FINAL REPORT
PHASE I

COMPUTATIONAL CRASH DYNAMICS METHODS
FOR FIFTH GENERATION SUPERCOMPUTERS

By
Robert E. Fulton
Kou-Ning Chiang

Submitted to
General Motors Research Laboratories
Warren, Michigan

December 1986

GEORGIA INSTITUTE OF TECHNOLOGY
A UNIT OF THE UNIVERSITY SYSTEM OF GEORGIA
SCHOOL OF MECHANICAL ENGINEERING
ATLANTA, GEORGIA 30332
COMPUTATIONAL CRASH DYNAMICS METHODS FOR FIFTH GENERATION SUPERCOMPUTERS

by

Robert E. Fulton
Kuo-Ning Chiang
George W. Woodruff
School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

Final Report Phase I Submitted to

General Motors Research Laboratories
Warren, Michigan

Georgia Tech Research Corporation E-25-633 to General Motors Research Laboratories

December 1986
INTRODUCTION

The structural design/analysis of future complex engineering products such as automobiles, trucks and aerospace vehicles subjected to crash impact conditions requires significantly greater analytical complexities than many other design conditions. For example, crash design can include the combined effects of dynamic loads, composite materials, multidisciplinary interactions, three-dimensional geometry and nonlinear behavior. Realistic structural models for such designs typically imply large-order finite element or finite difference models and excessively large computational requirements. Since the advent of NASTRAN (ref. 1), general-purpose finite element structural analysis computer programs (e.g., ref. 2) have provided the capability to address a wide range of structures problems including design, optimization, nonlinear and dynamic capabilities. However, when these programs are applied to crash dynamics problems, more computing effort is required than is available with current sequential computers. Moreover, sequential computers will be inadequate to support analysis requirements for many future engineering designs where effective speeds greater than $10^3$ MFLOPS (million floating point operations per second) will be required. For example, calculations for such problems as nonlinear dynamic response, multidisciplinary interactions and optimum design are all severely limited by computation speed, and models are usually restricted to small-order problems or highly simplifying approximations.
Projected advances in computer technology indicate that significant increases in effective calculation speed will be available in the 1990's, through fifth generation supercomputer architectures consisting of arrays of processors operating in parallel on different tasks (see e.g., ref. 3 for a survey). Such advanced supercomputers, denoted at MIMD (multiple instruction, multiple data) computers, have the potential for increasing effective calculation speeds by several orders of magnitude. But this potential increase in speed cannot be effectively utilized without the development and implementation of appropriate numerical algorithms for structures which take advantage of the parallel computation features of this new generation of computers. Use of existing conventional algorithms and software will not realize the full potential of these new MIMD computers, and research is needed in the development of parallel structural analysis/design algorithms for these computers (ref. 4-19). This research has been to develop and evaluate parallel methods for crash dynamic analyses of complex nonlinear finite element and/or finite difference structural problems.

Issues involved in implementation of parallel structural algorithms on MIMD supercomputers are much more complex than for current sequential computers, and the total hardware/software system must be taken into consideration. The implementation criteria that influence the efficiency of an algorithm include the amount of computation versus the number of processors, the communication paths and synchronization delays, and the size of a problem in relation to the number of processors used. The flow of the algorithm must be analyzed to identify those calculations which must be sequential and those which can be done in parallel. Tasks are partitioned onto a processor array so that the communication paths are most effective. Efficient algorithms typically maximize parallel and minimize sequential calculations.
The finite element equations governing the nonlinear crash dynamic response of a structure take the form:

\[
\ddot{U} + \dot{C}U + KU + G(U) = F(t)
\]  \hspace{1cm} (1)

where \( M, C, \) and \( K \) are the mass, damping and stiffness matrices, \( U \) the displacement vector, \( F \) the forcing function and \( G(U) \) represents the nonlinear effect. Dots denote differentiation with respect to time \( t \). Consider the integration of these equations from time step \( j \) to \( j+1 \) over the time interval \( k \) using, for example, the Newmark trapezoidal integration formula (see ref. 20 with \( \alpha = 1/2, \beta = 1/4 \))

\[
\begin{align*}
\ddot{U}_{j+1} &= \dot{U}_j + \frac{k}{2} [\dot{U}_j + \dot{U}_{j+1}] \\
\dot{U}_{j+1} &= \dot{U}_j + \frac{k}{2} [\dot{U}_j + \dot{U}_{j+1}] \\
U_{j+1} &= U_j + \frac{k}{2} [U_j + U_{j+1}] \\
&= U_j + k\frac{1}{4} [U_j + U_{j+1}]
\end{align*}
\]  \hspace{1cm} (2)

Writing equation (1) at station \( j+1 \) gives

\[
\ddot{U}_{j+1} + \dot{C}_{j+1}U_{j+1} + K_{j+1}U_{j+1} + G(U_{j+1}) = F(t_{j+1})
\]  \hspace{1cm} (4)

When solving equations (2), (3), and (4) on a parallel computer, there are at least two alternative solution approaches for distributing calculations across processors, an iterative approach and a direct approach. The following characterizes some of the key differences in the two approaches relative to a parallel computer implementation.

Consider, for example, an iterative approach to solving equations (2)-(4). The sequence of calculations in integrating from time step \( j \) to \( j+1 \) can be given by equations (5), (6), and (7) as follows:
\[ \dot{U}_{j+1}^{r+1} = U_j + \frac{k}{2} [U_j + U_{j+1}^{r+1}] \]  
(5)

\[ U_{j+1}^{r+1} = U_j + k \dot{U}_j + \frac{k^2}{4} [U_j + U_{j+1}^{r+1}] \]  
(6)

\[ M \ddot{U}_{j+1}^{r+1} = -CU_j + KU_{j+1}^{r+1} - G(U_{j+1}^{r+1}) + F(t_{j+1}) \]  
(7)

where \( r \) denotes the \( r \)th iteration. If \( M \) is diagonal, the equations for each \( j+1 \) variable are decoupled during each iteration cycle. If \( M \) is not diagonal the decoupling of the equations can be retained by using a Jacobi-type iterative approach to solve equation (7). Convergence can be obtained if the time step is sufficiently small.

A direct approach can also be used to solve equations (2)-(4) for \( U_{j+1} \). For example, equations (2) and (3) can be substituted into equation (4) to obtain

\[ KU_{j+1} + G(U_{j+1}) = AU_j + BU_j + DU_j + F(t_{j+1}) \]  
(8)

where

\[ K = \frac{4}{k^2} M + \frac{2}{k} C + K \]  
(9)

\[ A = \frac{4}{k^2} M + \frac{2}{k} C \]

\[ B = \frac{4M}{k} + C \]

\[ D = M \]
In this case the equations for the variables at \( j+i \) are tightly coupled and a solution can be obtained by decomposing the system matrix \( \bar{K} \) into upper/lower triangular matrices. Here a major parallel computation step is to map the decomposition across parallel processors. Similar parallel computational issues occur if equations (8) are solved by the Newton Raphson method.

An alternate timewise integration of equations (1) can also be done through integration procedures other than the Newmark method with similar effects. For example consider the explicit central difference approximation for \( \dot{U} \) and \( U \) at time \( t_j \)

\[
\dot{U}_j = \frac{1}{2k} \left[ U_{j-1} - 2U_j + U_{j+1} \right] \quad (10)
\]

\[
\ddot{U}_j = \frac{1}{2k} \left[ -U_{j-1} + U_{j+1} \right]
\]

Writing equation (1) at \( t_j \) and using the above approximation (10) leads to

\[
\bar{K}U_{j+1} + G(U_j) = AU_j + BU_{j-1} + F(t_j) \quad (11)
\]

where

\[
\bar{K} = \frac{1}{k^2} [M] + \frac{1}{2k} [C]
\]

\[
A = \frac{2M}{k^2} - K
\]

\[
B = \frac{1}{2k} C - \frac{1}{k^2} M
\]

The solution of equation (11) proceeds forward in time with a single \( LDL^T \) decomposition of \( \bar{K} \) and require no iterations. Since the \( \bar{K} \) contains only the mass and damping matrices it may be sparse and the decomposition is often fairly simple; furthermore, if \( M \) and \( C \) are diagonal, the decomposition is trivial.
The implementation of the solution to equations (2)-(4) on a parallel computer raises certain issues on the inherent parallelism in the problem and the way in which calculations can be most effectively carried out in parallel. Iteration-type implicit methods such as equations (5-7) or explicit methods such as equations (10,11) lend themselves to assigning equations for a select set of nodal variables to specific processors. The solution procedure has thus been parallelized according to physical regions, and might be denoted "physically parallel". On the other hand, if the solution procedure is based on a Cholesky type decomposition, the solution is parallelized by assigning to each processor various mathematical steps such as matrix algebra. The solution has been parallelized mathematically and might be denoted as "algorithmically parallel". Thus the integration approach, system matrices, number of available processors, and other features may significantly affect the solution approach most appropriate for a problem and the attendant benefits of parallelism.

Each time integration method has certain inherent properties in such areas as accuracy, convergence and stability. Once an integration method has been selected for a crash dynamics analysis certain key steps in the method lend themselves to parallel computations; some of these include

- Explicit time integration
- Element matrix generation
- System matrix assembly
- System matrix decomposition

The studies reported herein have focused on testing the parallel benefits of these key steps to help develop an overall crash analysis strategy. The results in these studies are summarized in the following sections.
EXPLICIT TIME INTEGRATION

To investigate explicit time integration methods consider the nonlinear dynamic response of a simply supported shallow arch subjected to a step pulse. A parallel solution to the arch was carried out at Georgia Tech on the FLEX/32 (Fig.1) and similar results were obtained at General Motors on the INTEL iPSC hypercube (Figures 2,3).

The governing equations for the arch are shown in Figure 4 as

\[
EI \left( \frac{\partial^4 w}{\partial x^4} - \frac{d^4 w_0}{dx^4} \right) - N \frac{\partial^2 w}{\partial x^2} + \rho A \frac{\partial^2 w}{\partial t^2} = -p(x,t)
\]

(13)

where

\[
N = \frac{EA}{2} \left[ \left( \frac{\partial w}{\partial x} \right)^2 - \left( \frac{d w_0}{dx} \right)^2 \right]
\]

in which:

\( w(x,t) \) = transverse displacement of the middle surface of the arch

\( w_0(x) \) = initial shape of the arch

\( \rho \) = mass density of the arch material

\( A \) = cross-sectional area of the arch

\( p \) = distributed force per unit length

To solve equations (13) finite differences are used to approximate spatial derivatives and central differences are used for time integration similar to that discussed in equation (4). Since the mass matrix is diagonal the computation sequences were organized as
\[ w_i^v = \frac{1}{h} \left( w_{i-2} - 4w_{i-1} + 6w_i - 4w_{i+1} + w_{i+2} \right) \]  \hspace{1cm} (14) \\
\[ w_i^{ii} = \frac{1}{h^2} \left( w_{i-1} - 2w_i + w_{i+1} \right) \]  \hspace{1cm} i = 1, 2, 3, \ldots m \\
\[ w_i^i = \frac{1}{2h} \left( -w_{i-1} + w_{i+1} \right) \]

where the \( i \) subscripts refers to the deflection at the \( i^{th} \) node point and superscripts refer to derivatives with respect to \( x \). With equations (14) equations (13) then take the form of equation (10-12) (with \( C=0 \) and \( M \) diagonal).

\[ U_j + KU_j + G(U_j) = F(t_j) \]  \hspace{1cm} (15) \\
\[ U_{j+1} = U_j^2 + 2U_j - U_{j-1} \]  \hspace{1cm} (16)

Here \( U_j \) represents the vector of \( w \) at the \( j \) the time step. If \( m \) is a multiple of number of the processors \( n \), then equations (15) and (16) can be mapped on a parallel computer with computation distributed equally across processors. If \( m \) is not a multiple, some imbalance in processor workload will occur. An executive may be designed in some cases to partition subordinate tasks among processors, help provide a work-leveling approach and minimize idle processors and attendant overhead.

Integration of non-linear structural dynamic equations by the central difference method (equations (15) and (16)) provides a direct approach for calculation the solution at the \( j+1 \) time step from information at the \( j \) time step. Figures 5, 6 illustrate, for example, how \( m \) dynamic equations of motion
are mapped onto m processors (i.e., m=n) for both the FLEX/32 and the INTEL iPSC. In this implementation each processor performs essentially the same task and the integration computations are interrupted once at the end of each time step for communication, as shown in Fig 5,6. Some minor imbalance in computation may occur for equations near the boundary due to minor differences in the equations there. This approach was implemented in concurrent FORTRAN programming language on the FLEX/32 using the shared memory common bus communication logic (Figure 7). It was also implemented at General Motors on the INTEL iPSC hypercube using the interprocessor communication logic (Figure 8).

Figure 9 shows typical results for the nonlinear dynamic response of a specific arch subjected to a step pulse. Figures 10 and 11 show results of computation speedup versus number of processors on the FLEX/32 and Intel iPSC, respectively. Speedup is defined as the time taken to solve a problem on one processor divided by the time to solve the same problem on n processors. Initialization and interprocessor I/O are not included for one processor but are included for n processors. Thus, the speedup on one processor is one, and the theoretical maximum speedup (with no overhead) on n processors is n.

Figure 10 shows that the computation speedup for the FLEX/32 implementation as the number of degrees of freedom (D.O.F.) increases. This result was obtained using the shared memory communication. Similar results for the iPSC are shown in Figure 11. The somewhat better efficiency of the iPSC on the FLEX/32 for the arch problem is due to the local communication nature of the problem which well suited for the iPSC architecture. Similar results for the arch were reported in reference 7 using the locally connected Finite Element Machine (FEM). In reference 7 the speedups approached the ideal values because the communication speed of the FEM was much faster than the processor computation speed. For example, Figure 11 shows that for 120 D.O.F. per
processor a significant improvement in computation speedup occurs and the processor efficiency approaches 100% (Figure 12) for eight processors. On the other hand, with 60 D.O.F. per processor, the efficiency drops off since less computations are required on each processor. Furthermore for 120 D.O.F. the overhead factors such as communication, lack of synchronization, unequal distribution of computation tasks are relatively small. The 99.1% efficiency achieved with eight processors provides a good indication of the potential benefits that can be expected for moderate numbers of processors. The basic results also indicate that the benefits of parallel calculations drop off for those problems which required less extensive calculations and more intensive data communications. A comparison of the FLEX/32 and iPSC results illustrates that a local connection computer such as the hypercube architecture may be more efficient than a shared data base computer like the FLEX/32 for small physically parallel problems such as the arch.
SYSTEM MATRIX DECOMPOSITION

When the time integration procedure uses a direct approach such as that shown in equation (8) a key computationally intensive step is the decomposition of the symmetric system matrix. To investigate the use of parallel processors for this task, a parallel Cholesky decomposition method was developed, implemented on the FLEX/32 and tested on representative matrices with varying band widths. The specific decomposition investigated was

\[ K = LDL^T \]  

(17)

where \( K \) is the system matrix, \( L \) a lower unit triangular matrix, and \( D \) a diagonal matrix

The elements of \( L \) and \( D \) are given by

\[ l_{ij} = \frac{1}{d_{jj}} \left[ a_{ij} - \sum_{k=1}^{j-1} d_{kk} l_{ik} l_{jk} \right] \quad j<i \]

(18)

\[ d_{ii} = a_{ii} - \sum_{k=1}^{i-1} d_{kk} l_{ik}^2 \]

The parallel approach used is based on assigning to each processor the row calculation steps of LD and synchronizing the specific matrix element calculations as required (see for example reference 19).

The results for computation speedup as a function of bandwidth are shown in Figure 13. The results indicated that for half bandwidths above about 50 the speedup with a moderate number of parallel processors is quite good and the average processor idle time is small.
ELEMENT MATRIX GENERATION/ASSEMBLY

A key step in all methods is the generation of element mass and stiffness matrices and the assembly of these matrices into system matrices. The generation task involves many decoupled calculations that are clearly well-suited for parallel computations. The assembly process for the system matrices can be done through row or column assemblies of appropriate element contributions. Some sample investigations were carried out on the FLEX/32 with two different types of elements, one more complex than the other. The generation of element matrices were distributed across available processors. The resulting computation speedups are given in Figure 14 and indicate the expected good performance of a parallel computer in this area. The generation of elements is a very fertile area for benefiting from parallelism in that element computations are largely independent. They do produce large quantities of data and the FLEX/32 shared memory was useful to provide a common data location preparation to assembly.

After the element stiffness matrices were generated, they were assembled into a master system matrix on a row by row basis. The speedup results for the generation and assembly process are given in Figure 14. It is clear from Figure 14 that the generation of elements benefits more from parallelism than does the assembly process. The FLEX/32 shared memory was also especially helpful in managing the large quantity of data associated with the element generation and assembly process.
CONCLUDING REMARKS

A study has been conducted of the potential of parallel computers to improve the computation capabilities for crash analysis of automotive vehicles. The numerical methods have been investigated on test problems with a view toward the steps which are computationally intensive and how these steps lend themselves to parallelism. Key computation steps investigated include matrix generation, assembly and decomposition. Results have also been obtained for an explicit time integration method where the mass matrix is diagonal.

The results indicate that the key computation steps in crash analysis can benefit significantly from parallel computers. They also indicate that there are minimum thresholds in computation tasks for multiple processors to be effective. Below these thresholds data communication becomes a bottleneck and parallel computation efficiency significantly deteriorates. The results suggest that with a relative large number of system degree of freedom and a moderate number of processors, good processor efficiency can be obtained. For example, with one processor per 100 degrees of freedom, parallel computer efficiencies may range from 50% to 90% dependent on the problem. This suggests that with 5000 degree of freedom system and with 50 processors, one might expect computation speedup for effectively coded crash dynamics algorithms to range between 25 and 50, a significant improvement over a sequential computer solution.

These initial results are encouraging and suggest that more detailed studies should be carried out on realistic problems with all major computation steps being implemented in parallel. Two alternate strategies should be considered, one based on an explicit time integration approach which exhibits "physical parallelism" and the second based on an implicit time integration approach which exhibits "algorithmic parallelism."
References


Terminal and locks

P/Mf-2 9

P/Mh 10

P/Mh

Shared Memory 2.25 Mb

Control

Common Bus 20.5 Mb/sec

Arbitration and locks

optional peripherals on any processor

Terminal

Disk

Tape

Printer

P/M exchangeable processor or 4 Mb memory cards
local memory/processor: 1 or 4 Mb

Figure 1. FLEX/32 Multicomputer
MIMD Machine
- $2^n$ processors with independent memory

Figure 2. iPSC Hypercube Architecture
Figure 3. Ring Computation Sequences for Intel iPSC
Figure 4. Geometry of a Shallow Arch
Figure 5. FLEX/32 Integration Approach
**Figure 6. Intel iPSC Integration Approach**

1. **Initialization**
2. **Send Input Message to Nodes**
3. **Receive Message from Nodes**
4. **Start Calculations**
5. **Communication**
6. **Print Out?**
   - **yes** → **Send Message to Host**
   - **no** → **Print Out**
7. **Receive Input from Host**
8. **Next Time Step**

- **Host**
- **Node 1, 2, 3, ..., n**
FLEX/32:

- Shared Memory ($Y(i), i = 1, n$)
- Common Bus
- Communication

Processor: (1) (2) ... (n)

Figure 7. FLEX/32 Communication Sequence

HYPERCUBE:

- Communication

Processor: (1) (2) (3) ... (n)

Figure 8. Intel iPSC Communication Sequence
Figure 9. Nonlinear Dynamic Response of Shallow Arch
Figure 10. Speedup for Shallow Arch
Speedup = \frac{(Time)_{1 \text{ processor}}}{(Time)_{n \text{ processors}}}

Figure II. Speedup for Shallow Arch (Intel iPSC)
Figure 14. Generation and Assembly of Truss Elements
Figure 12. Processor Efficiency (Intel iPSC)
Figure 13. LDLT Decomposition (Banded Matrix)
FLEX/32
COMPUTATIONAL CRASH DYNAMICS METHODS FOR
FIFTH GENERATION SUPERCOMPUTERS
COMPUTATIONAL CRASH DYNAMICS METHODS FOR FIFTH GENERATION SUPERCOMPUTERS

February 5th, 1988

Robert E. Fulton
Kuo-Ning Chiang

Georgia Institute of Technology
CAE/CAD Laboratory
Atlanta, GA 30332-0140
Abstract

In the area of crash impact, research is critically needed on the development and evaluation of parallel methods for crash dynamic analysis of complex nonlinear finite element and/or finite difference structure problems. The development of general-purpose finite element structure analysis computer programs have provided the capability to address a wide range of structures problems. However, these software systems are severely limited for nonlinear transient calculations because of the available speed on current sequential computers.

Projected advances in computer technology indicate that significant increases in effective calculation speed will available in the 1990's, through fifth generation supercomputer architectures consisting of arrays of processors operating in parallel on different tasks (see e.g., ref. 3 for a survey). Such advanced supercomputers, denoted as MIMD (multiple instruction, multiple data) computers, have the potential for increasing effective calculation speeds by several orders of magnitude. But this potential increase in speed can not be effectively utilized without the development and implementation of appropriate numerical algorithms for structures which take advantage of the parallel computation features of this new generation of computers. Use of existing conventional algorithms and software will not realize the full potential of these new MIMD computers, and research is needed in the development of parallel structure analysis/design algorithms for these computers (ref. 4-19). This research has been to develop and evaluate parallel methods for crash dynamic analyses of complex nonlinear finite element and finite difference structure problems.
Table of Contents

0. Abstract

1. Introduction .................................................................................................................. 1

2. Literature Survey ......................................................................................................... 3
   • Parallel Computers ..................................................................................................... 3
   • FLEX/32 and Hypercube iPSC Architecture ............................................................ 3
   • Parallel Algorithms .................................................................................................... 5
   • Crash Dynamics ........................................................................................................ 7

3. Proposed Research ..................................................................................................... 9

4. Basic Theory ................................................................................................................ 10
   • Explicit Time Integration Scheme ............................................................................. 10
   • Implicit Time Integration Scheme ............................................................................ 11

5. The FENRIS Cholesky Decomposition .................................................................... 15

6. Issues with Shared memory and Hypercube Implementation .................................. 17

7. Evaluation of FENRIS/ANSYS for Parallel Implementation .................................... 19
   • Wave Front and Cholesky Equation Solver ............................................................. 19
   • Portability and Mantainability ................................................................................. 20
   • Evaluation of FENRIS for Parallel Implementation ............................................... 21

8. Summary ..................................................................................................................... 24

   • Reference .................................................................................................................. 26
   • APPENDIX A - FENRIS .......................................................................................... 30
   • APPENDIX B - ANSYS ............................................................................................ 35
1. Introduction

Equations of large-order crash dynamic problems are often intractable on current sequential computers due to memory and execution time limitations (ref. 3). In the area of crash impact, research is critically needed on the development and evaluation of suitable parallel methods for nonlinear finite element structural programs. Multiple Instruction Multiple Data (MIMD) computers provides an opportunity for significant gains in computing speed (ref. 21, 24 and 26), and can make tractable the solution to large-order problems.

Recent advances in computer hardware technology have resulted in the introduction of a variety of innovative computer architectures, including vector and parallel processors. The computational power of these new computer systems offers the structural dynamics analyses the opportunity to solve refined models of complex engineering problems and reduce computation times. To exploit this opportunity, concurrent methods for solution to structural dynamic problems need to be investigated. New algorithms, programming rules, and operating systems are needed to utilize the full potential of these new systems. Existing sequential algorithm need to be modified or replaced by algorithms which take advantage of the power of the new systems.

Parallel computers offer the promise for solution to complex problems in detail considered too time-consuming for modern sequential processors. To benefit from advances in parallel computers, software must be developed to encourage and support the migration of programs to this new class of computers. Parallel processing implies two or more processes executing simultaneously on separate computers. Often one process depends on information calculated by other processors. Programming for this environment introduces increased complexity in the software design and algorithmic strategy.
In adapting a solution to parallel processing implementation, one looks for those tasks that can be carried out concurrently with a minimum exchange of information or data dependencies. The algorithm must consider such things as: communication overhead, parallel I/O, memory contention, critical regions of the code that must be executed sequentially, the time to initiate tasks, data interdependency, and idle time resulting from an imbalance of the work load. The implementation reported on this study addresses the above considerations with respect to the solution procedure, portability and maintainability.

Research has been continuing on methods developments and to incorporate a parallel approach into a commercial finite element system. As a result of the study Georgia Tech obtained for the study a reduced ANSYS system from Swanson Analysis and a complete FENRIS system from Veritas Research in Norway. Cooperative agreements have been achieved with both companies so that either ANSYS or FENRIS can be used in the crash dynamics study.
2. Literature Survey

- Parallel Computers

The impact of supercomputers, and highly parallel processing systems on finite element computation is profound. This topic has been reviewed in detail by Noor, Storaasli, and Fulton (ref. 3). Parallel computers can be classified as either SIMD (single instruction multiple data) or MIMD (multiple instruction multiple data) architectures (Figure 3). In the class of SIMD are such machines as the CDC STAR-100, CYBER 203, CYBER 205 and CRAY-1; these provide high computation speed, but are generally not well suited for large-scale finite element computations. Complex structural dynamics problems will require effective computer speeds much greater than $10^3$ MFLOPS (million floating point operations per second) for timely results (ref. 3 and Figure 4), and these speeds are well beyond the capabilities of current SIMD computers. A more effective way of increasing computation speeds is through the use of multiprocessor computers (ref. 4 - 19, 59, 60). MIMD computers can be classified by their memory arrangement and the means by which each processor can access data stored in memory. In this classification there are two types of architectures; (1) shared and/or distributed memory architectures such as the CRAY X-MP, CRAY-2, ETA-10, ALLIANT FX/8 and FLEX/32, and (2) network or local memory message based architectures such as the FPS T-series, iPSC INTEL, N-CUBE, AMETEK, and Caltech MARK 111 Hypercube. Representative of shared and local memory architectures are described in the following section.

- The FLEX/32 and iPSC Hypercube Architecture

The FLEX/32 shared memory multicomputer (ref. 61) currently installed at Georgia Tech CAE/CAD Laboratory is composed of 8 processors (can be extended to
20 processors). Each processor has 4 Mbyte of local memory, and all processors can access 2.25 Mbyte of shared memory through a common bus. Figure 1 illustrates the configuration of the 20 processor configuration. Pairs of processors reside on each of 10 local buses, and all 20 machines must access the shared memory by going through the local then the common bus. The processors are numbered as 1, 2, 3, 4, 11, 12, 13, 14 at Georgia Tech. Processor one operates under UNIX operation system and is used for software development and for loading and booting up the remaining processors when a parallel program is to be run. Thus, under normal operation, 7 processors are available for parallel processing and run under the MMOS concurrent operating system. MMOS was designed to fully support multicomputing, multitasking, and real-time programming on the FLEX/32 multicomputer. It is a real-time operating system that contains the facilities (utilities and library routines) to compile, assemble, link/load, and run concurrent and sequential programs.

The Intel iPSC Hypercube multiprocessor (ref. 40) consists of \( p = 2^D \) (Figure 2) independent processors or nodes where \( D \) is called the dimension of the Hypercube. The machine comes in three models - the D5, D6 and D7. These have respectively 32, 64 and 128 processors. Each processor is a sequential computer processing its own local memory. There is no shared memory available and the processors cooperate by message passing. In addition to the \( p \) nodes there must be one additional processor called the host. The host processor should have a relatively large amount of Random Access Memory (RAM) in comparison to the smaller amount of RAM available on the individual hypercube processors. The job of the host is to spawn processes on the nodes, to collect information from them and to handle I/O.

The nodes are identified with the binary representations of the numbers 0 through \( p-1 \). Two processors will be called "nearest neighbors" only if the binary representation
of their number differ by one bit. For example, when \( p = 8 \) (Figure 2), node number 7 (111) is directly connected to node number 3 (011) or 6 (110) but not to node number 1 (001) or 2 (010). The algorithms discussed in this study have been implemented on an Intel iPSC/d5 machine which is a 32 processors hypercube at the General Motors research Laboratories. The individual processors are the Intel 80286/80287 with up to 512Kb of RAM, The host processor is the Intel 286/310 microcomputer with 2 Mb of RAM and the interconnections are provided by high-speed Ethernets, using the Intel Ethernet chip.

- **Parallel Algorithms**

  In adapting a solution to parallel processing implementation, efficient algorithms typically maximize parallel and minimize sequential calculation. The algorithm must consider such things as: efficient interaction of vector and parallel computations, communication/waiting overhead, memory contention, critical regions of the code that must be executed sequentially, data interdependency, idle time resulting from imbalance of the workload, portability and reliability. In general, for concurrent transient finite element systems two alternate strategies should be considered, one based on an explicit method and the other based on an implicit method. An obvious way to carry out a concurrent execution of explicit finite element method using \( n \) processors is to decompose the structural domain into \( n \) regions and give each processor “responsibility” for solving an entire subproblem. A substructuring method for implementation on multiprocessor computers has been recently proposed by Storaasli and Bergan (ref. 31). An automated mesh decomposition and concurrent finite element analysis has been discussed by Malone (ref. 26). His approach used a central difference time integration scheme and selected test problems were implemented on Hypercube multiprocessor computers. The formulation includes a new decomposition algorithm which automatically divides an arbitrary finite element mesh into region and assigns each region to a processor on the Hypercube, The algorithm has been implemented on a 32 processor
Hypercube for plane-stress analysis and exhibits more than 95% efficient use of machine. The Newmark trapezoidal integration approach has been carried out by Storaasli, Ransom and Fulton (ref. 6), and their results for a two dimensional beam grillage problem gave computation speedups reaching a value of 6.5 for eight processors.

When the time integration procedure uses a direct approach, a key computationally intensive step is the decomposition of the symmetric/unsymmetric system matrix. The Cholesky decomposition (a scheme widely used in the existing finite element equation solver) has been studied by Goehlich, Komzsik and Fulton (ref. 21). Their parallel approach was incorporated in MSC/Nastran and tested for several static stress analysis demonstration problems on CRAY X-MP and IBM 3090 computers with up to four processors. The results indicated that a parallel processing approach can significantly reduce execution time for large scale finite element problems. A parallel Cholesky scheme is also discussed by Bostic and Fulton (ref. 58) in their Lanczos method implementations. In the Lanczos method the decomposition step is the most time-consuming calculation step for large problems and this step benefited the most from the parallel procedure. A speedup of 7.8 on eight processors was obtained for the decomposition step of the mast problem on the FLEX/32 Multicomputer.

It is well known that local memory machines have difficulty in achieving a good speedup for algorithms such as the Cholesky decomposition. The sequential Frontal method was first described by Irons (ref. 37) to solve finite element problems, and is based on the observation that Gaussian elimination can begin before assembly of the K matrix is completed. In fact a node in the finite element mesh can be eliminated as soon as all the elements associated with it are assembled. A parallel multifrontal equation solver was investigated by Geist (ref. 22) using a local memory multiprocessor (iPSC Hypercube). The approach is to have one front on each processor and is designed so that
each processor solves an entire subproblem. While the limited results are encouraging it requires regeneration of the K matrix in every iteration, a strategy not well suited for a general purpose nonlinear finite element analysis.

An alternative approach is the conjugate gradient iteration equation solver, a method studied by Lyzenga, et al (ref. 29), and Nour-Omid and Parks (ref. 56) on a 32 processor Caltech Hypercube. It was used successfully for a large number of finite element problems. The conjugate gradient algorithm is well-suited for vector computation but, because of its many synchronization points, it more difficult for parallel computation. A block conjugate gradient algorithm has been discussed by O'Leary (ref. 53), and the parallel efficiency of his algorithm exceeded that of the standard conjugate gradient algorithm. The block preconditioned conjugate gradient method has also been investigated by Meurant (ref. 56) on a four processors computer CRAY X-MP/48, where the numerical results exhibit good parallel and vector efficiency. Other studies of parallel equation solvers have focused on narrow banded systems using such methods as cyclic reduction, recursive doubling, divide and conquer and twisted factorization methods (ref. 42, 54 and 55). The results show a good speedup potential in both parallel and vector computations.

• Crash Dynamics

The associated tasks for crashworthiness are: prediction of the crashworthiness of a specific structure, improvement in design, and possibly optimization of the structure with respect to crashworthiness. In the past, the evaluation of crash performance was dependant on the impact tests, but such tests are costly and require a long preparation time (ref. 23, 39). Therefore, numerical studies of crash analyses are becoming increasingly important in this area. For a complicated crash simulation, the amount of data required to describe the problem is enormous, and results in extremely large computational costs. One of the first finite element programs to take full advantage of vector
processes, such as the CRAY-1, is DYNA3D (ref. 52, 57). The DYNA3D system has been reported by Benson, Hallquist and Stillman (ref. 57). One of the test problem shown at Figure 5, is a curved S-Frame impacted by a large mass at 30 km/hour. The algorithm used for the dynamic analysis was the explicit central difference method. The finite element model consists of 1600 shell elements with five integration points through the thickness. The computation for 35 ms of real time required nearly 4.2 CPU hours on the CRAY-1.

Computer simulation of crash phenomena has also been studied by Argyris, Balmer and Kurz (ref. 30) using the S-Frame test problem (see Figure 6). Their finite element model of the S-Frame is composed of 388 TRUMP element with 1200 unknowns, and the dynamic computation was performed via a matrix solution of Newmark’s method (ref. 20). The time increment was 0.0005 sec and the number of iterations per time step was limited to 5. Computation up to 45 ms (90 time intervals) required 1.5 CPU hours on the CRAY-1M (the estimated CPU-time on VAX-11/780 was 300 hours).

The results obtained do not adequately fit the experimental result, and a refined mesh and smaller time step is required to obtain better results. The results discussed here show that today’s supercomputers are marginal at best for sophisticated crashworthiness calculation. This implies that the solution algorithm must not only be inherently efficient, but also that it must take advantage of both vector and parallel computers so that all of a supercomputer’s potential is realized. In the area of crash impact, research is critically needed on the development and evaluation of vector and parallel methods for crash dynamic analysis of complex nonlinear transient finite element problems.
3. Proposed Research

The research reported herein is to investigate the potential of parallel computers to significantly improve the capability for crash analysis of automotive vehicles. Task planned are:

(1) the development of concurrent algorithms for nonlinear transient analysis which make effective use of multiprocessor computing power. These algorithms include (1) implicit methods such as Newmark type which build on Cholesky decomposition strategy and exhibits "algorithmic parallelism", and (2) explicit method such as the central difference time integration scheme which exhibits "physical parallelism"

(2) An assessment of the potential of the shared memory and local memory computers for use in parallel computations

(3) Implement the FENRIS nonlinear finite element commercial program on the FLEX/32 MMOS (Multicomputing Multitasking Operating System) for parallel solution

(4) Incorporate a parallel Cholesky decomposition method in FENRIS and evaluate the performance of the resulting parallel nonlinear finite element system

(5) Investigate the implementation and performance of a parallel force vector generation in FENRIS.

(6) Evaluation the methods with the GM S-Frame test problem.

Work has progressed well toward the research goals, some specific comments relative to the above six goals will be discussed on the following sections.
4. Basic Theory

There are two basic integration approaches used in the transient finite element systems: (1) explicit method and (2) implicit method. In an implicit formulation, the temporal difference equations such as velocities and accelerations are combined with the equation of motion so that the displacements at a new time step can be calculated directly. For nonlinear problems, however, some iteration approaches like the Newton Raphson method must also be used. In an explicit formulation the response quantities at the new time step are expressed in terms of previously determinated values of displacement, velocity and require no iterations.

- Explicit Time Integration Scheme

In general the finite element equations governing the dynamic response of a structure may take the form

\[ R_{int} = R^E - R^D - R^I \]  \hspace{1cm} (1)

where \( R^E, R_{int}, R^D \) and \( R^I \) denote external forces, internal forces, damping force, inertia force and can be expressed as

\[
\begin{align*}
R^D &= C \ddot{u} \\
R_{int} &= K \Delta u \\
R^I &= M \dddot{u}
\end{align*}
\]  \hspace{1cm} (2)

where C is the damping matrix (Appendix A), K is the stiffness matrix and M is the mass matrix.

The explicit central difference approximation for \( u \) and \( \dot{u} \) from time step \( j \) to \( j+1 \) over the time interval \( h \) is
\[ \dot{u}_{j+1/2} = \dot{u}_{j+1/2} + h \ddot{u}_j \]
\[ \ddot{u}_{j+1} = \ddot{u}_{j+1} + 2h \dddot{u}_j \]
\[ u_{j+1} = u_j + h \dot{u}_{j+1/2} \]  

Rewriting equation (1) and using the above approximations leads to

\[ M\ddot{u}_{j+1} = R_{j+1}^{E} - C\dot{u}_{j+1} - K\Delta u_{j+1} \]  

(4)

For both explicit or implicit time integration schemes, the residual force \( \hat{Q} \) vector must vanish; thus

\[ \hat{Q} = R_{j+1}^{E} - M\ddot{u}_{j+1} - C\dot{u}_{j+1} - K\Delta u_{j+1} = 0 \]  

(5)

The explicit time integration scheme automatically satisfies equation (5) and require no iteration; if \( M \) is lumped, the decomposition is trivial.

Parallel flow chart for explicit finite element program is shown at Figure 7. Each processor computes all assigned element quantities such as \( C_{\text{sub}}, K_{\text{sub}} \) matrix, stresses, strains and accelerations without data exchanging and these quantities are stored in the local memory. However, the lumped mass matrix, nonlinear force vectors \( \hat{F}_{\text{eq}} \), displacement and velocity vectors are put into global memory for use by more than one processor. From transient finite element system points of view, parallel I/O, less memory contention and high parallel computation efficiency make explicit methods well suited for parallel computations.

- Implicit Time Integration Scheme

In this algorithm the discussion will focus on parallel matrix decomposition, and
how to use local memory as a secondary memory storage. A general incremental-iterative scheme combine with the Newton Raphson family and Newmark method are used for solving the dynamic equations and the final equation can be expressed as (more detail can see Appendix A)

\[ \hat{K} \Delta \hat{u}_{i+1} = \Delta \hat{R}_{i+1} \]  

(6)

The effective stiffness matrix \( \hat{K} \) and the incremental residual vector \( \Delta \hat{R}_{i+1} \) are

\[ \hat{K} = a_0 M + c_0 C_v + K_T \]

\[ \Delta \hat{R}_{i+1} = R_{t+1}^E - R_{\text{int},t} + M[\hat{a}_t + \hat{b}_t(\alpha_1 - \alpha_2) - \hat{u}_t] + C_v \hat{b}_t(1 - \alpha_2) \]

The corresponding increments of the velocities and accelerations are

\[ \Delta \hat{u}_{i+1} = f_1(\Delta u_{i+1}, \hat{u}_i, \hat{u}_i) \]  

(7)

\[ \Delta \hat{u}_{i+1} = f_2(\Delta u_{i+1}, \hat{u}_i, \hat{u}_i) \]  

(8)

If the vectors \( u_{i+1}, \hat{u}_{i+1} \) and \( \dot{u}_{i+1} \) are substituted into the dynamic equilibrium equation (1), it will be found for nonlinear cases that this equation is not completely satisfied, and equilibrium iterations must be carried out.

In the implicit time integration scheme the nonlinear force vectors are generated by "physical/substructural parallelism"; however, the Cholesky decomposition is solved by "algorithmic parallelism". The major time-consuming steps of this algorithm are (1) nonlinear force vectors generation, (2) input/output, and (3) matrix decomposition. The mass matrix, stiffness matrix, viscous damping matrix, internal force vectors can be computed element by element and assembled into the global matrices \( M, K, \)
These computational tasks are independent and highly parallelizable and can be assigned across processors. The Cholesky decomposition step is the most time-consuming calculation for large size finite element problems, and this method has shown good speedups through parallel and vector processing, especially when the half-bandwidth is sufficiently large (ref. 21, 25).

For a large structural systems the effective stiffness matrix cannot be accommodated in core storage (shared memory), and must be segmented into blocks (ref. 27). These blocks must then be stored temporary on low speed auxiliary storage, usually as disk files, where the CPU time is relatively high. Shared/Local memory can take advantage of the local memory available on each processor, Data blocks are stored in local core memory by use of standard FORTRAN 77 calculations such as Local Variables 1 to n equal to Share Variables 1 to n, these data mapping CPU times are much less than the standard I/O operation. Therefore, the use of local memory as a secondary storage unit is needed for parallel finite element implementations.

The equation (6) may take the form as \( A \mathbf{X} = \mathbf{B} \), where the effective stiffness matrix \( A = LU = L D^t L^t \). Here the matrix \( L \) is a lower unit triangular matrix and \( D \) is a diagonal matrix, the decomposition is carried out in three step:

\[
\begin{align*}
l_{ij} &= a_{ij} - \sum_{n=1}^{i-1} l_{ni} l_{nj} & i < j \\
l_{ij} &= l_{ij} d_{ij}^{-1} \\
d_{ij}^{-1} &= 1 / \left( a_{ij} - \sum_{n=1}^{j-1} l_{nj} l_{nj} d_{nn} \right) & \text{for all } j
\end{align*}
\]

The matrix \( L \) has the same shape as \( A \) (skyline form) and is therefore stored in the same location. The diagonal matrix \( D^t \) is stored in an array residing in core. The stiffness
matrix $A$ is divided into several blocks, each consisting of several columns and $A$ is
triangulized block by block. Some typical results of this algorithm are shown on the
following section.
5. The FENRIS Cholesky Decomposition

Work has progress well toward the research goals. A significant effort has been completed to evaluate, select and obtain a software testbed for parallel implementation of crash dynamics analyses. Criteria for selecting the system include the following: commercial quality finite element system, nonlinear material and geometry, dynamic loads, FORTRAN, suitably documented and available source code. A number of candidate systems were investigated including NASTRAN, ANSYS, ADINA, FENRIS, DYNA-3D, DYCAST, and SPAR. The availability of both the technical capability and also the source code was difficult to achieve. As a result of the study Georgia Tech obtained a reduced ANSYS system from Swanson Analysis and a complete FENRIS system from Veritas Research in NORWAY. Cooperative agreements have been achieved with both companies so that either ANSYS or FENRIS can be used in the crash dynamics study.

Research has continued on methods development and a parallel FENRIS LDL$^T$ method was developed. Speedups of this parallel Cholesky decomposition (compare with the sequential FENRIS Cholesky Decomposition) are very encouraging. The improvements are achieved by refined parallel computation strategies, and a highly machine independent waiting control routines were used which reduce wait time in the matrix decomposition. A selected static test problem is shown at Figure 8, which is a cantilever beam subjected to a concentrated force. This finite element model comprises 100 quadrilateral elements with 444 unknowns and the maximum half-bandwidth equal to 48. Typical results showing the improved computation speedups are given in Figure 9. The results show computation speedups of up to 6.54 for seven processors and indicate that significant speedups can be achieved through the use of many processors for an appropriate finite element problem. It should be noted that the Cholesky decomposition
is also well-suited for vector computation (Figure 10). In the parallel Cholesky decomposition, each processor is responsible for decomposing the whole column of the system matrix; therefore no vector length penalty is introduced. From above observations the $LDL^T$ algorithm shows a good promise for parallel/vector computation and this algorithm will be the baseline method for the crash dynamic studies.
6. Issues with the Shared Memory and Hypercube Implementations

These studies reported herein investigate the use of the FLEX/32 shared memory and the iPSC Hypercube architectures for similar structural calculations; some observations on the relative merits are useful. As discussed earlier the implementations illustrate how both shared and local memory machines can give good parallel efficiency on problems which have physical characteristics oriented to parallel computations. In the local memory machine, however the communication distance between processors exchanging data is determined by the choice of processor mapping. Thus the major mapping issues are (1) to make data passing between processors possible through a shortest communication path, and (2) to minimize the number of other processors with which each processor must communicate. For the shallow arch test problem this mapping is very easy since the processors can be arranged from the left hand boundary (Figure 11) to the right one. In such a case each interprocessor is connected by a direct communication channel and the last processor does not have to communicate with the first one. However, for some structures such as closed arch or ring problems subjected to unsymmetrical loads, this processors mapping topology would have problems since the last processor must communicate with the first processor. For the iPSC Hypercube the data passing distance for the ring would require twice the time needed for the shallow arch. Optimizing processors mapping and data passing length in a local memory machine is therefore a highly machine dependent task and different local memory machines have different communication efficiency; thus software portability and reliability are major issues for local memory machines.

For shared memory machines one can take advantage of global memory where any processor can access this memory. Communication subroutine can be written in a high level language or transferred to assembler language for better efficiency. Some approach
such as a shared lock flag must be established to provide controlled access to the shared memory. For example, if the lock flag is on, the parallel task is placed in a waiting mode with no data exchange. When the lock flag is off, the parallel task is removed from waiting and global data can be accessed by each processor. Since the flag concept can be coded by high level language, it can be made machine independent. Thus parallel speedups will be consistent across different shared memory computers and there is little difficulty in migrating code to different shared memory computers, such as from the FLEX/32 to the CRAY X-MP/48.

A second strategy is associated with "algorithmic parallelism" such as in the case of the Cholesky decomposition (a scheme widely used in the existing finite element equation solver). Here the matrix diagonal terms are required by each processor for calculating $L^T$. In the local memory machines these diagonal terms have to be broadcasted to each processor, which results in a sequential communication task. Since the data processing time is relatively large the algorithm did not show a good parallel efficiency on local memory machine. Another disadvantage of local memory machines for parallel finite element implementations is that the system matrices such as K, M, C are generated in blocks ("substructure"), whereas the parallel decomposition algorithm needs the information assigned to processors columnwise. Combination of these two computational steps in a local memory machine is a problem, whereas it is more easily dealt with for shared memory machines because the shared data base is readily available for all processing tasks.
7. Evaluation of FENRIS/ANSYS for Parallel Implementation

Figure 12, 13 show the solution flow charts of ANSYS/FENRIS for the Nonlinear Transient Dynamic Analysis. The major solution procedures include element matrix generation/assembly, effective stiffness matrix \((\hat{K})\) triangulization and nonlinear load vector \((\hat{R})\) calculation. The matrix generation and the nonlinear load vector calculation tasks involve many decoupled calculations that are clearly well suited for parallel computations on both codes (APPENDIX A and B). For nonlinear transient dynamics problems the ANSYS performs a wave front solution strategy, and the system matrix will be regenerated at every iteration. The equation solver adopted by FENRIS is the Cholesky decomposition, and there are four iterations strategies which can be used (see Figure 14). These are:

1. **Incrementation with unbalanced force correction**: no iteration will be performed within each incremental step and the effective stiffness matrix \(\hat{K}\) will be decomposed at every steps.

2. **Initial stiffness iteration**: \(\hat{K}\) matrix will not be updated and only decomposed once with the other steps using the backward/forward substitutions, this method is not well suited for parallel computations since the backward/forward substitutions is not efficiently carried out in parallel.

3. **Modified Newton-Raphson**: \(\hat{K}\) matrix will be updated and decomposed at each new incremental step.

4. **True Newton-Raphson**: \(\hat{K}\) matrix will be updated and decomposed at every iterations.

Two of these four methods, (1) Incrementation with unbalanced force correction and (4) true Newton Raphson show better potential for parallel implementations, however, the first method is not suitable for highly nonlinear problems.

- **Wave Frontal and Cholesky Equation Solvers**

  The ANSYS program uses the wave front or frontal equation solver first discussed
by Irons (ref.37) for the system of the simultaneous linear equations associated into the finite element method, the method is based on the observation that Gaussian elimination can begin before the assembly of effective stiffness matrix is completed. The number of equations which are active after any element has been processed during the solution procedure is called the wave front at that point. In this method calculations and I/O are tied together and the front width has to be kept as small as possible in order for frontal method to be efficient. Elements are sequentially added to the effective stiffness matrix resulting in a poor speedup for a parallel version of this algorithm.

The p-frontal (multifrontal) method has been investigated by Geist (ref.22) on the Intel Hypercube. The p-frontal approach is designed so that each processor solves an entire subproblem. The testbed routines read the reorganizing information from data file set up by the engineer. The automation of the partitioning and the reordering of the problem is still under investigation. However, it is not easy to incorporate a parallel approach into ANSYS, because several pre-tasks have to be done on each processor before p-front decomposition. Some of these include new node renumbering routine, sub-matrix (stiffness, mass, damping) generation, sub-nonlinear load vectors calculation, parallel I/O and data rearrange. In contrast, a parallel Cholesky decomposition can be tested independently in a program like FENRIS and does not require the reconstruction of the rest of the software. The speedup of the Cholesky method is good for seven processors when the half-bandwidth greater than 50 (ref.21, 25 and Figure 9). In light of the above discussion a Cholesky based finite element system such as FENRIS appears well suited for parallel implementations.

- **Portability and Maintainability**

  A key issue for a successful finite element system is its availability, portability, and maintainability on different computers. The major advantages of FENRIS are its por-
tability, building block concept, programming and documentation standards. It should be noted that Georgia Tech migrated the full FENRIS VAX Version to FLEX/32 UNIX and MMOS operating system without much difficulty. The building block concept adopted in FENRIS enables a person who is working on a special topic, e.g. on a new solution algorithm such as the Parallel Cholesky Decomposition or on a new constitutive law for a material, this is allow him to program and test his work without having to develop a whole new program. This possibility is especially important for a joint project like a Parallel FENRIS.

Swanson Analysis, Inc. has recently improved its ANSYS User’s Manual and Theoretical Manual, and enlarged its well-organized Verification Manual to more than 164 problems. These significant efforts are very helpful for the users who want to know how to use the ANSYS to model and solving their problems. However, for parallel implementations such as recoding or adding new subroutines (e.g. multifrontal algorithm) into ANSYS, more documentations are required than that provided in the source codes and manuals.

Both ANSYS and FENRIS are good commercial finite element system and have their own fans, but from the viewpoint of portability, maintainability, code documentation and algorithms the FENRIS appeared to have more promise as a testbed for the parallel implementations.

• Evaluation of FENRIS Performance for Parallel Implementation

The FENRIS system has been initially installed at Georgia Tech on VAX/750 VMS, FLEX/32 UNIX and MMOS operating systems; and initial results obtained for the S-Frame test problem have been achieved on both machines. The test finite element model comprises 180 triangular shell elements with 456 unknowns. The impact phe-
nomenon was computed on the basis of elastic-perfectly plastic constitutive material (A-36 Steel). Dynamic computations were performed via a matrix solution of Newmark's unconditional stable scheme, with a time increment equaled to 0.0005 second. A fine mesh or smaller time increment was not used because the limitation of CPU time and disk space on the VAX/750 and FLEX/32. Typical results are shown as Figure 15, 16; these results compare with favorably with similar results by Argyris, et al, (ref. 30) and indicate that FENRIS system provides an adequate basis for the parallel crash dynamics investigation.

The computation times attributed to both I/O and CPU for the S-Frame test problem are shown at Figure 17. This Figure illustrates that the percentage of computation associated with effective stiffness matrix, internal and external force vectors generation is 84.6 % whereas for equation solving is 11.90 %. However, for a larger models the percentage for equation solving will increase rapidly and it is expected that the equation solving will become the dominant time consuming phase for very large problem with large bandwidths. It is a well known that the Cholesky decomposition has difficulty obtaining an attractive speed up for a small half bandwidth system. For such a small finite element system like Figure 18, where the half bandwidth is 12 and the total D.O.F equal to 30, the percentage of CPU associated with equation solving is negligible (0.5 %). Most of the times are spent on effective stiffness matrix, nonlinear force vectors generation and I/O. These parts have been discussed at previous chapter and it is believed that high parallel efficiency can be achieved. For a large half bandwidth finite element problem such as a half bandwidth equal to 100, the equation solving might become the dominant CPU time-consuming, however, the Cholesky decomposition manifest a good speed up using a suitable amount of processors. From above investigations FENRIS appears well suited for parallel implementations.
8. Summary

This is to summarize the status of the work of this research. The main concerns of this research are to (1) develop a parallel finite element approach for crash dynamics analysis, (2) incorporate the parallel approach into a commercial finite element system, and (3) evaluate the performance of the methods with the GM S-shape test problem. In summary, work to date has progressed well on several fronts toward the research objectives. A baseline software system, good parallel crash dynamics methods, and validation results on the test problem have all been achieved. The initial results are encouraging and indicate that the key computation steps in crash analysis can benefit significantly from parallel computers. More detailed studies should be carried out on realistic problems with all major computation step being implemented in parallel.

Work is now proceeding to investigate a parallel element generation and nonlinear force vectors capability and evaluate the results on the S-Frame test problem. A recent meeting was held with CRAY Research representative who expressed interest in cooperative efforts to migrate FENRIS to the CRAY X-MP for parallel testing.
References


55. Delves, L. M. and Sanba, A. S., Band Matrices on the DAP, GR/B/24332, University of Liverpool and University of Birmingham.


APPENDIX

A. FENRIS - A General Purpose Nonlinear Finite Element Program

FENRIS is a large scale, general program for nonlinear finite element analysis. The program name is an abbreviation for Finite Element Nonlinear Integrated System. The development of FENRIS started in 1980 as a project between the Norwegian Institute of Technology (NTN), The Society for Industrial and Technical Research (SINTEF) and The Norwegian VERITAS. The main intention of the project has been to develop a new type of "truly nonlinear" program package. The program, as well as the technique of data storage, has been designed to correspond with the theoretical basis and major computational steps of nonlinear analyses. Another design concept for FENRIS has been to construct the program as a highly modular building kit. This has been done through isolating the various computational steps as much as possible from each other in the program software. This building block philosophy makes it easy to extend the program step by step and to readily accommodate new programming staff during development.

- Dynamic Algorithms

For dynamic problems, the body forces also contain the inertia force from D'Alembert's principle; the body force can be written as (Ref.32)

\[ R^E = R_{int} + R^D + R^I \]  \hspace{1cm} (A.1)

The subscript E, int, D and I denote external forces, internal forces, damping forces, and inertia forces, respectively. \( R^0 \) may be used on the form
where $C$ is a damping coefficient. The program also includes capabilities for other types of damping. $R'$ is obtained from the D'Alembert's principle as

$$R' = \rho \frac{d^2 u}{dt^2}$$  \hspace{1cm} (A.3)  

where $\rho$ is density. For time step $i$, eq.(A.1) may be written as

$$R_i^I + R_i^D + R_{int,i} = R_i^E$$  \hspace{1cm} (A.4)  

The internal reaction force vectors are computed element by element then transformed from the local co-rotated element coordinate system (ref.32) to the global reference system and assembled for all elements. The inertia force vector and the damping force vector are also in reality computed element-wise and assembled with use of the global matrices $M$ and $C$. The incremental form of the governing equation may be written as

$$M_i \Delta \ddot{u} + C_i \Delta \dot{u} + K_i \Delta u = R_i^E - (R_{i-1}^I + R_{i-1}^D + R_{int,i-1})$$  \hspace{1cm} (A.5)  

$M_i$ is the mass matrix, $C_i$ the incremental damping matrix and $K_i$ the incremental stiffness matrix. $K_i$ may be taken as the tangential element stiffness $K_T$, assembled to a global system matrix, or, the $K_i$ may be used for several consecutive steps. $\Delta u, \Delta \dot{u}, \Delta \ddot{u}$ are the incremental displacement, velocity, and acceleration vectors. The $\Delta$ denotes finite but "small" increments corresponding to the difference between the two states considered. Equation (A.5) is used in connection with load increments and equilibrium iterations.
The program assumes that the Rayleigh damping matrix (ref. 44) is constructed from the following form

\[ C = \alpha_1 M + \alpha_2 K_T + C_v \]  

(A.6)

where \( \alpha_1, \alpha_2 \) are damping factors and \( C_v \) is a diagonal matrix representing viscous dashpots prescribed at nodes.

**Implicit Time Integration Algorithm**

A general incremental-iterative scheme combined with the Newmark family of implicit time integration operators is used to solve the dynamic equations. The basic assumptions of the Newmark operators are (ref. 20)

\[ \dot{u}_{i+1} = \dot{u}_i + (1 - \gamma)\ddot{u}_i h + \gamma \ddot{u}_{i+1} h \]  

(A.7)

\[ u_{i+1} = u_i + \dot{u}_i h + (\frac{1}{2} - \beta)\ddot{u}_i h^2 + \beta \ddot{u}_{i+1} h^2 \]  

(A.8)

Here \( h \) is the incremental time step

\[ h = \Delta t = t_{i+1} - t_i \]

which may vary throughout the analysis. From equations (A.7) and (A.8) the corresponding increments of velocities and accelerations are

\[ \Delta \dot{u}_{i+1} = \frac{\gamma}{\beta h} \Delta u_{i+1} - \frac{\gamma}{\beta} \dot{u}_i - (\frac{\gamma}{2\beta} - 1)\ddot{u}_i \]  

(A.9)

\[ \Delta \ddot{u}_{i+1} = \frac{1}{\beta h^2} \Delta u_{i+1} - \frac{1}{\beta h} \dot{u}_i - \frac{1}{2\beta} \ddot{u}_i \]  

(A.10)
Substitution equations (A.9) and (A.10) into equation (A.5) yields

\[
\begin{bmatrix}
  M - \frac{1}{\beta h^2} + C \frac{\gamma}{\beta h} + K
\end{bmatrix}\Delta u_{l+1} = R_{r_{l+1}}^E - (R_{r_{l}}^l + R_{r_{l}}^D + R_{r_{l}}^{l+1})
\]

\[
+ M\left[\frac{1}{\beta h} \dot{u}_{l} + \frac{1}{2\beta} \ddot{u}_{l}\right] + C\left[\frac{\gamma}{\beta} \dot{u}_{l} + (\frac{\gamma}{2\beta} - 1)\ddot{u}_{l}\right]
\]

(A.11)

By combining equations (A.2), (A.3), (A.6) and (A.11) the incremental - corrective equations become

\[
\hat{K}\Delta \hat{u}_{l+1} = \Delta \hat{R}_{l+1}
\]

(A.12)

where the effective stiffness matrix \(\hat{K}\) is

\[
\hat{K} = a_o M + c_o C \gamma + K_T
\]

(A.13)

with the following constants

\[
a_o = \frac{1}{1 + \frac{\alpha_2\gamma}{\beta h}} \left(\frac{-1}{\beta h^2} + \frac{\alpha_1\gamma}{\beta h}\right)
\]

(A.14)

\[
c_o = \frac{1}{1 + \frac{\alpha_2\gamma}{\beta h}} \left(\frac{\gamma}{\beta h}\right)
\]

Here \(\alpha_1\) and \(\alpha_2\) are proportionality factors for the Rayleigh damping matrix (see equation A.6). Note that the effective stiffness depends on the time step \(h\). The increment of the effective load is

\[
\Delta \hat{R}_{l+1} = R_{r_{l+1}}^E - R_{r_{l}}^{l+1} + M[\hat{\alpha}_l + \hat{b}_l(\alpha_1 - a_o\alpha_2) \cdot \dot{u}_l]
\]

\[
+ C\gamma \hat{b}_l(1 - c_o\alpha_2)
\]

(A.15)
where

\[ \hat{b}_i = \left( \frac{\gamma}{\beta} - 1 \right) u_i + \left( \frac{\gamma}{2\beta} - 1 \right) \ddot{u}_i \]  \hspace{1cm} (A.16)

\[ \hat{a}_i = \frac{1}{\beta h} \dot{u}_i + \frac{1}{2\beta} \ddot{u}_i \]  \hspace{1cm} (A.17)

The set of equations is solved with respect to the effective displacement increment \( \Delta \hat{u}_{i+1} \). The increment of the real displacement vector is the

\[ \Delta u_{i+1} = \frac{1}{\alpha \gamma} \left( \Delta \hat{u}_{i+1} + \alpha \hat{b}_i \right) \]  \hspace{1cm} (A.18)

and \( \Delta \dot{u}_{i+1}, \Delta \ddot{u}_{i+1} \) can be obtained from equation (A.9), (A.10). The corresponding total vectors are

\[ u_{i+1} = u_i + \Delta u_{i+1} \]

\[ \dot{u}_{i+1} = \dot{u}_i + \Delta \dot{u}_{i+1} \]  \hspace{1cm} (A.19)

\[ \ddot{u}_{i+1} = \ddot{u}_i + \Delta \ddot{u}_{i+1} \]
B. Nonlinear Analyses on ANSYS

The nonlinear options in ANSYS include geometric nonlinearities, (large deflection, large rotation, and stress stiffening), special nonlinear elements, and nonlinear material behavior. Large deflection analysis is available for both static and dynamic problems. Geometry modifications are made at the end of each load increment; stress stiffening and large rotation effects are available for most libraries.

The nonlinear transient dynamic analysis is an extension of the static analysis that solves for the dynamic response of a structure under the action of applied time dependent loads. Inertia and damping effects may be included as well as all nonlinearities shown with the static analysis. The basic equation being solved in the nonlinear transient dynamic analysis is (ref.41)

\[ M\ddot{u} + C\dot{u} + Ku = F(t) \]  

(B.1)

where

\[ F(t) = \text{applied nodal force load vector} \]

- **Implicit Time Integration Algorithm**

  The equation is solved by an implicit direct integration scheme based on a modified Houbolt method. The method uses a cubic displacement function during an iteration. Thus, the velocity varies quadratically and acceleration varies linearly during the iteration. The displacement function is assumed as

\[ u_t = a + b \ t + c \ t^2 + d \ t^3 \]  

(B.2)
a set of four equations may be defined (at \( t = 0, h_1, h_2, h_3 \))

\[
\begin{align*}
  u_{i,3} &= a \\
  u_{i,2} &= a + bh_i + ch_i^2 + dh_i^3 \\
  u_{i,1} &= a + bh_i + ch_i^2 + dh_i^3 \\
  u_i &= a + bh_i + ch_i^2 + dh_i^3
\end{align*}
\]  

(B.3)

where

\[
\begin{align*}
  h_1 &= t_{i,2} - t_{i,3} \\
  h_2 &= t_{i,1} - t_{i,3} \\
  h_3 &= t_i - t_{i,3}
\end{align*}
\]

Solving equations (B.3) gives the four constants \( a, b, c, d \) as

\[
\begin{align*}
  a &= u_{i,3} \\
  b &= e_1 u_i + e_2 u_{i,1} + e_3 u_{i,2} + e_4 u_{i,3} \\
  c &= g_1 u_i + g_2 u_{i,1} + g_3 u_{i,2} + g_4 u_{i,3} \\
  d &= k_1 u_i + k_2 u_{i,1} + k_3 u_{i,2} + k_4 u_{i,3}
\end{align*}
\]  

(B.4)

where

\[
\begin{align*}
  L &= h_2^3 - h_1^3 \\
  M &= h_2^3 h_1 - h_1^2 h_2 \\
  N &= h_2^3 h_1^2 - h_1^3 h_2^2 \\
  P &= h_3^3 - h_1^3 \\
  Q &= h_3^3 h_1 - h_1^2 h_3 \\
  R &= h_3^3 h_1^2 - h_1^3 h_3^2 \\
  \hat{e} &= RM - NQ
\end{align*}
\]

\[
\begin{align*}
  e_1 &= \frac{(Nh_i^3)}{\hat{e}} \\
  e_2 &= \frac{-(Rh_{i}^3)}{\hat{e}} \\
  e_3 &= \frac{(Rh_{i}^3 - Nh_{i}^3)}{\hat{e}} \\
  e_4 &= \frac{(NP - RL)}{\hat{e}}
\end{align*}
\]

\[
\begin{align*}
  g_1 &= \frac{-(Mh_i^3)}{\hat{e}} \\
  g_2 &= \frac{(Qh_{i}^3)}{\hat{e}} \\
  g_3 &= \frac{(Mh_{i}^3 - Qh_{i}^3)}{\hat{e}} \\
  g_4 &= \frac{(QL - MP)}{\hat{e}}
\end{align*}
\]
Differentiating equation (B.2) gives

\[
\begin{align*}
\dot{u} &= b + 2ct + 3d t^2 \\
\ddot{u} &= 2c + 6dt
\end{align*}
\]  

(B.6)

By substituting equation (B.4) into equation (B.6), \( \dot{u}, \ddot{u} \) can be rewritten as

\[
\begin{align*}
\dot{u}_i &= (e_1 + 2g_1 h_3 + 3k_1 h_3^2)u_i + (e_2 + 2g_2 h_3 + 3k_2 h_3^2)u_{i-1} \\
&\quad + (e_3 + 2g_3 h_3 + 2k_3 h_3^2)u_{i-2} + (e_4 + 2g_4 h_3 + 2k_4 h_3^2)u_{i-3} \\
&= c_1 u_i + c_3 u_{i-1} + c_4 u_{i-2} + c_5 u_{i-3}
\end{align*}
\]

\[
\begin{align*}
\ddot{u}_i &= (2g_1 + 6h_3 k_1)u_i + (2g_2 + 6h_3 k_2)u_{i-1} + (2g_3 + 6h_3 k_3)u_{i-2} \\
&\quad + (2g_4 + 6h_3 k_4)u_{i-3} \\
&= c_2 u_i + c_6 u_{i-1} + c_7 u_{i-2} + c_8 u_{i-3}
\end{align*}
\]  

(B.7)

Combining equations (B.1) and (B.7) results in the governing dynamics equation in the following form

\[
\hat{K} u_i = \hat{R}
\]  

(B.8)

The effective stiffness matrix \( \hat{K} \) and the effective load vector \( \hat{R} \) are

\[
\begin{align*}
\hat{K} &= c_2 M + c_1 C + K \\
\hat{R} &= F(t_i) - M(c_6 u_{i-1} + c_7 u_{i-2} + c_8 u_{i-3}) - C(c_3 u_{i-1} + c_4 u_{i-2} + c_5 u_{i-3})
\end{align*}
\]  

(B.9)
In the case of a constant time step size $h$, equation (B.7) reduce to the well known Houbolt algorithm (ref. 46).

\[
\dot{u}_i = \frac{1}{6 \, h} (11u_i - 18u_{i-1} + 9u_{i-2} - 2u_{i-3}) \\
\ddot{u}_i = \frac{1}{h^2} (2u_i - 5u_{i-1} + 4u_{i-2} - u_{i-3})
\]  

(B.10)

where the effective stiffness matrix and load vector are

\[
\hat{K} = \frac{2}{h^2} M + \frac{11}{6 \, h} C + K \\
\hat{R} = F(t_i) + M (\frac{5}{h^2} u_{i-1} - \frac{4}{h^2} u_{i-2} + \frac{1}{h^2} u_{i-3}) \\
+ C (\frac{3}{h} u_{i-1} - \frac{3}{2 \, h} u_{i-2} + \frac{1}{3 \, h} u_{i-3})
\]

(B.11)

For nonlinear transient analyses, the default of ANSYS extrapolates the displacements to the next time point for the stress solution if nonlinear materials are included. The displacement extrapolation is used to calculate the plastic strains which are, in turn, used to calculate the plastic load vector for the next time step. Extrapolation use the same form as the original assumption of cubic variation of displacements (equation B.2), but now the values are explicitly known. The last calculation relating to the nonlinear transient dynamic analysis iteration is to determine the iteration time step size to be used during the next iteration; this value of next time iteration step size must undergo various tests otherwise overshoot and unstable conditions might occur.
Figure 1. FLEX/32 Multicomputer.

Figure 2. iPSC Hypercube Architecture.

Figure 3. A Simple Taxonomy of Parallel Computers
Figure 4. Growth in Computer Speed
Figure 5. Deformed shapes at 10ms output intervals.
Impact mass $M = 1932\, \text{kg}$
Impact velocity $V = 4.65\, \text{m/s}$

Steel material:
$E = 2.07 \times 10^5\, \text{N/mm}^2$
$\sigma_y = 250\, \text{N/mm}^2$
$\sigma / \sigma_y = 1 + \left( \frac{\dot{\epsilon}}{\dot{\epsilon}_y} \right)^2$
$\rho = 7.9\, \text{kg/dm}^3$

Torque box subjected to impact

Figure 6. S-Frame Test Problem
Shared Displacement $U$, Velocity $\dot{U}$

Waiting for Time Step Open

Strain $\varepsilon^{(p)}$

Stress $\sigma^{(p)}$

$M^{(p)}$, $C^{(p)}$, $K^{(p)}$

$R_{\text{int.}}^{(p)}$, $R_{II}^{(p)}$

Shared $M_{\text{system}}$, $\hat{F}_{\text{system}}$

Acceleration $\ddot{U}^{(p)}$

Figure 7. Concurrent Explicit Finite Element Method.
Figure 8. Cantilever Beam (FENRIS)
Figure 9. LDLT Decomposition (FLEX/32).

Static Problem:
- Total elements = 100
- Total D.O.F. = 444
- Max. Half-Bandwidth = 48
Figure 10. LDLT Decomposition (CRAY X-MP/48).
Geometry of a Shallow Arch.

Figure 11. Intel iPSC Communication Sequence.
Figure 12. Nonlinear Transient Dynamic (ANSYS)
Figure 15. Nonlinear Analysis (FENRIS)
Figure 14. Illustration of the Alternative Solution Methods (FENRIS)
**A. Rigid-viscoplastic (TRUMP, Argyris)**

**B. Elastic-perfect plastic (FENRIS)**

**C. Elastic-perfect plastic (simple lumped, Argyris)**

**Figure 15. Reaction Force During Impact.**
Impact Velocity = 461 cm/sec

Figure 16. Time - Deflection.
Figure 17. 449 D.O.F S-Frame.
Figure 18. 30 D.O.F. Cantilever Beam.