second edition, John Wiley & Sons, Inc., New York NY, 1981
- [17] Crisfield M.A., "Incremental / Iterative Solution Procedures for Non-Linear Structural Analysis," *Numerical Methods for Non-Linear Problems*, Ed by C. Taylor et al, Pineridge, Swansea, 1980, pp. 261-290
- [18] Crisfield M.A., "Some Recent Research on Numerical Techniques for Structural Analysis," *Proceedings of the NUMETA '85 Conference*, Swansea, 1985, pp. 565-575
- [19] Crisfield M.A., "A Faster Modified Newton-Raphson Iteration," *Computer Methods in Applied Mechanics and Engineering*, Vol. 20, 1979, pp. 267-278
- [20] Dienes J.K., "On the Analysis of Rotation And Stress Rate In Deforming Bodies," *Acta Mechanica*, Vol. 32, 1979, pp. 217-232
- [21] Dodds R.H., "Numerical Techniques for Plasticity Computations in Finite Element Analysis," *Computers and Structures*, Vol. 26, 1987, pp. 767-779
- [22] Dongarra J.J., "Unrolling Loops in FORTRAN," *Software - Practice and Experience*, Vol. 9, 1979, pp. 219-226
- [23] Dongarra J., Bunch J.R., Moler C.B., and Stewart G.W., "LINPACK User's Guide," SIAM Publications, Philadelphia, 1978
- [24] Dongarra J.J., and Eisenstat S.C., "Squeezing the Most Out of an Algorithm in CRAY FORTRAN," *ACM Transactions on Mechanical Software*, Vol. 10, No. 3, Sept. 1984, pp. 219-230
- [25] Farhat C., Wilson E., and Powell G., "Solution of Finite Element Systems on Concurrent Processing Computers," *Engineering with Computers*, Vol. 2, 1987, pp. 157-165

- [26] Flanagan D.P., and Taylor L.M., "An Accurate Numerical Algorithm for Stress Integration with Finite Rotations," *Computer Methods in Applied Mechanics and Engineering*, Vol. 62, 1987, pp. 305-320
- [27] Flanagan D.P., and Taylor L.M., "Structuring Data for Concurrent Vectorized Processing in a Transient Dynamics Finite Element Program," *Parallel Computations and their Impact on Mechanics*, ed. by A.K. Noor, American Society of Mechanical Engineers, New York, 1987, pp. 291-299
- [28] Fletcher R., "A New Approach to Variable Metric Algorithms," *The Computer Journal*, Vol. 13, August 1970, pp. 317-322
- [29] Fletcher R., and Powell M.J.D., "A Rapidly Convergent Descent Method for Minimization," *The Computer Journal*, Vol. 6, 1963, pp. 163-168
- [30] Fletcher R., and Reeves C.M., "Function Minimization by Conjugate Gradients," *The Computer Journal*, Vol. 6, 1964, pp. 149-154
- [31] Geradin M., Idelsohn S., and Hogge M., "Nonlinear Structural Dynamics via Newton and Quasi-Newton Methods," *Nuclear Engineering and Design*, Vol. 58, 1980, pp. 339-348
- [32] Gill P.E., and Murray W., "Quasi-Newton Methods for Unconstrained Optimization," *J. Inst. Maths Applics*, Vol. 9, 1972, pp. 91-108
- [33] Goel R.P., and Malvern L.E., "Biaxial Plastic Simple Waves with Combined Kinematic and Isotropic Hardening," *Journal of Applied Mechanics*, December 1970, pp. 1100-1106
- [34] Golub G.H., and Van Loan C.F., "Matrix Computations," The Johns Hopkins University Press, Baltimore Maryland, 1983
- [35] Green A.E., and Naghdi P.M., "A General Theory of an Elastic-plastic ccontinuum," *Arch. Rat. Mech. Anal.*, Vol. 18, No. 1965, pp. 251
- [36] Havner, K. S., "The Theory of Finite Plastic Deformation of Crystalline Solids," pp. 265-302
- [37] Hoger A., and Carlson D.E., "Determination of the Stretch and Rotation in the Polar Decomposition of the Deformation Gradient," *Quarterly of Applied Mathematics*, Vol. 42, 1984, pp. 113-117

- [38] Hughes, T.J.R., "Stability, Convergence, and Growth and Decay of Energy of the Average Acceleration Method in Nonlinear Structural Dynamics," *Computers and Structures*, Vol. 6, 1976, pp. 313-324
- [39] 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," *Computer Methods in Applied Mechanics and Engineering*, Vol. 61, 1987, pp. 215-248
- [40] Hughes T.J.R., and Winget J., "Finite Rotation Effects in Numerical Integration of Rate Constitutive Equations Arising in Large Deformation Analysis," *International Journal for Numerical Methods in Engineering*, Vol. 15, 1980, pp. 1862-1867
- [41] Irons B.M., "Quadrature Rules for Brick Based Finite Elements," *International Journal for Numerical Methods in Engineering*, Vol. 3, No. 2, 1971, pp. 293-294
- [42] Irons B., and Elsawaf A., "The Conjugate Newton Algorithm for Solving Finite Element Equations," *Proceedings of the U.S. - German Symp. on Formulations and Algorithms in Finite Element Analysis*, Ed by Bathe, Oden, and Wunderlich, MIT, 1977, pp. 656-672
- [43] Jalby, W., and Meier, U., "Optimizing Matrix Operations on a Parallel Multiprocessor with a Hierarchical Memory System," 1986, pp. 429-432
- [44] Johnson G.C., and Bammann D.J., "A Discussion of Stress Rates in Finite Deformation Problems," *International Journal of Solids and Structures*, Vol. 20, 1984, pp. 725-737
- [45] Joyce, J. A., "Ductile to Brittle Toughness Transition Characterization of A533B Steel," U.S.Nuclear Regulation Commission NUREG/CR-4818, February, 1987
- [46] Kamath C., Sameh A.H., Yang G.C., Kuck D.J., "Structural Computations on The Cedar System," *Computers and Structures*, Vol. 20, No. 1-3, 1985, pp. 47-54
- [47] Key S.W., "Spectrom-331, A Finite Element Computer Program for the Large Deformation, Elastic and Inelastic, Transient Dynamic Response of Three Dimensional Solids and Structures," Topical Report RSI-0299, RE/SPEC Inc, P.O.Box 14984, Albuquerque, NM 87191, 1988
- [48] Key S.W., and Krieg R. D., "On the Numerical Implementation of Inelastic Time Independent and Time Dependent, Finite Strain Constitutive Equations in Structural Mechanics," *Computer Methods in Applied Mechanics and Engineering*, Vol. 33, 1982, pp. 439-452

- [49] Klessig R., and Polak E., "Efficient Implementations of the Polak–Ribiere Conjugate Gradient Algorithm," *SIAM Journal Control Optim.*, Vol. 10, 1972, pp. 524–549
- [50] Kolsky H., "Stress Waves in Solids," Dover Publications Inc., New York NY, 1963
- [51] Krieg R.D., and Krieg D.B., "Accuracies of Numerical Solution Methods for the Elastic–Perfectly Plastic Model," *Journal of Pressure Vessel Technology*, November 1977, pp. 510–515
- [52] Krieg R.D., and Key S.W., "Implementation of a Time Independent Plasticity Theory Into Structural Computer Programs," *Constitutive Equations in Viscoplasticity: Computational And Engineering Aspects*, Ed. by J.A. Stricklin and K.J. Saczalski, AMD Vol. 20 (ASME, New York, 1985), pp. 125–138
- [53] Mathies H., and Strang G., "The Solution of Nonlinear Finite Element Equations," *International Journal for Numerical Methods in Engineering*, Vol. 14, 1979, pp. 1613–1626
- [54] Mendelson A., "Plasticity: Theory and Application," Robert E. Krieger Publishing Company, Inc., Malabar, FL, 1968
- [55] Meirovitch L., "Elements of Vibration Analysis," McGraw–Hill, Inc., New York, NY, 1975
- [56] Nagtegaal J.C., and de Jong J.E., "Some Computational Aspects of Elastic–Plastic, Large Strain Analysis," *International Journal for Numerical Methods in Engineering*, Vol. 12, 1981, pp. 15–41
- [57] Nakamura T., Shih C.F., and Freund L.B., "Three–Dimensional Transient Analysis of a Dynamically Loaded Three–Point–Bend Ductile Fracture Specimen," Office of Naval Research ONR0365/3, September, 1986
- [58] Nazareth L., "A Conjugate Direction Algorithm Without Line Searches," *J. Optim. Theory Appl.*, Vol. 23, No. 3, November 1977, pp. 373–387
- [59] Newmark N.M., "A Method of Computation for Structural Dynamics," *Journal of the Engineering Mechanics Division*, ASCE, Vol. 32, No. EM3, 1959, pp. 67–94
- [60] Noor A.K., "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

- [61] O'Dowd N.P., and Shih C.F., "Family of Crack Tip Fields Characterized by a Triaxiality Parameter: Part I – Structure of Fields," *Journal of the Mechanics and Physics of Solids*, to appear later
- [62] Oren S.S., "Self-Scaling Metric Algorithms Without Line Search for Unconstrained Minimization," *Mathematics of Computation*, Vol. 27, No. 124, October 1973, pp. 873–885
- [63] Ortega J.M., "Introduction to Parallel and Vector Solution of Linear Systems," Plenum Press, New York NY, 1988
- [64] Papadrakakis M., and Ghionis P., "Conjugate Gradient Algorithms in Nonlinear Structural Analysis Problems," *Computer Methods in Applied Mechanics and Engineering*, Vol. 59, 1986, pp. 11–27
- [65] PATRAN-II Modeling Software, PDA Engineering, 1560 Brookhollow Dr., Santa Ana, California, 92705
- [66] Pegon P., Halleux J.P., and Donea J., "A Discussion of Stress Rates and their Application in Finite Strain Plasticity," *Proceedings of the NUMETA '85 Conference*, Swansea, 1985, pp. 315–325
- [67] Perry A., "A Self Correcting Conjugate Gradient Algorithm," *Internat. J. Comput. Math.*, Vol. 6, 1978, pp. 327–333
- [68] Pinsky, P.M., Ortiz M., and Pister K.S., "Numerical Integration of Rate Constitutive Equations in Finite Deformation Analysis," *Computer Methods in Applied Mechanics and Engineering*, Vol. 40, 1983, pp. 137–158
- [69] Polak E., and Ribiere G., "Note sur la Convergence de Methodes de Directions Conjugees," *Revue Francaise Inf. Rech. Oper.*, Vol. 16, RI 1964, pp. 35–43
- [70] POLO-FINITE User's Manual, Civil Engineering Systems Laboratory, University of Illinois, Urbana-Champaign, IL
- [71] Powell M.J.D., "Restart Procedures for the Conjugate Gradient Method," *Mathematical Programming*, Vol. 12, 1977, pp. 241–254
- [72] Rankin C.C., and Brogan F.A., "An Element Independent Corotational Procedure for the Treatment of Large Rotations," *Collapse Analysis of Structures*, ASME, PVP Vol. 84, 1984, pp. 85–100

- [73] Sarigul N., Maitan J., and Kamel H.A., "Solution of Nonlinear Structural Problems Using Array Processors," *Computer Methods in Applied Mechanics and Engineering*, Vol. 34, 1982, pp. 939-954
- [74] Sawyers, K., " ",
- [75] Schoeberle, D.F., and Belytschko, T., "On the Unconditional Stability of an Implicit Algorithm for Nonlinear Structural Dynamics," *Journal of Applied Mechanics*, Vol. 42, 1975, pp. 865-869
- [76] Schreyer H.L., Kulak R.F., and Kramer J.M., "Accurate Numerical Solutions for Elastic-Plastic Models," *Journal of Pressure Vessel Technology*, August 1979, Vol. 101, pp. 226-234
- [77] Shanno D.F., "Conjugate Gradient Methods with Inexact Line Searches," *Mathematics of Operations Research*, Vol. 3, No. 3, August 1978, pp. 244-256
- [78] Simo J.C., and Taylor R.C., "Consistent Tangent Operators for Rate-Independent Elastoplasticity," *Computer Methods in Applied Mechanics and Engineering*, Vol. 48, 1985, pp. 101-118
- [79] Sorenson H.W., "Comparison of Some Conjugate Direction Procedures for Function Minimization," *Journal of the Franklin Institute*, Vol. 288, No. 6, December 1969, pp. 421-441
- [80] Taylor L.M., and Flanagan D.P., "PRONTO 2D, a Two-dimensional Transient Solid Dynamics Program," SAND86-0594, Sandia National Laboratories, Albuquerque, NM, 1987
- [81] Taylor L.M., and Flanagan D.P., "PRONTO 3D, a Three-dimensional Transient Solid Dynamics Program," SAND87-1912, Sandia National Laboratories, Albuquerque, NM, 1988
- [82] Thomas, G.B., "Calculus and Analytic Geometry," Addison-Wesley Publishing Company, Inc., Philippines, Alternate Edition, 1972
- [83] Ting T.C.T., "Determination of $C^{\frac{1}{2}}$, $C^{-\frac{1}{2}}$ and More General Isotropic Tensor Functions of C ," *Journal of Elasticity*, Vol. 15, 1985, pp. 319-323
- [84] Yaghmai S., and Popov E.P., "Incremental Analysis of Large Deflections of Shells of Revolution," *International Journal of Solids and Structures*, Vol. 7, 1971, pp. 1375-1393

- [85] Zienkiewicz, O.C., Irons, B. M., Scott, F.C., and Cambell, J., "Three Dimensional Stress Analysis," *Proc. IUTAM Symp. on High Speed Computing of Elastic Structures*, University of Liege Press, pp. 413-433
- [86] Zienkiewicz, O.C., and Nayak, G., "A General Approach to Problems of Plasticity and Large Deformation Using Isoparametric Elements," *Conference on Matrix Methods in Structural Mechanics*, Wright-Patterson Air Force Base, No. 3, 1971, pp. 881-928

APPENDIX A

Appendix A contains information regarding various matrix forms and transformation matrices referred to in the main body of the thesis.

UPPER TRIANGULAR VECTOR FORM FOR SYMMETRIC MATRIX STORAGE

The upper triangular vector form for the storage of a symmetric matrix is illustrated in Fig. A-1. Basically, the upper triangle of the symmetric matrix is stored by columns.

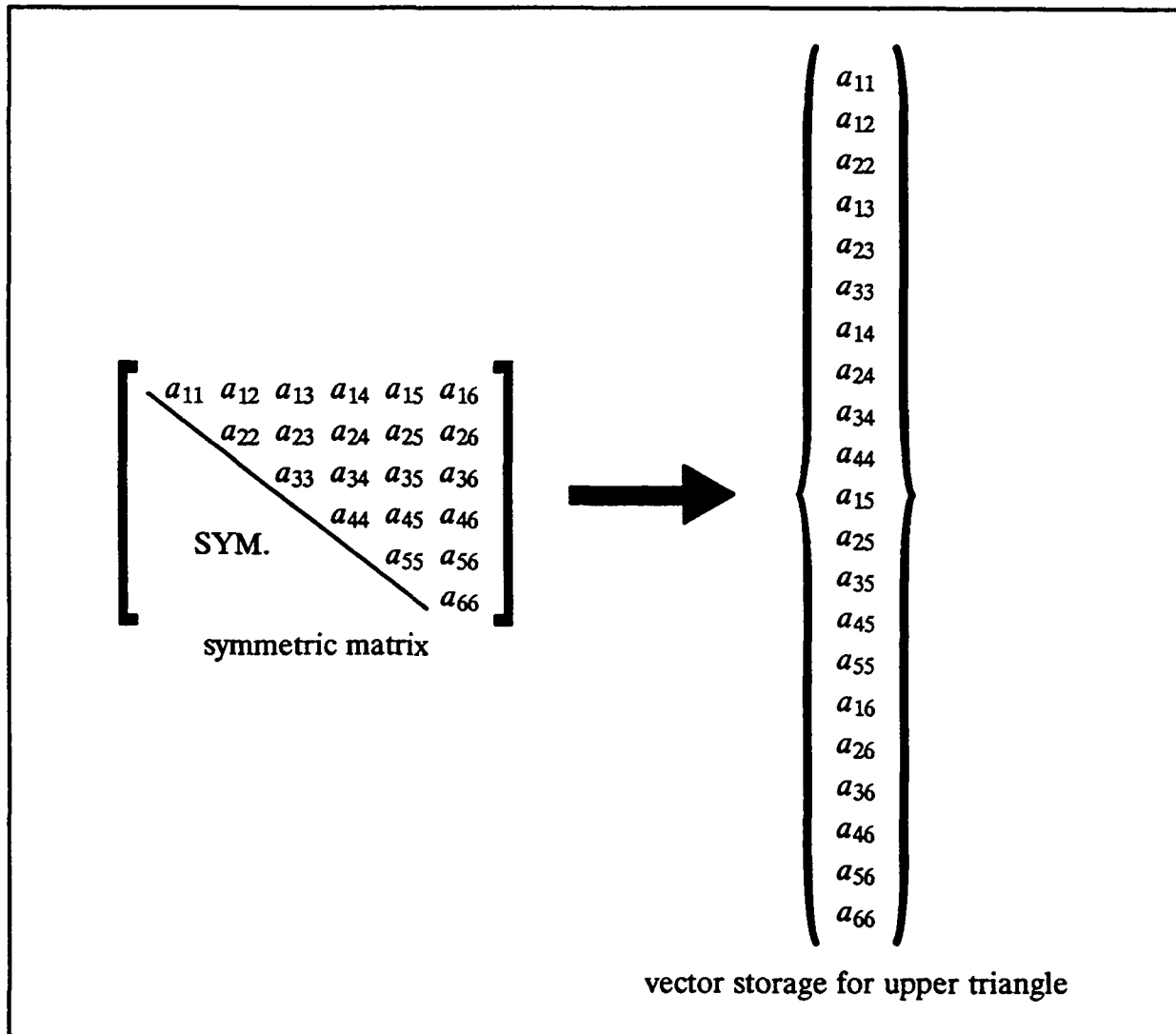


Fig. (A-1), upper triangular vector form for symmetric matrix storage

HIERARCHICAL TRANSFORMATION MATRIX

Equations (55) and (56) of Section 2.2 define the transformation of the serendipity element degrees of freedom to the hierarchical form and the reverse transformation, respec-

tively. The composition of the transformation matrix driving these equations is presented here. This transformation matrix depends on the connectivity of the quadratic element nodes and on the arrangement of the element degrees of freedom, which are illustrated in Fig. A-2. Note that the element corner nodes precede the midside nodes for the u,v, and w degrees of freedom.

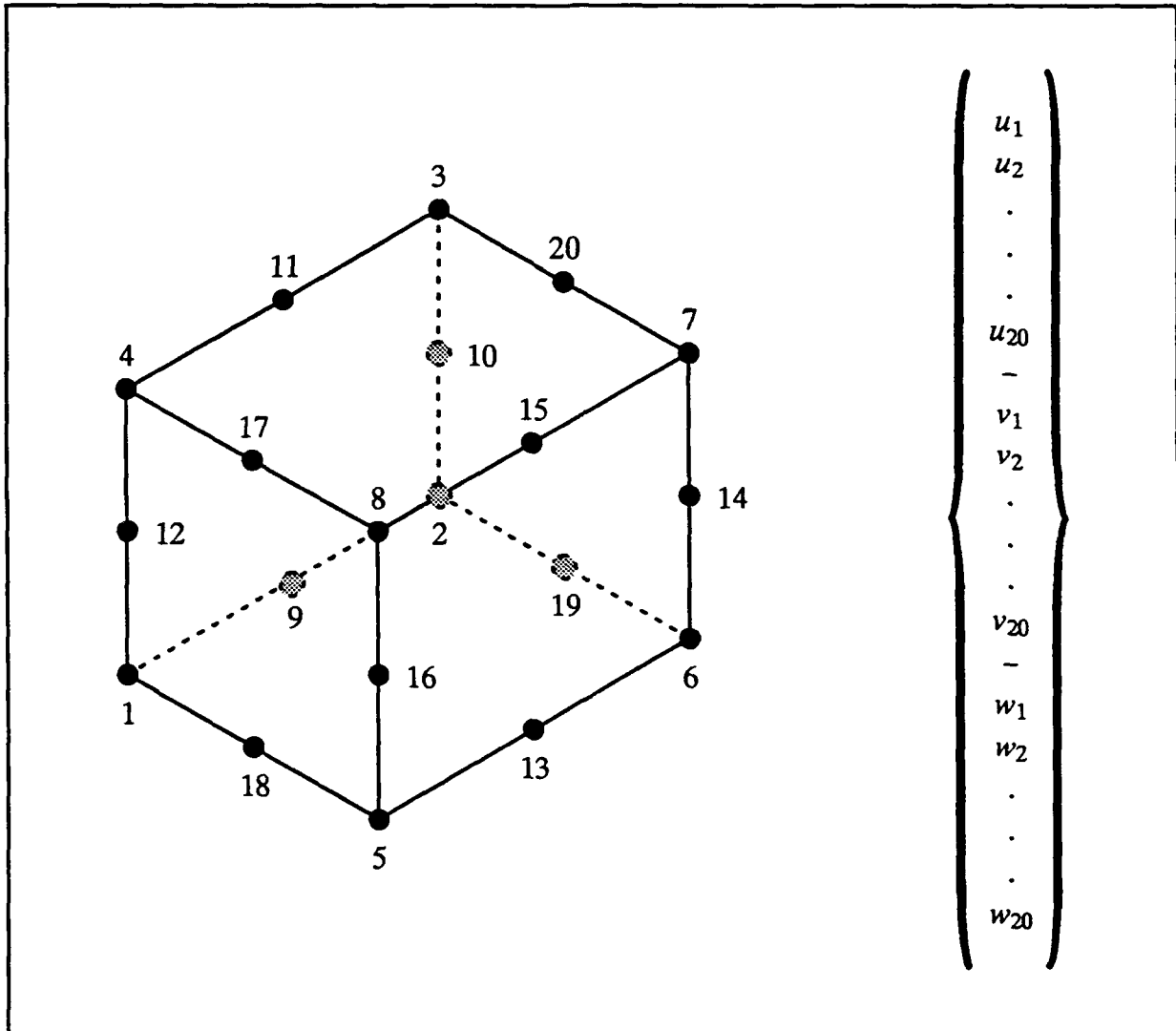


Fig. (A-2), quadratic element connectivity and ordering of element dof

The u,v, and w degrees of freedom transform independently of each other, resulting in the following form for the transformation matrix:

$$T^h = \begin{bmatrix} t^h & 0 & 0 \\ 0 & t^h & 0 \\ 0 & 0 & t^h \end{bmatrix} \quad (A-1)$$

The nodal submatrix is described by

$$t^h = \left[\begin{array}{c|c} I_{8 \times 8} & 0_{12 \times 8} \\ \hline t_{21}^h & -I_{12 \times 12} \end{array} \right] \quad (A-2)$$

where the lower left submatrix of the nodal submatrix is defined as

$$t_{21}^h = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 \\ 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 \end{bmatrix} \quad (A-3)$$

In practice, the full form of the transformation matrix is not used, with the matrix multiplication performed in discrete scalar multiplies to take advantage of the sparse nature of the transformation matrix.

STRESS AND STRAIN TRANSFORMATION MATRICES

The matrices governing the transformation of unrotated Cauchy stress to and from second Piola–Kirchoff stress, Cauchy stress to and from second Piola–Kirchoff stress, the unrotated rate of deformation strain measure to the rate of deformation strain measure, and the Cauchy stress to reflect the velocity gradient terms in the Truesdell stress rate are presented in the following.

UNROTATED CAUCHY STRESS AND SECOND PK STRESS

The transformation of a second Piola–Kirchoff stress vector to an unrotated Cauchy stress vector is represented by equation (100) of Section 2.4. The transformation matrix contained in this equation is defined as

$$[T_U] = \frac{1}{J} \begin{bmatrix} U_1^2 & U_2^2 & U_4^2 & 2U_1U_2 & 2U_2U_4 & 2U_1U_4 \\ U_2^2 & U_3^2 & U_5^2 & 2U_2U_3 & 2U_3U_5 & 2U_2U_5 \\ U_4^2 & U_5^2 & U_6^2 & 2U_4U_5 & 2U_5U_6 & 2U_4U_6 \\ U_1U_2 & U_2U_3 & U_4U_5 & U_1U_3 + U_2^2 & U_3U_4 + U_2U_5 & U_1U_5 + U_2U_4 \\ U_2U_4 & U_3U_5 & U_5U_6 & U_2U_5 + U_3U_4 & U_3U_6 + U_5^2 & U_2U_6 + U_4U_5 \\ U_1U_4 & U_2U_5 & U_4U_6 & U_1U_5 + U_2U_4 & U_2U_6 + U_4U_5 & U_1U_6 + U_4^2 \end{bmatrix} \quad (A-4)$$

The U terms in equation (4) refer to the terms of the right stretch tensor arranged in upper triangular vector form. J refers to the determinate of the deformation gradient. The transformation matrix of the reverse transformation of equation (103) in Section 2.4 is identical to equation (4) except that J is multiplied, not divided, and the right stretch terms are replaced by the inverse of the right stretch tensor arranged in upper triangular vector form.

CAUCHY STRESS AND SECOND PK STRESS

The transformation of a second Piola–Kirchoff stress vector to a Cauchy stress vector is represented by equation (111) of Section 2.4. The transformation matrix contained in this equation is defined as

$$[T_T] = \frac{1}{J} \begin{bmatrix} F_{11}^2 & F_{12}^2 & F_{13}^2 & 2F_{11}F_{12} & 2F_{12}F_{13} & 2F_{11}F_{13} \\ F_{21}^2 & F_{22}^2 & F_{23}^2 & 2F_{21}F_{22} & 2F_{22}F_{23} & 2F_{21}F_{23} \\ F_{31}^2 & F_{32}^2 & F_{33}^2 & 2F_{31}F_{32} & 2F_{32}F_{33} & 2F_{31}F_{33} \\ F_{11}F_{21} & F_{12}F_{22} & F_{13}F_{23} & F_{11}F_{22} + F_{12}F_{21} & F_{12}F_{23} + F_{13}F_{22} & F_{11}F_{23} + F_{13}F_{21} \\ F_{21}F_{31} & F_{22}F_{32} & F_{23}F_{33} & F_{21}F_{32} + F_{22}F_{31} & F_{22}F_{33} + F_{23}F_{32} & F_{21}F_{33} + F_{23}F_{31} \\ F_{11}F_{31} & F_{12}F_{32} & F_{13}F_{33} & F_{11}F_{32} + F_{12}F_{31} & F_{12}F_{33} + F_{13}F_{32} & F_{11}F_{33} + F_{13}F_{31} \end{bmatrix} \quad (A-5)$$

The F terms in equation (5) refer to the terms of the deformation gradient. J again refers to the determinate of the deformation gradient. The transformation matrix of the reverse transformation of equation (116) in Section 2.4 is identical to equation (5) except that J is again multiplied, not divided, and the deformation gradient terms are replaced by the inverse of the deformation gradient.

UNROTATED RATE OF DEFORMATION AND RATE OF DEFORMATION

The transformation of the vector form of a one point integrated rate of deformation tensor over a time step to the similarly integrated vector form of an unrotated rate of deformation tensor, represented by equation (102) of Section 2.4, is driven by the transformation matrix defined as

$$[Q_U] = \begin{bmatrix} R_{11}^2 & R_{21}^2 & R_{31}^2 & R_{11}R_{21} & R_{21}R_{31} & R_{11}R_{31} \\ R_{12}^2 & R_{22}^2 & R_{32}^2 & R_{12}R_{22} & R_{22}R_{32} & R_{12}R_{32} \\ R_{13}^2 & R_{23}^2 & R_{33}^2 & R_{13}R_{23} & R_{23}R_{33} & R_{13}R_{33} \\ R_{11}R_{12} & R_{21}R_{22} & R_{31}R_{32} & R_{11}R_{22} + R_{12}R_{21} & R_{21}R_{32} + R_{31}R_{22} & R_{11}R_{32} + R_{12}R_{31} \\ R_{12}R_{13} & R_{22}R_{23} & R_{32}R_{33} & R_{12}R_{23} + R_{22}R_{13} & R_{22}R_{33} + R_{23}R_{32} & R_{12}R_{33} + R_{13}R_{32} \\ R_{11}R_{13} & R_{21}R_{23} & R_{31}R_{33} & R_{11}R_{23} + R_{21}R_{13} & R_{21}R_{33} + R_{31}R_{23} & R_{11}R_{33} + R_{13}R_{31} \end{bmatrix} \quad (A-6)$$

The R terms of equation (6) refer to the terms of the rotation tensor.

VELOCITY GRADIENT TRANSFORMATION

The transformation of a Cauchy stress vector to reflect the velocity gradient terms of the Truesdell stress rate is described by equation (112) of Section 2.4. The matrix governing this equation is defined as

$$[Q_T] = \begin{bmatrix} 2\Delta V_{11} & 0 & 0 & 2\Delta V_{12} & 0 & 2\Delta V_{13} \\ 0 & 2\Delta V_{22} & 0 & 2\Delta V_{21} & 2\Delta V_{23} & 0 \\ 0 & 0 & 2\Delta V_{33} & 0 & 2\Delta V_{32} & 2\Delta V_{31} \\ \Delta V_{21} & \Delta V_{12} & 0 & \Delta V_{11} + \Delta V_{22} & \Delta V_{13} & \Delta V_{23} \\ 0 & \Delta V_{32} & \Delta V_{23} & \Delta V_{31} & \Delta V_{22} + \Delta V_{33} & \Delta V_{21} \\ \Delta V_{31} & 0 & \Delta V_{13} & \Delta V_{32} & \Delta V_{12} & \Delta V_{11} + \Delta V_{22} \end{bmatrix} - \Delta D_{kk}[I] \quad (A-7)$$

The ΔV terms in equation (7) refer to the terms of the one point integrated velocity gradient over the time step and ΔD_{kk} is the trace of the one point integrated rate of deformation tensor.

APPENDIX B

Appendix B contains all the FORTRAN code included in the thesis. This code encompasses the element blocking schemes, the element common areas of all considered element computation algorithms, the internal force vector calculation for all considered algorithms, the step length calculation for all implemented algorithms, the ebe factorization for all implemented algorithms, and the ebe preconditioning for all implemented algorithms.

ELEMENT BLOCKING

Included in the FORTRAN code for the element blocking schemes are, in order, the segment of code calling the element grouping (for similarity) and element blocking subroutines, which is valid for all element computation algorithms, the element grouping and blocking subroutines, again valid for all element computation algorithms, and the segment of code subdividing the element blocks into groups that can be processed concurrently, as required by CBEV.


```

      ESSPNS(0,BLK)= NOMP
      END IF
C
      AUGBLK= 0
      LSTEL= 1
      ESSPNS(1,BLK)= 1
C
      DO 170 I= 2,ESSPNS(0,BLK)
      IF(AUGBLK.LT.NUMDR) THEN
        LSTEL= LSTEL+1
        AUGBLK= AUGBLK+1
      END IF
      LSTEL= LSTEL+BLRPM
      ESSPNS(I,BLK)= LSTEL
      CONTINUE
170
C
      ESSPNS(ESSPNS(0,BLK)+1,BLK)= ELBLK(0,BLK)+1
C
160 CONTINUE

```

7.2.2 ELEMENT COMMON AREAS

The element common areas included here belong to, in order, BEC, BEV, CBEV, BEC/MPV, BEV/MPC, and BMPC. The BEV element common area applies to both the Alliant FX/8 and Convex C240 implementations.

REAL*8 CE,UE,DUE,UEH1,UEH2
 REAL*8 PKEN1,BKEN1,RTSE,PPE,DDTSE,DDTSE,DDTSE
 REAL*8 ELEIFV,TRNTE,EMAT,ELESTR,EK(MXVL,MXUTS2),MEL(MXVL,MXUTS2)
 LOGICAL TRNE
 DIMENSION ESTATE(MXVL,MXGP)
 EQUIVALENCE (EOSTAT,ESTATE), (EMAT,EK), (EMAT,MEL)

.....

Element common area for CBEV:

.....

COMMON/DDSPV/ CE(MXVL,MXECOR,MXNP),UE(MXVL,MXEDOF,MXNP),
 & DUE(MXVL,MXEDOF,MXNP)

COMMON/DSTRM/ PPE(MXVL,MXGP,MXNP),RTSE(MXVL,MXSTR,MXGP,MXNP),
 & PKEN(MXVL,MXSTR,MXGP,MXNP),BKEN1(MXVL,MXSTR,MXGP,MXNP),
 & BKEN(MXVL,MXSTR,MXGP,MXNP),DDTSE(MXVL,MXSTR,MXGP,MXNP),
 & DDTSE(MXVL,MXSTR,MXGP,MXNP),DDTSE(MXVL,MXSTR,MXGP,MXNP),
 & USTRSE(MXVL,MXSTR,MXGP,MXNP)

COMMON/DSOLA/ ELEIFV(MXVL,MXEDOF,MXNP),EMAT(MXVL,MXUTS2,MXNP),
 & ELESTR(MXVL,MXOUPL,MXGP,MXNP),TRNTE(MXVL,MXEDOF,MXNP)

COMMON/IGENV/ TRNE(MXVL,MXDEL,MXNP)

COMMON/IGENV/ EOSTAT(MXVL,MXGP,MXNP),ENSTAT(MXVL,MXGP,MXNP),
 & ELINCD(MXVL,MXDEL,MXNP),BELDST(MXVL,MXEDOF,MXNP)

REAL*8 CE,UE,DUE,UEH1(MXVL,MXEDOF,MXNP)
 REAL*8 PKEN1,BKEN1,RTSE,PPE,DDTSE,DDTSE,DDTSE
 REAL*8 ELEIFV,TRNTE,EMAT,ELESTR,EK(MXVL,MXUTS2,MXNP),
 & MEL(MXVL,MXUTS2,MXNP)

LOGICAL TRNE
 DIMENSION ESTATE(MXVL,MXGP,MXNP)
 EQUIVALENCE (EOSTAT,ESTATE), (UE,UEH1), (EMAT,EK), (EMAT,MEL)

.....

Element common area for DEC/MPV:

.....

COMMON/DDSPV/ CE(MXECOR,MXNP),CG(MXGP,MXECOR,MXNP),
 & UE(MXEDOF,MXNP),UG(MXGP,MXEDOF,MXNP),DUE(MXEDOF,MXNP),
 & DUG(MXGP,MXEDOF,MXNP)

COMMON/DSTRM/ PPE(MXGP,MXNP),RTSE(MXGP,MXSTR,MXNP),
 & PKEN(MXGP,MXSTR,MXNP),BKEN1(MXGP,MXSTR,MXNP),BKEN(MXGP,MXSTR,MXNP),

.....

 Element common area for BEC:

COMMON/DDSPV/ CE(MXECOR,MXNP),UE(MXVL,MXEDOF,MXNP),DUE(MXEDOF,MXNP)

COMMON/DSTRM/ PPE(MXGP,MXNP),RTSE(MXSTR,MXGP,MXNP),
 & PKEN(MXSTR,MXGP,MXNP),BKEN1(MXSTR,MXGP,MXNP),BKEN(MXSTR,MXGP,MXNP),
 & BKEN(MXSTR,MXGP,MXNP),DDTSE(MXSTR,MXGP,MXNP),
 & UDDTSE(MXSTR,MXGP,MXNP),USTRSE(MXSTR,MXGP,MXNP)

COMMON/DSOLA/ ELEIFV(MXEDOF,MXNP),EMAT(MXUTS2,MXNP),
 & TRNTE(MXEDOF,MXNP),ELESTR(MXOUPL,MXGP,MXNP)

COMMON/IGENV/ TRNE(MXDEL,MXNP)

COMMON/IGENV/ EOSTAT(MXGP,MXNP),ENSTAT(MXGP,MXNP),
 & ELINCD(MXDEL,MXNP),BELDST(MXEDOF,MXNP)

REAL*8 CE,UE,DUE,UEH1(MXEDOF,MXNP)
 REAL*8 PKEN1,BKEN1,RTSE,PPE,DDTSE,DDTSE,DDTSE
 REAL*8 ELEIFV,TRNTE,EMAT,ELESTR,EK(MXUTS2,MXNP),MEL(MXUTS2,MXNP)

LOGICAL TRNE

DIMENSION ESTATE(MXGP,MXNP)
 EQUIVALENCE (EOSTAT,ESTATE), (UE,UEH1), (EMAT,EK), (EMAT,MEL)

.....

Element common area for BEV:

.....

COMMON/DDSPV/ CE(MXVL,MXECOR),UE(MXVL,MXEDOF),DUE(MXVL,MXEDOF),
 & UEN1(MXVL,MXEDOF),UEN2(MXVL,MXEDOF)

COMMON/DSTRM/ PPE(MXVL,MXSTR,MXGP),BKEN1(MXVL,MXSTR,MXGP),
 & BKEN(MXVL,MXSTR,MXGP),BKEN1(MXVL,MXSTR,MXGP),RTSE(MXVL,MXSTR,MXGP),
 & PPE(MXVL,MXSTR,MXGP),DDTSE(MXVL,MXSTR,MXGP),DDTSE(MXVL,MXSTR,MXGP),
 & USTRSE(MXVL,MXSTR,MXGP)

COMMON/DSOLA/ ELEIFV(MXVL,MXEDOF),TRNTE(MXVL,MXEDOF,MXNP),
 & EMAT(MXVL,MXUTS2),ELESTR(MXVL,MXOUPL,MXGP)

COMMON/IGENV/ TRNE(MXVL,MXDEL)

COMMON/IGENV/ EOSTAT(MXVL,MXGP),ENSTAT(MXVL,MXGP),
 & ELINCD(MXVL,MXDEL),BELDST(MXVL,MXEDOF)

```

C      6 BREN1(MGCP,NSTR,MKNP)
COMMON/DSOLA/ ELEIFV(MKEDOF,MKNP),EK(MKUTSZ,MKNP),
6 MEL(MKUTSZ,MKNP),TRNMTZ(MKEDOF,MKNDUF,MKNP),
6 ELESTR(MGCP,MKQUPR,MKNP)
C
C      COMMON/LGNHV/ TRNE(MNDEL,MKNP)
COMMON/IGENM/ EOSTAT(MGCP,MKNP),ENSTAT(MGCP,MKNP),
6 ELINC(MNDEL,MKNP),BELDST(MKEDOF,MKNP)
C
REAL*8 CE,CG,UE,UG,DUE,DUG,UEN1(MKEDOF,MKNP),
6 UGN1(MGCP,MKEDOF,MKNP)
REAL*8 PREN,PREN1,BREN,BREN1,RTSE,PPE
REAL*8 ELEIFV,TRNMTZ,EK,MEL,ELESTR
LOGICAL TRNE
DIMENSION ESTATE(MGCP,MKNP)
EQUIVALENCE (EOSTAT,ESTATE), (UE,UEN1), (UG,UGN1)

.....
.....
Element common area for BEV/MPC:
.....
.....

C      COMMON/DDSVP/ CE(MKEDOF,MKNP),CG(MGCP,MKEDOF,MKNP),
6 UE(MKEDOF,MKNP),UG(MGCP,MKEDOF,MKNP),DUE(MKEDOF,MKNP),
6 DUG(MGCP,MKEDOF,MKNP)
C
COMMON/DSTRM/ PPE(MGCP,NPP,MKNP),RTSE(MGCP,NSTR,MKNP),
6 PREN(MGCP,NSTR,MKNP),PREN1(MGCP,NSTR,MKNP),BREN(MGCP,NSTR,MKNP),
6 BREN1(MGCP,NSTR,MKNP)
C
COMMON/DSOLA/ ELEIFV(MKEDOF,MKNP),EK(MKUTSZ,MKNP),
6 MEL(MKUTSZ,MKNP),TRNMTZ(MKEDOF,MKNDUF,MKNP),
6 ELESTR(MGCP,MKQUPR,MKNP)
C
COMMON/LGNHV/ TRNE(MNDEL,MKNP)
C
COMMON/IGENM/ EOSTAT(MGCP,MKNP),ENSTAT(MGCP,MKNP),
6 ELINC(MNDEL,MKNP),BELDST(MKEDOF,MKNP)
C
REAL*8 CE,CG,UE,UG,DUE,DUG,UEN1(MKEDOF,MKNP),
6 UGN1(MGCP,MKEDOF,MKNP)
REAL*8 PREN,PREN1,BREN,BREN1,RTSE,PPE
REAL*8 ELEIFV,TRNMTZ,EK,MEL,ELESTR
LOGICAL TRNE
DIMENSION ESTATE(MGCP,MKNP)
EQUIVALENCE (EOSTAT,ESTATE), (UE,UEN1), (UG,UGN1)

.....
.....
Element common area for BNPC:
.....
.....

```

```

.....
.....
C      COMMON/DDSVP/ CE(MKEDOF,MKNP),UE(MKEDOF,MKNP),DUE(MKEDOF,MKNP)
COMMON/DSTRM/ PREN(NSTR,MGCP,MKNP),PREN1(NSTR,MGCP,MKNP),
6 BREN(NSTR,MGCP,MKNP),BREN1(NSTR,MGCP,MKNP),RTSE(NSTR,MGCP,MKNP),
6 PPE(NPP,MGCP,MKNP)
C
COMMON/DSOLA/ GPIFY(MKEDOF,MKNP),TRNMTZ(MKEDOF,MKNDUF,MKNP),
6 GK(MKEDOF,MKEDOF,MKNP),MEL(MKEDOF,MKEDOF,MKNP),
6 ELESTR(MKQUPR,MGCP,MKNP)
C
COMMON/DGENV/ NEWVOL(MKNP),OLDVOL(MKNP)
C
COMMON/LGNHV/ TRNE(MNDEL,MKNP)
C
COMMON/IGENM/ EOSTAT(MGCP,MKNP),ENSTAT(MGCP,MKNP),
6 ELINC(MNDEL,MKNP),BELDST(MKEDOF,MKNP)
C
REAL*8 CE,UE,DUE,UEN1(MKEDOF,MKNP)
REAL*8 PREN,PREN1,BREN,BREN1,RTSE,PPE
REAL*8 GPIFY,TRNMTZ,EK,MEL,ELESTR
REAL*8 NEWVOL,OLDVOL,VOLUME(MKNP)
LOGICAL TRNE
DIMENSION ESTATE(MGCP,MKNP)
EQUIVALENCE (EOSTAT,ESTATE), (UE,UEN1), (OLDVOL,VOLUME)

```

7.2.3 INTERNAL FORCE VECTOR

FORTTRAN code for the internal force vector calculation is provide here, in order, for the following element computation algorithms: BEC, BEV, CBEV, BEC/MPV, BEV/MPC, and BMPC for the Alliant FX/8 and BEV for the Convex C240. All code directly relevant to the internal force vector calcualtion is listed, but code for support routines pertinent to many different calculations, such as for the computation of the deformation gradient, is not given for the sake of brevity.


```

C      END IF
C
C      GATHER ELEMENT PK STRESSES AT STATE (N+1).
C
C      CALL ASSIGN(PKEM1(1,1,PNUM),PKM1(STRNAP(ELEM)),NSTR*NGP)
C
C      RETURN
C      END
C
C      SUBROUTINE RMIVFV(ELEM,PNUM)
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'common.main'
C      LOGICAL GEONL
C      NOCONCUR
C
C      TYPE= IPROPS(1,ELEM)
C      NMODE= IPROPS(2,ELEM)
C      ORDER= IPROPS(3,ELEM)
C      NGP= IPROPS(4,ELEM)
C      NDOF= IPROPS(5,ELEM)
C      GEONL= LPROPS(18,ELEM)
C
C      BRANCH ON ELEMENT TYPE.
C
C      GO TO (100,100) TYPE
C
C      ELEMENT TYPES ONE AND TWO, 3D LINEAR
C      AND QUADRATIC ISOPARAMETRICS.
C
C      100 CONTINUE
C
C      INITIALIZE IFV'S.
C
C      DO 105 I= 1,3*NMODE
C      IFV(I,PNUM)= 0.0
C      105 CONTINUE
C
C      COMPUTE ELEMENT IFV, ONE GAUSS POINT AT
C      A TIME.
C
C      MOVECTOR
C      DO 110 GPN= 1,NGP
C
C      CALL GPIFV(ELEM,TYPE,ORDER,GPN,NMODE,PKEM1(1,GPN,PNUM),
C      IFV(I,PNUM),CE(1,PNUM),UEN1(1,PNUM),GEONL)
C
C      110 CONTINUE
C
C      TRANSFORM ELEMENT INTERNAL FORCE
C      VECTOR TO CGS COORDINATES.
C
C      CALL TRNVEC(ELEIFV(1,PNUM),TRNNT(1,1,PNUM),TIME(1,PNUM),
C      NDOF,NMODE,1)
C      GO TO 9999

```

```

C
C      9999 RETURN
C      END
C
C
C      SUBROUTINE GPIFV(ELEM,TYPE,ORDER,GPN,NMODE,PKM1,ELEIFV,CE,UEN1,
C      GEONL)
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'param_def'
C      REAL*8 CE(1),UEN1(1),PKM1(1),ELEIFV(1),NXX(MXDEL),NXX(MXDEL),
C      NXX(MXDEL),JAC(MXDEL),JAC(MXDEL),GAMA(MXDEL),GAMA(MXDEL),
C      B(MXDOF,NSTR),BL(MXDOF,NSTR),BNL(MXDOF,NSTR),
C      BS(MXDOF,NSTR),N,XI,ETA,ZETA,DJ
C      LOGICAL GEONL
C      NOCONCUR
C
C      COMPUTE THE SHAPE FUNCTION DERIVATIVES
C      FOR THE GIVEN GAUSS POINT.
C
C      IF(TYPE.EQ.1) THEN
C      CALL QUAD1(ORDER,GPN,XI,ETA,ZETA,N)
C      CALL DERIV1(XI,ETA,ZETA,NXI,NETA,NZETA)
C      ELSE
C      CALL QUAD2(ORDER,GPN,XI,ETA,ZETA,N)
C      CALL DERIV2(XI,ETA,ZETA,NXI,NETA,NZETA)
C      END IF
C
C      COMPUTE THE STRAIN-DISPLACEMENT MATRICES
C      FOR THE GIVEN GAUSS POINT.
C
C      CALL JACOBI(ELEM,GPN,JAC,DJ,GAMA,NXI,NETA,NZETA,CE,NMODE)
C      IF(GEONL) CALL TCOMPL(THETA,NXI,NETA,NZETA,GAMA,UEN1,NMODE)
C      CALL BCOMPL(B,BS,BL,BNL,THETA,GAMA,NXI,NETA,NZETA,NMODE,GEONL)
C
C      COMPUTE THE INTERNAL FORCE VECTOR
C      FOR THE ELEMENT.
C
C      DO 10 I= 1,3*NMODE
C      ELEIFV(I)= ELEIFV(I) + B(I,1)*PKM1(1)*N*DJ +
C      B(I,2)*PKM1(2)*N*DJ +
C      B(I,3)*PKM1(3)*N*DJ +
C      B(I,4)*PKM1(4)*N*DJ +
C      B(I,5)*PKM1(5)*N*DJ +
C      B(I,6)*PKM1(6)*N*DJ
C      10 CONTINUE
C
C      RETURN
C      END
C
C      SUBROUTINE ADDIFV(ELEM,PNUM)
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'common.main'
C      INCLUDE 'common.elblk'
C      NOCONCUR
C
C      CVD$R NOCONCUR
C
C
C

```



```

C
C      SUBROUTINE ADDIPV(SPAN,BLK)
C      IMPLICIT INTEGER (A-S)
C      INCLUDE 'common.main'
C      INCLUDE 'common.eibit'
CVD$R NOCONCUR
C
C      TOTDOF= IPROPS(2,ELEMS(1,BLK)),IPROPS(4,ELEMS(1,BLK))
C      DO 10 J= 1,TOTDOF
C        DO 10 I= 1,SPAN
C          IPV(BELDST(I,J))= IPV(BELDST(I,J)) + BLZIPV(I,J)
C        10 CONTINUE
C      RETURN
C      END
C
C      SUBROUTINE IFVDV(STRT,NSTRT,BLK,PNUM)
C      IMPLICIT INTEGER (A-S)
C      INCLUDE 'common.main'
CVD$R NOCONCUR
C
C      COMPUTE AND ASSEMBLE THE IPV'S
C      FOR A BLOCK OF SIMILAR, NON-
C      CONFLICTING ELEMENTS.
C
C      SPAN= NSTRT-STRT
C      CALL DUPIPV(SPAN,ELEMS(STRT,BLK),PNUM)
C      CALL RMIPV(SPAN,ELEMS(STRT,BLK),PNUM)
C      CALL ADDIPV(SPAN,ELEMS(STRT,BLK),PNUM)
C      RETURN
C      END
C
C      SUBROUTINE DUPIPV(SPAN,BELEMS,PNUM)
C      IMPLICIT INTEGER (A-S)
C      INCLUDE 'common.main'
C      INCLUDE 'common.eibit'
C      LOGICAL GEONL
C      DIMENSION BELEMS(1)
CVD$R NOCONCUR
C
C      NNODE= IPROPS(2,BELEMS(1))
C      NGP= IPROPS(6,BELEMS(1))
C      NDOF= IPROPS(4,BELEMS(1))
C      GEONL= LPROPS(18,BELEMS(1))
C      TOTDOF= NNODE*NDOF
C      GATHER ELEMENT DESTINATION ARRAYS.
C      DO 10 J= 1,TOTDOF
C        DO 10 I= 1,SPAN
C          BELDST(I,J,PNUM)= EDEST(J,BELEMS(1))
C        10 CONTINUE
C      GATHER ELEMENT TRANSFORMATION FLAGS.

```

```

C
C      SUBROUTINE ADDIPV(SPAN,BLK)
C      IMPLICIT INTEGER (A-S)
C      INCLUDE 'common.main'
C      INCLUDE 'common.eibit'
CVD$R NOCONCUR
C
C      TOTDOF= IPROPS(2,ELEMS(1,BLK)),IPROPS(4,ELEMS(1,BLK))
C      DO 10 J= 1,TOTDOF
C        DO 10 I= 1,SPAN
C          IPV(BELDST(I,J))= IPV(BELDST(I,J)) + BLZIPV(I,J)
C        10 CONTINUE
C      RETURN
C      END
C
C      SUBROUTINE IFVDV(STRT,NSTRT,BLK,PNUM)
C      IMPLICIT INTEGER (A-S)
C      INCLUDE 'common.main'
CVD$R NOCONCUR
C
C      COMPUTE AND ASSEMBLE THE IPV'S
C      FOR A BLOCK OF SIMILAR, NON-
C      CONFLICTING ELEMENTS.
C
C      SPAN= NSTRT-STRT
C      CALL DUPIPV(SPAN,ELEMS(STRT,BLK),PNUM)
C      CALL RMIPV(SPAN,ELEMS(STRT,BLK),PNUM)
C      CALL ADDIPV(SPAN,ELEMS(STRT,BLK),PNUM)
C      RETURN
C      END
C
C      SUBROUTINE DUPIPV(SPAN,BELEMS,PNUM)
C      IMPLICIT INTEGER (A-S)
C      INCLUDE 'common.main'
C      INCLUDE 'common.eibit'
C      LOGICAL GEONL
C      DIMENSION BELEMS(1)
CVD$R NOCONCUR
C
C      NNODE= IPROPS(2,BELEMS(1))
C      NGP= IPROPS(6,BELEMS(1))
C      NDOF= IPROPS(4,BELEMS(1))
C      GEONL= LPROPS(18,BELEMS(1))
C      TOTDOF= NNODE*NDOF
C      GATHER ELEMENT DESTINATION ARRAYS.
C      DO 10 J= 1,TOTDOF
C        DO 10 I= 1,SPAN
C          BELDST(I,J,PNUM)= EDEST(J,BELEMS(1))
C        10 CONTINUE
C      GATHER ELEMENT TRANSFORMATION FLAGS.

```



```

C
      B(KIVL,MIEDOF,NSTR),BL(KIVL,MIEDOF,NSTR),XI,ETA,SETA,M,
      B(KIVL,MIEDOF,NSTR),BS(KIVL,MIEDOF,NSTR)
      LOGICAL GEONL
      DIMENSION BELENS(1)
      CVD$R NOCONCUR
C
      COMPUTE THE SHAPE FUNCTION DERIVATIVES
      FOR THE GIVEN GAUSS POINT.
C
      IF(TYPE.EQ.1) THEN
        CALL QUAD1(ORDER,GM,XI,ETA,SETA,M)
        CALL DERIV1(XI,ETA,SETA,MXI,META,MSETA)
      ELSE
        CALL QUAD2(ORDER,GM,XI,ETA,SETA,M)
        CALL DERIV2(XI,ETA,SETA,MXI,META,MSETA)
      END IF
C
      COMPUTE THE STRAIN-DISPLACEMENT MATRICES
      FOR THE GIVEN GAUSS POINT.
C
      CALL JACOBI(SPAN,BELENS,GM,JAC,DJ,GAMA,MXI,META,MSETA,CE,MNODE)
      IF(GEONL) CALL TCOMPI(SPAN,THETA,MXI,META,MSETA,GAMA,UEM1,MNODE)
      CALL BCOMPI(SPAN,B,BS,BL,BML,THETA,MXI,META,MSETA,MNODE,
      GEONL)
C
      COMPUTE THE INTERNAL FORCE VECTORS
      FOR THE ELEMENT BLOCK.
C
      DO 10 J= 1,MNODE
        DO 10 I= 1,SPAN
          ELIFV(I,J)= ELIFV(I,J) +
            B(I,J,1)*PGINI(1,1)*W*DJ(I) +
            B(I,J,2)*PGINI(1,2)*W*DJ(I) +
            B(I,J,3)*PGINI(1,3)*W*DJ(I) +
            B(I,J,4)*PGINI(1,4)*W*DJ(I) +
            B(I,J,5)*PGINI(1,5)*W*DJ(I) +
            B(I,J,6)*PGINI(1,6)*W*DJ(I)
        CONTINUE
      CONTINUE
C
      RETURN
      END
C
      SUBROUTINE ADDIFV(SPAN,BELENS,PHUM)
      IMPLICIT INTEGER (A-Z)
      INCLUDE 'COMMON.MELA'
      INCLUDE 'COMMON.EIBLK'
      DIMENSION BELENS(1)
      CVD$R NOCONCUR
      CVD$R MODEPCNR
C
      TOTDOF= IPROPS(2,BELENS(1))*IPROPS(4,BELENS(1))
C
      DO 20 J= 1,TOTDOF
        DO 20 I= 1,SPAN
          IFV(BELOST(I,J,PHUM))= IFV(BELOST(I,J,PHUM)) +
            ELIFV(I,J,PHUM)
        CONTINUE
      CONTINUE
C
      SET FLAG INDICATING THAT THE INTERNAL FORCE
      VECTOR HAS BEEN CALCULATED.
      IFVCNP= .TRUE.

```



```

C
C      VECTOR TO CGG COORDINATES.
C
C      CALL TRNVEC(IELEFV(1,PNUM), TRNMTS(1,1,PNUM), TRNE(1,PNUM),
C      NDOF,NODE,1)
C
C      GO TO 9999
C
C      9999 RETURN
C      END
C
C      SUBROUTINE GPTEV(IELEM,TYPE,ORDER,NGP,NODE,PREN1,ELEFV,CE,UEM1,
C      GEOML)
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'param.def'
C      REAL*8 CE(MGP,1),UEM1(MGP,1),PREN1(MGP,1),ELEFV(1),
C      META(MGP,MDEL),MSETA(MGP,MDEL),THETA(MGP,MXNSZ),
C      JAC(MGP,NDIM,NDIM),GAMA(MGP,NDIM,NDIM),NMI(MGP,MNDEL),
C      B(MGP,MXDOF,MSTR),BL(MGP,MXDOF,MSTR),
C      ENL(MGP,MXDOF,MSTR),BS(MGP,MXDOF,MSTR),N(MGP),
C      XI(MGP),ETA(MGP),ZETA(MGP),DJ(MGP)
C      LOGICAL GEOML
C      NOCONCUR
C
C      COMPUTE THE SHAPE FUNCTION DERIVATIVES.
C
C      IF(TYPE.EQ.1) THEN
C      CALL QUAD(ORDER,NGP,XI,ETA,ZETA,N)
C      CALL DERIV1(NGP,XI,ETA,ZETA,NMI,META,MSETA)
C      ELSE
C      CALL QUAD2(ORDER,NGP,XI,ETA,ZETA,N)
C      CALL DERIV2(NGP,XI,ETA,ZETA,NMI,META,MSETA)
C      END IF
C
C      COMPUTE THE STRAIN-DISPLACEMENT MATRICES.
C
C      CALL JACOBI(NGP,ELEM,JAC,DJ,GAMA,NMI,META,MSETA,CE,NNODE)
C      IF(GEOML) CALL TCOMP1(NGP,THETA,NMI,META,MSETA,GAMA,UEM1,NNODE)
C      CALL BCOMP1(NGP,B,BS,BL,ENL,THETA,GAMA,NMI,META,MSETA,NODE,GEOML)
C
C      COMPUTE THE INTERNAL FORCE VECTOR
C      FOR THE ELEMENT.
C
C      DO 10 J= 1,3*NNODE
C      DO 10 I= 1,NGP
C      ELEFV(J)= ELEFV(J) + B(I,J,1)*PREN1(I,1)*N(I,1)*DJ(I) +
C      B(I,J,2)*PREN1(I,2)*N(I,1)*DJ(I) +
C      B(I,J,3)*PREN1(I,3)*N(I,1)*DJ(I) +
C      B(I,J,4)*PREN1(I,4)*N(I,1)*DJ(I) +
C      B(I,J,5)*PREN1(I,5)*N(I,1)*DJ(I) +
C      B(I,J,6)*PREN1(I,6)*N(I,1)*DJ(I)
C
C      10 CONTINUE
C
C      RETURN
C      END
C
C      SUBROUTINE ADDIFV(ELEM,PNUM)
C
C      SET FLAG INDICATING THAT THE INTERNAL FORCE
C      VECTOR HAS BEEN CALCULATED.
C
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'common.main'
C      INCLUDE 'common.elblk'
C
C      CVD$R NOCONCUR
C
C      TOTDOF= IPROPS(2,ELEM)*IPROPS(4,ELEM)
C
C      DO 10 I= 1,TOTDOF
C      IFV(BELDST(I,PNUM))= IFV(BELDST(I,PNUM)) + ELEFV(I,PNUM)
C      10 CONTINUE
C
C      RETURN
C      END
C
C      Internal force vector calculations code for BEV/WOC.
C      *****
C
C      SUBROUTINE INFVEC
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'common.main'
C
C      C
C      C      INITIALIZE THE GLOBAL INTERNAL
C      C      FORCE VECTOR.
C
C      DO 10 I= 1,NODOF
C      IFV(I)= 0.0
C      10 CONTINUE
C
C      COMPUTE AND ASSEMBLE THE IFV'S
C      FOR EACH BLOCK OF SIMILAR, NON-
C      CONFLICTING ELEMENTS.
C
C      CVD$ NOVECTOR
C      CVD$ NOCONCUR
C      DO 15 BLK= 1,NELBLK
C
C      SPAN= ELBLKS(0,BLK)
C
C      CALL DUPIFV(SPAN,BLK)
C
C      CALL RENIFV(SPAN,BLK)
C
C      CALL ADDIFV(SPAN,BLK)
C
C      15 CONTINUE
C
C      SET FLAG INDICATING THAT THE INTERNAL FORCE
C      VECTOR HAS BEEN CALCULATED.

```



```

25      CE(I,PNUM)= C(CDEST(I,ELEM))
      CONTINUE
C
C
C      IF(GEONL) THEN
C
C          GATHER ELEMENT DISPLACEMENTS AT STATE (N+1).
C
C          DO 30 I= 1,TOTDOF
C              UENI(I,PNUM)= U(BELDST(I,PNUM))*DU(BELDST(I,PNUM))
C              CONTINUE
C
C          TRANSFORM ELEMENT DISPLACEMENTS TO
C          UG COORDINATES.
C
C          CALL TRNVEC(UENI(1,PNUM),TRNMT(1,1,PNUM),TRNE(1,PNUM),
C          NDOF,NNODE,2)
C
C          END IF
C
C          GATHER ELEMENT PK STRESSES AT STATE (N+1).
C
C          CALL ASSIGN(PKENI(1,1,PNUM),PENI(STRMAP(ELEM)),NSTR*NGP)
C
C          P1TURN
C          END
C
C          SUBROUTINE RENIV(ELEM,GNP,PNUM)
C              IMPLICIT INTEGER (A-S)
C              INCLUDE 'common.mis'
C              INCLUDE 'common.elblk'
C              LOGICAL GEONL
C              NOCONCUR
C
C              TYPE= IPROPS(1,ELEM)
C              NNODE= IPROPS(2,ELEM)
C              ORDER= IPROPS(5,ELEM)
C              NDOF= IPROPS(4,ELEM)
C              GEONL= IPROPS(18,ELEM)
C
C              GO TO (100,100) TYPE
C
C              ELEMENT TYPES ONE AND TWO: 3D LINEAR
C              AND QUADRATIC ISOPARAMETRICS.
C
C              100 CONTINUE
C
C              COMPUTE THE INTERNAL FORCE VECTOR FOR THE
C              CURRENT GLOBAL GAUSS POINT.
C
C              CALL GPIFV1(ELEM,TYPE,ORDER,GNP,NNODE,PKENI(1,GNP,PNUM),
C              GPIV(1,PNUM),CE(1,PNUM),UENI(1,PNUM),GEONL)

```

```

C
C      TRANSFORM GLOBAL GP INTERNAL FORCE VECTOR TO
C      CGC COORDINATES.
C
C      CALL TRNVEC(GPIFV(1,PNUM),TRNMT(1,1,PNUM),TRNE(1,PNUM),NDOF,
C      NNODE,1)
C
C      GO TO 9999
C
C      9999 RETURN
C      END
C
C      SUBROUTINE GPIFV1(ELEM,TYPE,ORDER,GNP,NNODE,PKENI,GPIFV,CE,
C      UENI,GEONL)
C          IMPLICIT INTEGER (A-S)
C          INCLUDE 'PARAM.def'
C          REAL*8 CE(1),UENI(1),PKENI(1),GPIFV(1),NII(NNODE),NETA(NNODE),
C          NZETA(NNODE),JAC(NDIM,NDIM),GAMA(NDIM,NDIM),THETA(KXTNSZ),
C          B(MIEDOF,NSTR),BL(MIEDOF,NSTR),BNL(MIEDOF,NSTR),
C          BS(MIEDOF,NSTR),N,XI,ETA,ZETA,DJ
C          LOGICAL GEONL
C          NOCONCUR
C
C          COMPUTE THE SHAPE FUNCTION DERIVATIVES
C          FOR THE GIVEN GAUSS POINT.
C
C          IF(TYPE.EQ.1) THEN
C              CALL QUAD1(ORDER,GNP,XI,ETA,ZETA,N)
C              CALL DERIV1(XI,ETA,ZETA,NXI,NETA,NZETA)
C          ELSE
C              CALL QUAD2(ORDER,GNP,XI,ETA,ZETA,N)
C              CALL DERIV2(XI,ETA,ZETA,NXI,NETA,NZETA)
C          END IF
C
C          COMPUTE THE STRAIN-DISPLACEMENT MATRICES
C          FOR THE GIVEN GAUSS POINT.
C
C          CALL JACOBI(ELEM,GNP,JAC,DJ,GAMA,NXI,NETA,NZETA,CE,NNODE)
C          IF(GEONL) CALL TCOMP1(THETA,NXI,NETA,NZETA,GAMA,UENI,NNODE)
C          CALL BCOMP1(B,BS,BL,BNL,THETA,GAMA,NXI,NETA,NZETA,GEONL)
C
C          COMPUTE THE INTERNAL FORCE VECTOR
C          FOR THE GAUSS POINT.
C
C          DO 10 I= 1,3*NNODE
C              GPIFV(I)= B(1,1)*PKENI(1)*N*DJ +
C              B(1,2)*PKENI(2)*N*DJ +
C              B(1,3)*PKENI(3)*N*DJ +
C              B(1,4)*PKENI(4)*N*DJ +
C              B(1,5)*PKENI(5)*N*DJ +
C              B(1,6)*PKENI(6)*N*DJ
C          10 CONTINUE
C
C          RETURN
C      END
C
C
C

```



```

10 CONTINUE
C
C
RETURN
END
C
C
C
SUBROUTINE ADDIFV(SPAN,BLK)
IMPLICIT INTEGER (A-Z)
INCLUDE 'common.m4a1a'
INCLUDE 'common.elblk'
C
C
TOTDOP= IPROPS(2,ELBLES(1,BLK))*IPROPS(4,ELBLES(1,BLK))
C
DO 10 J= 1,TOTDOP
NO_RECURRENCE
DO 10 I= 1,SPAN
IFV(BELDST(1,J))= IFV(BELDST(1,J)) + ELEIFV(1,J)
10 CONTINUE
C
C
RETURN
END

```

7.2.4 STEP LENGTH CALCULATION

FORTTRAN code is provided, in order, for the step length calculation of the following element computation algorithms: BEC, BEV, and CBEV for the Alliant FX/8 and BEV for the Convex C240. Again, auxillary code is not provided, only code directly relevant to the step length calculation.

.....

step length calculation for BEC:

.....

```

SUBROUTINE LFINDA(STEP,STPTR,ITER,TRCPG,SLTR1,ALPHA)
  IMPLICIT INTEGER (A-Z)
  INCLUDE 'common.meln'
  REAL*8 SLTR1,ALPHA,NFAC,PTAP
  LOGICAL TRCPG

```

C SET MENARK MULTIPLICATION FACTOR.

```

NFAC= 1.0/(NBETA*DT)

```

C PERFORM LINEAR MATRIX PRODUCT. THE LM
C PRODUCT IS STORED IN THE INTERNAL FORCE
C VECTOR, AS THE IFV WILL NOT BE USED AGAIN
C BEFORE ITS NEXT UPDATE.

```

CALL LMPRD(NFAC,IFV)

```

C COMPUTE THE TRIPLE PRODUCT SDIR TRANSPOSE * LM *
C SDIR.

```

PTAP= 0.0
DO 10 I= 1,NODOF
  PTAP= PTAP+SDIR(I)*IFV(I)
CONTINUE

```

C COMPUTE THE STEP LENGTH.

```

ALPHA= SLTR1/PTAP

```

C OUTPUT THE VALUE OF ALPHA IF THE TRACE FLAG
C IS ON.

```

IF (TRCPG) THEN
  CALL QUALPR(STEP,STPTR,ITER,ALPHA)
END IF

```

```

RETURN
END

```

```

SUBROUTINE LAPRD(NFAC,PRD)
  IMPLICIT INTEGER (A-Z)
  INCLUDE 'common.meln'
  INCLUDE 'common.elblk'
  REAL*8 PRD(1)

```

C INITIALIZE THE PRODUCT VECTOR.

```

DO 10 I= 1,NODOF
  PRD(I)= 0.0

```

10 CONTINUE
C PERFORM THE MULTIPLICATION IN ELEMENT BLOCKS.

```

CVD$ NOVECTOR
NCONCUR
DO 15 BLK= 1,NELBLK

```

C SPAN= ELBLKS(0,BLK)

```

C CND$ CND$
DO 10 BELE= 1,SPAN
  CALL ELNGTH(ELBLKS(BELE,BLK),NFAC,CP(1,BELE),EMAT(1,BELE))
CONTINUE

```

C CND\$

```

DO 25 BELE= 1,SPAN
  CALL ELMPRD(ELBLKS(BELE,BLK),CP(1,BELE),
    EDEST(1,ELBLKS(BELE,BLK)),
    EMAT(1,BELE),SDIR,PRD)

```

25 CONTINUE

15 CONTINUE

C RETURN
C END

```

SUBROUTINE ELNGTH(ELEM,NFAC,LCP,MATBL)
  IMPLICIT INTEGER (A-Z)
  INCLUDE 'common.meln'
  DIMENSION LCP(1)
  REAL*8 MATBL(1),NFAC

```

CVD\$R NOCONCUR

```

TOTDOF= IPR*OS(2,ELEM)*IPROPS(4,ELEM)
UTSZ=((TOTDOF*TOTDOF)-TOTDOF)/2+TOTDOF

```

C GATHER THE LINEAR PCG MATRIX FOR THE CURRENT
C BLOCK ELEMENT.

```

DO 10 I= 1,UTSZ
  MATBL(I)= K(I,ELEM)*NFAC*M(I,ELEM)
CONTINUE

```

10 CONTINUE
C CONSTRAIN THE LINEAR PCG MATRIX FOR THE
C CURRENT BLOCK ELEMENT.

```

DO 20 J= 1,ECSTMP(0,ELEM)

```

C JJ= ECSTMP(J,ELEM)

```

DO 25 I= 1,JJ-1
  MATBL(LCP(JJ)+I)= 0.0
CONTINUE

```

25 MATBL(LCP(JJ)+JJ)= 1.0

C DO 30 I= JJ+1,TOTDOF

MATBL(LCP(I)+JJ)= 0.0

30 CONTINUE

C

20 CONTINUE

C

RETURN

C

END

C

C

C

SUBROUTINE ELMPRD(ELEM,LCP,EDBL,MATBL,X,Y)
IMPLICIT INTEGER (A-S)
INCLUDE 'common.mai'
DIMENSION EDBL(1),LCP(1)
REAL*8 B,MATBL(1),X(1),Y(1),XE(MXEDOF),YE(MXEDOF)

CVD\$R NOCONCUR

C

C

TOTDOF= IPROPS(2,ELEM)*IPROPS(4,ELEM)

C

C

C

DO 10 I= 1,TOTDOF

XE(I)= X(EDBL(I))

YE(I)= 0.0

CONTINUE

10

C

C

DO 15 K= 1,TOTDOF

C

C

CCP= LCP(ROW)

C

DO 20 I= 1,ROW

YE(ROW)= YE(ROW)+MATBL(I+CCP)*XE(I)

CONTINUE

20

C

DO 25 I= ROW+1,TOTDOF

YE(ROW)= YE(ROW)+MATBL(ROW+LCP(I))*XE(I)

CONTINUE

25

C

C

CONTINUE

C

CVD\$R MODEPCHK

DO 30 I= 1,TOTDOF

Y(EDBL(I))= Y(EDBL(I))+YE(I)

CONTINUE

30

C

C

RETURN

END

C

.....

Step length calculation for Alliant BEV:

.....

SUBROUTINE LPFINDA(STEP,STPITR,ITER,TRCPCG,ZITRI,ALPHA)
IMPLICIT INTEGER (A-S)
INCLUDE 'common.mai'
REAL*8 ZITRI,ALPHA,NFAC,PTAP
LOGICAL TRCPCG

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C


```

C
SPAN= ELES(0,BLK)
TOTDOF= IPROPS(3,ELES(1,BLK))*IPROPS(4,ELES(1,BLK))
UTS2=((TOTDOF-TOTDOF-TOTDOF)/2+TOTDOF
C
CALL ELMPTR(SPAN,ELES(1,BLK),NFAC,UTS2,TOTDOF,ENAT,BELDST)
C
CALL ELMPD(SPAN,CP(1,1),TOTDOF,BELDST,ENAT,SDIR,PRD)
C
15 CONTINUE
C
RETURN
END
C
SUBROUTINE ELMPTR(SPAN,ELES,NFAC,UTS2,TOTDOF,MATBL,EDBL)
IMPLICIT INTEGER (A-Z)
INCLUDE 'COMMON.MEIN'
DIMENSION ELES(1),EDBL(MNVL,1)
REAL*8 MATBL(MNVL,1),NFAC
C
C GATHER THE LINEAR PCG MATRIX FOR THE CURRENT
C ELEMENT BLOCK.
C
DO 10 J= 1,UTS2
DO 10 I= 1,SPAN
MATBL(I,J)= ELES(I,J)+NFAC*M(ELES(1),J)
10 CONTINUE
C
DO 15 J= 1,TOTDOF
DO 15 I= 1,SPAN
EDBL(I,J)= EDBL(J,ELES(1))
15 CONTINUE
C
CNCALC
DO 20 BELE= 1,SPAN
CALL APCHL(BELE,TOTDOF,ELES(BELE),ECSTMP(0,ELES(BELE)),
ECSTMP(1,ELES(BELE)),CP(1,BELE),MNVL,MATBL)
20 CONTINUE
C
RETURN
END
C
SUBROUTINE ELMPD(SPAN,CP,TOTDOF,EDST,MAT,X,Y)
IMPLICIT INTEGER (A-Z)
INCLUDE 'PARAM.DAT'
DIMENSION EDST(MNVL,1),CP(1)
REAL*8 MAT(MNVL,1),X(1),Y(1),X2(MNVL,M2EDOF),YE(MNVL,M2EDOF),
AXE(MNVL,M2EDOF)
C
DO 15 J= 1,TOTDOF

```

```

C      CALL LMPDR(NFAC,ITV)
C
C      COMPUTE THE TRIPLE PRODUCT SDIR TRANSPOSE * IM *
C      SDIR.
C
C      PTAP= 0.0
C      DO 10 I= 1,NODOF
C      PTAP= PTAP+SDIR(I)*ITV(I)
C      CONTINUE
C
C      COMPUTE THE STEP LENGTH.
C
C      ALPHA= 2*ITR/PTAP
C
C      OUTPUT THE VALUE OF ALPHA IF THE TRACE FLAG
C      IS ON.
C
C      IF (TRCPG) THEN
C      CALL OUALP(STEP,STEPTR,ITER,ALPHA)
C      END IF
C
C      RETURN
C      END
C
C      SUBROUTINE LMPDR(NFAC,PRD)
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'COMMON.main'
C      REAL*8 NFAC,PRD(1)
C
C      INITIALIZE THE PRODUCT VECTOR.
C
C      DO 10 I= 1,NODOF
C      PRD(I)= 0.0
C      CONTINUE
C
C      PERFORM THE MULTIPLICATION IN BLOCKS OF
C      CONCURRENT, NON-CONFLICTING ELEMENT BLOCKS.
C
C      NOVECTOR
C      NOCONCUR
C      DO 20 BLK= 1,NELBLK
C
C      CSPAN= EBSFNS(0,BLK)
C
C      CMCALL
C      DO 25 CBLK= 1,CSPAN
C      CALL LMPDR(EBSFNS(CBLK,BLK),EBSFNS(CBLK+1,BLK),BLK,CBLK,
C      NFAC,PRD)
C      CONTINUE
C
C      CONTINUE
C
C      RETURN
C      END
C
C
C      SUBROUTINE LMPDR(STRT,NSTRT,BLK,PNUM,NFAC,PRD)
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'COMMON.main'
C      REAL*8 NFAC,PRD(1)
C
C      PERFORM THE MULTIPLICATION OF THIS ELEMENT
C      BLOCK.
C
C      CVD$R NOCONCUR
C
C      SPAN= NSTRT-STRT
C
C      TODOF= IPROPS(2,ELEBLK(STRT,BLK))*IPROPS(4,ELEBLK(STRT,BLK))
C      UTSSZ= ((TODOF+TODOF)-TODOF)/2+TODOF
C
C      CALL ELNGTH(SPAN,ELEBLK(STRT,BLK),CP(1,PNUM),NFAC,UTSSZ,TODOF,
C      ENAT(1,1,PNUM),BEIDST(1,1,PNUM))
C
C      CALL ELMPDR(SPAN,CP(1,PNUM),TODOF,BEIDST(1,1,PNUM),
C      ENAT(1,1,PNUM),SDIR,PRD)
C
C      RETURN
C      END
C
C      SUBROUTINE ELNGTH(SPAN,ELEMS,ICP,NFAC,UTSSZ,TODOF,MATBL,EDBL)
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'COMMON.main'
C      REAL*8 MATBL(MATVL,1),NFAC
C      DIMENSION ELEMS(1),EDBL(MATVL,1),ICP(1)
C
C      CVD$R NOCONCUR
C
C      GATHER THE LINEAR PGM MATRIX FOR THE CURRENT
C      ELEMENT BLOCK.
C
C      DO 10 J= 1,UTSSZ
C      DO 10 I= 1,SPAN
C      MATBL(I,J)= K(ELEMS(I),J)+NFAC*M(ELEMS(I),J)
C      CONTINUE
C
C      GATHER EDEST FOR THE CURRENT ELEMENT BLOCK.
C
C      DO 15 J= 1,TODOF
C      DO 15 I= 1,SPAN
C      EDBL(I,J)= EDEST(J,ELEMS(I))
C      CONTINUE
C
C      CONSTRAIN THE LINEAR PGM MATRIX FOR THE
C      CURRENT ELEMENT BLOCK.
C
C      DO 20 BELE= 1,SPAN
C
C      DO 25 J= 1,ECSFNP(0,ELEMS(BELE))
C      JJ= ECSFNP(J,ELEMS(BELE))
C      DO 30 I= 1,JJ-1
C      MATBL(BELE,ICP(JJ)+I)= 0.0

```


7.2.5 EBE FACTORIZATION

FORTTRAN code is furnished, in order, for the ebe factorization of the following element computation algorithms: BEC, BEV, and CBEV for the Alliant FX/8 and BEV for the Convex C240. Again, auxillary code is not provided, only code directly relevant to the ebe factorization. The operations included in the ebe factorization are the element Winget regularization, the element Crout factorization, and the scatter of the element diagonal scaling vectors to the global level.


```

C
C
C      CALL EBGTHR(SPAN,ELBLS(1,BLK),UTSZ,TOTDOF,PCP,EMAT,EDEST,
C      BELDST)
C
C      CALL WREGGL(SPAN,CP,DCP,TOTDOF,BELDST,DIAG,EMAT)
C      CALL CROUTF(SPAN,CP,TOTDOF,EMAT)
C      CALL EDGSL(SPAN,DCP,TOTDOF,BELDST,EMAT,PDIA)
C
C
C      NODEPCHK
C      DO 20 J= 1,UTSZ
C      DO 30 I= 1,SPAN
C      PCP(ELBLS(1,BLK),J)= EMAT(I,J)
C      CONTINUE
C      CONTINUE
C
C      DEACTIVATE FLAGS NECESSITATING A NEW PCP.
C
C      9999 NEMCHS= .FALSE.
C      NEMHAS= .FALSE.
C      NEMSTF= .FALSE.
C
C      CALL TRYME(10,2)
C
C      RETURN
C      END
C
C
C      SUBROUTINE EBGTHR(SPAN,ELBLS,UTSZ,TOTDOF,MAT,MATBL,EDEST,EDBL)
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'param_def'
C      DIMENSION ELBLS(1),EDEST(MXEDOF,1),EDBL(MXVL,1)
C      REAL*8 MAT(MXEL,1),MATBL(MXVL,1)
C
C      GATHER MAT FOR THE ELEMENTS IN THE CURRENT BLOCK.
C
C      DO 10 J= 1,UTSZ
C      DO 10 I= 1,SPAN
C      MATBL(I,J)= MAT(ELBLS(1),J)
C      CONTINUE
C
C      GATHER EDEST FOR THE ELEMENTS IN THE CURRENT BLOCK.
C
C      DO 15 J= 1,TOTDOF
C      DO 15 I= 1,SPAN
C      EDBL(I,J)= EDEST(J,ELBLS(1))
C      CONTINUE
C
C      RETURN
C      END
C
C
C      SUBROUTINE WREGGL(SPAN,CP,DCP,TOTDOF,EDEST,DIAG,PCP)
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'param_def'
C      DIMENSION EDEST(MXVL,1),DCP(1),CP(1)
C      REAL*8 DIAG(1),PCP(MXVL,1),DINV(MXVL,MXEDOF)

```


298


```

15      PCN(I,COL+CCP)= PCN(I,COL+CCP)-PCN(I,J+CCP)*PCBAR(I,J)
C      CONTINUE
C 15      CONTINUE
C      RETURN
C      END
C
C      SUBROUTINE EDGSCI(SPAN,DCP,TOTDOF,EDEST,PCN,2)
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'param.def'
C      DIMENSION EDEST(NVIL,1),DCP(1)
C      REAL*8 PCN(NVIL,1),Z(1),ZE(NVIL,NEDOF)
CVD$R NOCONCUR
C
C      DO 10 J= 1,TOTDOF
C      DO 10 I= 1,SPAN
C      ZE(I,J)= Z(EDEST(I,J))
C      CONTINUE
C
C      DO 20 J= 1,TOTDOF
C      DO 20 I= 1,SPAN
C      ZE(I,J)= ZE(I,J)/PCN(I,DCP(J))
C      CONTINUE
C
C      CVD$R NODEPCBK
C      DO 30 J= 1,TOTDOF
C      DO 30 I= 1,SPAN
C      ZE(EDEST(I,J))= ZE(I,J)
C      CONTINUE
C
C      RETURN
C      END
C
C.....
C.....
C.....
C      Ebe factorization for Convex BEV,
C.....
C.....
C.....
C      SUBROUTINE UPDPCH(DIAPLG)
C      IMPLICIT INTEGER (A-Z)
C      INCLUDE 'common-main'
C      INCLUDE 'common-ebit'
C      LOGICAL DIAPLG
C
C      CALL TTIME(10,1)
C
C
C      COMPUTE THE PRECONDITIONING MATRIX AND THE
C      DIAGONAL VECTOR.
C
C      CALL SETPCN
C
C      APPLY THE CONSTRAINTS TO THE PCN AND THE
C      DIAGONAL VECTOR.
C
C      CALL APPLICN(DIAPLG)
C
C      IF THE PCN IS DIAGONAL, SKIP THE EBE FACTOR-
C      IZATION PROCESS.
C
C      IF(DIAPLG) GO TO 9999
C
C      INVERT AND TAKE THE SQUARE ROOT OF THE
C      DIAGONAL VECTOR. INITIALIZE THE ELEMENT
C      DIAGONAL SCALING VECTOR.
C
C      DO 10 I= 1,NODOF
C      DIAG(I)= 1.0/DSQRT(DIAG(I))
C      PDJAG(I)= 1.0
C      CONTINUE
C
C      PERFORM MINDET REGULARIZATION AND CROUT
C      FACTORIZATION. MULTIPLY THE ELEMENT DIAGONAL
C      SCALING VECTOR.
C
C      CVD$R SCALAR
C      DO 15 BLK= 1,NELBLK
C
C      SPAN= ELBLKS(0,BLK)
C
C      TOTDOF= IPROPS(2,ELBLKS(1,BLK))*IPROPS(4,ELBLKS(1,BLK))
C      UTSE= (TOTDOF*TOTDOF)-TOTDOF/2*TOTDOF
C
C      CALL EBCIIR(SPAN,ELBLKS(1,BLK),UTSE,TOTDOF,PCN,EMAT,EDEST,
C      BELDST)
C
C      CALL WNRGL(SPAN,CP,DCP,TOTDOF,BELDST,DIAG,EMAT)
C      CALL CROUTF(SPAN,CP,TOTDOF,EMAT)
C      CALL EDGSCI(SPAN,DCP,TOTDOF,BELDST,EMAT,PDJAG)
C
C      DO 20 J= 1,UTSE
C      NO_RECURRENCE
C      DO 20 I= 1,SPAN
C      PCN(ELBLKS(1,BLK),J)= EMAT(I,J)
C      CONTINUE
C
C      15      CONTINUE
C
C      DEACTIVATE FLAGS NECESSITATING A NEW PCN.
C
C      9999  MEMCN= .FALSE.
C      MEMNAS= .FALSE.
C      MEMSTF= .FALSE.
C
C      CALL TTIME(10,2)
C
C

```



```

C      DO 30 J= 1, TOTDOP
      CIDIR  NO_OCCURENCE
      DO 30 I= 1, SPAN
        S(EDST(I,J))= SE(I,J)
      30 CONTINUE
      C
      C      RETURN
      C      END

```

7.2.6 EBE PRECONDITIONING

FORTRAN code is supplied, in order, for the ebe preconditioning of the following element computation algorithms: BEC, BEV, and CBEV for the Alliant FX/8 and BEV for the Convex C240. Again, supplementary code is not provided, only code directly relevant to the ebe preconditioning. The operations included in the ebe preconditioning are the element forward reduction and back substitution and the element and global diagonal scaling.

304


```

C      DO 25 J= 1,TOTDOP
      CSDIR  NO_RECURRENCE
      DO 25 I= 1,SPAN
        R(DEST(I,J))= SE(I,J)
      25 CONTINUE
      C
      C      RETURN
      C      END

```