ABSTRACT

DISCRETE ELEMENT MODELING OF DRY GRANULAR MATERIAL USING A MASSIVELY PARALLEL SUPERCOMPUTER

by
David W. Washington

It is the state-of -the-art within Geotechnical Engineering to model soils as systems of particles rather than using the traditional continuum approach. Simulating these systems of particles for geotechnical boundary value problems results in systems which are of necessity large, motivating the application of massively parallel supercomputers. This thesis pursues such an approach.

The following work describes numerical experiments using a Discrete Element Method (DEM) paradigm for soils (Trubal) together with massively parallel computers with Single Instruction Multiple Data (SIMD) architecture. The discrete element method describes the behavior of granular assemblies using the classical mechanics of discrete bodies. The computational requirements of DEM algorithms introduce time complexities, which mandate a compatible topology for massively parallel machines in order to achieve optimal performance. This thesis demonstrates the compatibility of a Single Instruction Multiple Data (SIMD) topology in performing discrete element simulations for 3-d spherical dry granular media.

The *serial* algorithm, Trubal, was first modified to run with a parallel data structure on a SIMD architecture. The modified version, known as Trubal for Parallel Machines (TPM), is the *data parallel* version that was tested on the connection machines (CM-2) and (CM-5), consisting of 32,768 processors and 512 nodes, respectively. The first version of TPM was tested on the CM-2 machine before its use was discontinued.

Because the architecture is synchronized at each instruction, elemental data movements reduce the performance of the machine's overall resources and increase the latency of the communication between processors. This issue is addressed within the design of the algorithm so that the SIMD vector processing capability can adapt to a dynamic memory data structure.

A second version of TPM was subsequently designed for the CM-5 machine using a more efficient parallel data structure to improve the performance of the simulations. TPM version 2.0 was able to obtain a speedup in performance by handling all possible contacts within each processor, thereby creating a homogeneous data structure. The overall efficiency is governed by the global communication which is a function of the speed of the interconnection network within the architecture.

TPM's improved performance is demonstrated using two different triaxial simulations. One of them involved a physical triaxial experiment with steel spheres performed by Rowe (1962) and later simulated by Cundall (1979). The remodeling of this numerical simulation validated TPM version 2.0 overall performance where a nine-fold speedup was obtained. TPM's reproduction of these results and its improved speedup encourage further investigations using discrete models on parallel platforms.

This thesis substantiates the use of parallel computing as a technique for geotechnical applications. It is further anticipated that developing and adapting heterogeneous platforms to DEM models will make the application of parallel computing more attractive in geotechnical engineering.

DISCRETE ELEMENT MODELING OF DRY GRANULAR MATERIAL USING A MASSIVELY PARALLEL SUPERCOMPUTER

by
David W. Washington, P.E.

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Civil and Environmental Engineering

May 1996

APPROVAL PAGE

DISCRETE ELEMENT MODELING OF DRY GRANULAR MATERIAL USING A MASSIVELY PARALLEL SUPERCOMPUTER

David W. Washington

Dr. Jay N. Meegoda, Dissertation Advisor — Date
Associate Professor of Civil and Environmental Engineering, NJIT

Dr. William Spillers, Committee Member — Date
Chairman and Professor of Civil and Environmental Engineering, NJIT

Dr. Dorairaja Raghu, Committee Member — Date
Professor of Civil and Environmental Engineering, NJIT

Dr. Mary Eshaghian, Committee Member — Date
Assistant Professor of Computer and Information Science, NJIT

Dr. Anthony Rosato, Committee Member — Date
Associate Professor of Mechanical Engineering, NJIT

BIOGRAPHICAL SKETCH

Author:	David Waymon Washington, P.E.
Degree:	Doctor of Philosophy
Date:	May 1996

Undergraduate and Graduate Education:

•Doctor of Philosophyin Civil
New Jersey Institute of Technology, Newark, N.J., 1996

•Master of Science in Civil Engineering,
Manhattan College, Riverdale, N.Y.,1988

•Bachelor of Science in Civil Engineering and Applied Mechanics,
Columbia University, New York, New York,1984

Major: Civil Engineering

Presentations and Publications:

N. Meegoda and D.Washington(1993) "Trubal for Massively Parallel Machines", *Proceedings of the Second International Conference on Micromechanics Of Granular Media*,Powders & Grain 93, Birmingham, UK, July 12-16,Poster Presentation

Meegoda N.J.and Washington D.W.(1994),"Massively Parallel Computers for Microscopic Modeling of Soils", *Proceedings of the Eighth International Conference on Computer Methods and Advances in Geomechanics*,Morgantown, West Virginia, May 22-24, pp.617-622

Washington D.W and Meegoda N.J(1996), "Micromechanical Simulation of Geotechnical Problems using Massively Parallel Supercomputers", *Proceedings of the Eleventh ASCE Engineering Mechanics Conference*, Fort Lauderdale, Florida, May 19-22,to be presented

This thesis is dedicated to
The Lord Jesus Christ who is the Author and Finisher of my Faith

ACKNOWLEDGMENT

This research could not have been done without the motivation and encouragement from my advisor, Dr . Jay Namunu Meegoda. He has placed so much confidence in me and has allowed me to take on a great deal of responsibility, which inevitably led to my maturity in this field.

My committee has my warmest appreciation and deepest gratitude for assisting me in various facets of my research. The hours that I spent with Prof. Spillers discussing applied mechanics and other topics was phenomenal and very encouraging.

Prof. Eshaghian is my mentor for learning parallel computing and various architectures. She exposed me to a great deal of information in a very short time.

Dr. Rosato was very supportive of my work and gave me much insight in particle technology. This research has led us both to England and France.

Dr. Raghu and I have had many wonderful moments as my instructor in geotechnical engineering. He has always encouraged me to take a practical approach to my work.

Ms. Sheridan Quarless, soon to be a Ph.D., has become a "household" name in my academic pursuits. She has been my motivator from the first day I entered the doctoral program, and has plowed the way for me in many of my endeavors.

Much appreciation and gratitude to Dr. Kane and the graduate studies office for their support. Dr. Kane literally financed and sustained me through this program.

The international exchange program enabled me to spend many months in France, and this collaboration led to many new relations with people in my field. Special thanks to

Katie Byrne for coordinating my trips to France with Dr. Kane and introducing me to Prof. Martin Raynaud of the Institut National Des Sciences Appliquees De Lyon, Departement de Genie Civil,Villeurbanne, Lyon.

Prof. Raynaud had a leading role in my studies abroad as well as Dr. Bernard Cambou from Ecole Centrale de Lyon of Laboratoire de Tribologie et Dynamique des Systemes Departement de Mecanique des Solides, Ecully, Lyon. This exchange brought great insight to my work and developed many good relations abroad.

A lot of motivation has come from my wife, Jackie, and daughter, Amy Joella. My new born daughter, Amy, made me realize that the Ph.D. is not the end of the road, but there are much more challenges ahead. My wife, who is also a doctoral candidate in Microbiology and Molecular Genetics, has kept me focused in my academic pursuits, and hopefully Amy will follow the path that we are paving.

I was fortunate to have my family and close friends support me through my arduous studies. Their support meant very much to me.

TABLE OF CONTENTS

Chapter Page

1. INTRODUCTION 1

1.1 Modeling the Behavior of Soils 1

1.2 Overview of the Discrete Element Method 3

1.3 The Discrete Element Method's Formulation 5

1.4 Experimental Validation of the DEM Model 12

1.5 DEM and Other Modeling Techniques in Continuum Mechanics 13

1.6 Parallel Computing Terminology Terminology 16

1.7 Literature Survey 18

1.7.1 Ghaboussi Approach 18

1.7.2 Kuriaoka's Approach 19

1.7.3 Hustrulid's Approach 19

1.7.4 O'Connor Approach 20

1.8 Research Program 20

2. TPM VERSION 1.0 22

2.1 Introduction 22

2.2 General Description of Trubal's Algorithm 23

2.2.1 Gen Routine 24

2.2.2 Cycle Routine 24

2.2.3 Motion Routine 24

TABLE OF CONTENTS
(continued)

Chapter Page

2.2.4 Rebox Routine 25

2.2.5 Search Routine 26

2.2.6 Bbtest Routine...26

2.2.7 Ford Routine..27

2.3 CM-2 Architecture..27

2.4 TPM Version 1.0...31

2.5 Advantages of TPM's Data Structure vs. Trubal 38

2.6 Evaluation and Results of TPM Version 1.0...39

2.7 Discussion and Other Modifications..42

2.8 The Last Modification of TPM 1.0...43

2.9 Conclusion...45

3. TPM VERSION 2.0...46

3.1 Introduction..46

3.2 CM-5 Architecture..47

3.3 TPM Version 2.0...52

3.4 Experimental Verification and Performance Evaluation.....................................57

3.5 Conclusion..60

4. SIMULATIONS OF ROWE'S EXPERIMENT..61

4.1 Introduction...61

TABLE OF CONTENTS
(continued)

Chapter **Page**

4.2 Rowe's Physical Laboratory Test....62

4.3 Cundall's Numerical Simulation....63

4.4 TPM's Simulation of Rowes Model....66

4.5 Discussion....68

4.6 Conclusion....69

5. FUTURE RESEARCH AND CONCLUSION....71

5.1 Conclusion....71

5.2 Future Research....72

5.2.1 The MIMD Approach....73

5.2.2 Combining Control Systems....75

5.2.3 Heterogeneous Platform....76

5.3 Cray T3D and PVM....78

5.4 Implementation of the Proposed Heterogenous Platform....79

APPENDIX TPM Version 2.0 (ROWE'S MODEL)....81

REFERENCES....130

LIST OF TABLES

Table **Page**

2.1 CM-2 Resource usage viewed in PRISM........40

2.2 Comparison of timing between CM-2 and Vax 8800........45

3.1 Comparison between the CM-2 and the CM-5........51

3.2 Declaration statements and psuedocode flow chart for TPM version 2.0........54

3.3 Pseudocode for contact routine in TPM version 2.0........55

3.4 Pseudocode for globe routine in TPM version 2.0........56

4.1 Parameters of material properties used in Cundall's numerical simulation........64

4.2 Speedup chart for the 403 and 1672 sphere simulation........68

5.1 Control System Optimization at Routine Level of the CM-5........76

5.2 Control System Optimization at Routine Level of a Heterogeneous Platform........72

LIST OF FIGURES

Figure **Page**

1.1 Cartesian coordinate system for spheres....7

1.2 a)Contact vectors between spheres and
b)Internal moments and forces due to contacts....8

1.3 Schematic showing principle rheological elements of DEM (2D)....12

1.4 a) Force vector plots obtained by De Josselin de Jong and Verruijt(1969) and
b) Program "BALL" numerical reproduction of the 1969 experiment....13

2.1 Flow Chart for Trubal's Algorithm....23

2.2 Spheres encompassed by cubes mapped within Boxes....25

2.3 Data structure for Trubal's one dimensional array....26

2.4 CM-2 Architecture....28

2.5 An array (64,64) layout over sixteen processor....29

2.6 The CM-2 parallel processing unit....30

2.7 Interconnect networks systems....31

2.8 Flow chart for TPM version 1.0....32

2.9 Data structure for the arrays of TPM version 1.0....33

2.10 An example of two spheres within four boxes....34

2.11 TPM's Box Array Initial Setup....35

2.12 TPM's Box Array after "Spread" command and "Forall" command is used....36

2.13 One dimensional contact arrays for TPM 1.0....37

3.1 CM-5 Architecture....47

LIST OF FIGURES

Figure **Page**

3.2 Vector unit functional architecture 48

3.3 Organization of the Connection Machine (CM-5) 49

3.4 Binary and Fat tree interconnection network for 15 node system 50

3.5 Flow chart for TPM version 2.0 52

3.6 A comparison between Trubal and TPM of a simulated triaxial test....................59

4.1 Layers of spheres used in physical test by Rowe(1962)....................63

4.2 Layers of spheres in the numerical test....................63

4.3 Exposed lengths for plane through large layer in x-y and x-z direction....................65

4.4 Comparison between Rowe's results and Cundall's numerical results....................66

4.5 Instability in TPM version 2.0 simulation due to improper damping....................67

4.6 Results presented by Rowe, Cundall, and TPM version 2.0....................67

4.7 Speedup plot for TPM algorithm simulating the 403 sphere Rowe model....................69

5.1 Clustering Schemes of Particles within Multiple Processors 74

5.2 3D Torus Communication Network....................78

5.3 TPM's proposed heterogeneous algorithm using PVM....................79

CHAPTER 1

INTRODUCTION

In order to understand the necessity of using massively parallel supercomputers in discrete element modeling for geotechnical applications, the development of soil models and numerical techniques is discussed in this chapter. Thc fundamentals of the discrete element method are also presented along with a discussion of how the discrete element method fits into the scheme of available modeling techniques. Then, parallel computing terms are defined so that the unfamiliar reader can follow the subsequent chapters describing computer architecture. Finally, a literature search presents the history of parallel computing using discrete models.

1.1 Modelling the Behavior of Soils

It is the state-of-the-art in geotechnical engineering to model soils as a system of particles rather than using the more traditional continuum approach. In the case of this thesis, the particle approach is driven by physical concerns over how particles really interact in a soil mass. This approach allows contact friction between particles, for example, to be studied in detail. Friction is, of course, a basic mechanism in the performance of soil. The down-side to this approach is that the particles used in this thesis are round and smooth while soil particles typically possess neither property.

Soil modeling at the particle level has been studied for many years. For example, Deresiewicz (1958) and Duffy (1959) pursued a micromechanic approach of assemblies with regular packing. In this case, the constitutive relationship for a granular assembly was derived considering particle interaction and structure of the material. However, the ability of particles to slip at contact surfaces was not included in their model. In the absence of computers, it was difficult, if not impossible to model sophisticated configurations of particles. That, of course, is no longer the case.

The range of particle models now used in mechanics is quite broad. Probably the oldest model is the paricle-in-cell model developed in the 1950's at the now defunct Atomic Energy Commission. This model is actually a combination of a particle approach and a continuum approach. At the other end of the spectrum are "atomic" models, such as Molecular Dynamics (MD), which now include hundreds of thousands of particles. The model used here lies somewhere in the middle and has been chosen for use in this thesis because of its practical applicability. That is, it is not as computationally demanding as the "atomic" models and it has the potential of explaining the frictional failure mechanism of soil. The model being described incorporates micromechanics to define interparticle behavior.

Micromechanics is the mechanics or the physical laws used to describe the interactions between individual bodies. The micromechanical modeling of soils used in this thesis considers the intergranular effects that contribute to the overall behavior of the soil assembly. This type of modeling coupled with a technique that applies it to granular systems, has the potential to realistically describe global behavior. In this case, the individual particles of the global assembly are actually governed by force/displacement law approximations. The necessity for modeling soil behavior from the granular interaction level was recognized for many years before computer simulations of these micromechanical models originated. The traditional methods, such as the Mohr-Coulomb theory, viewed soils as a continuum. However, Rowe (1962) stated that the dilatancy and strength of an assembly are dependent upon its angle of friction between particle surfaces, on the geometrical angle of packing, and on the degree of energy loss during remoulding. He also showed that the Mohr-Coulomb criterion of failure does not have general application to a discontinuous assembly of particles. Terzaghi (1920) stated that Coulomb introduced a fundamental error by ignoring the fact that sand consists of individual grains. Other researchers studied intergranular behavior from an experimental perspective.

Research performed by Dantu (1957) and De Josselin de Jong .et al. (1969) presented pictures of actual interparticle behavior using photoelastic discs. Under polarized light the contact forces and the particle displacements were measured. Observations of the soil behavior from these type of experiments, revealed the rearrangement of soil particles to form internal structures to withstand the loads that were being applied. For example, columns of particles aligned themselves in the direction of the load in order to withstand applied forces. The discrete element method was developed to numerically describe this type of formation within the particle assembly.

1.2 Overview of the Discrete Element Method

Discrete Element Method (DEM) used in this thesis is an application of Cundall and Strack (1979b) model. Cundall (1971) first applied his model to the stability of high, fractured rock slopes. This method was further improved when a two-dimensional disc assembly was used to simulate soil conditions (Cundall and Strack 1978). Later, three-dimensional spheres were used (Cundall and Strack 1979a). Many other research efforts have derived from Cundall's work.

The 2nd International Conference on DEM (Williams and Mustoe, 1993) states, *the Discrete Element Method models a physical system as an assemblage of distinct bodies each having internal structure and specific geometry. At a basic level the individual elements can simulate continuum stress and deformation states, while the multiplicity of bodies provide the degrees of freedom otherwise unavailable in methods based solely on continuum assumptions.* The continuum approach often overlooks interparticle behavior and therefore the mode of failure within the assembly has questionable physical representation. One reason is that discontinuities are found in the stress and the displacement fields in granular assemblies, which are difficult to describe in a continuum formulation. These difficulties were resolved with global stress tensor formulations in the discrete element method.

The DEM is based on a dynamic formulation, which describes a static equilibrium in asymptotic sense after a number of time-steps. Hence, it is able to model the quasi-static case found within many geotechnical boundary value problems. Spheres are placed in a computational cell having periodic boundaries. This boundary allows a sphere which exits from one face of the cell to re-enter through the opposite face. Consequently, the influence of real walls is removed and the assembly of spheres is effectively a representative volume of material within a large system. Stresses and strains are applied to the boundaries of the assembly and then propagated through the medium until some state of equilibrium is reached. This method allows particles to form contacts and break them at will. Each time-step cycle includes the application of Newton's Second Law to the centroid of each sphere followed by the application of force-displacement laws between each contact. By maintaining small time-steps, disturbances (that initiate at the assembly boundaries) will not propagate further than a neighboring contact sphere during a time-step. Accelerations and velocities of each sphere calculated from Newton's Second Law are assumed to be constant over the time-step, and the net forces and moments acting on each sphere are updated from force-displacement laws applied at the contacts of neighboring spheres. The following section describes the mechanics involved within the DEM model used in this thesis.

1.3 The Discrete Element Methods Formulation

The mechanics of DEM for a dry spherical granular assembly is used to determine the motion of spheres, to perform force-displacement calculations at contacts, and finally to sum the overall global stress tensors. As mentioned before, the time step is assumed to be so small that the velocities and accelerations are assumed to remain constant during that interval. The time step is computed from a single degree-of-freedom mass-spring model:

$$\Delta t_C = 2 * (m_{min}/k_n)^{1/2} \text{ (critical time step)} \qquad (1.1)$$

$$\Delta t = \Delta t_C * FRAC \qquad \text{(time step used)} \qquad (1.2)$$

where Δt_C equals the critical time step, m_{min} is the minimum mass of the spheres, k_n is the normal stiffness, Δt is the time step used and FRAC is a fraction multiplied of the critical time step which allows for a stable simulation. The time step used in this explicit numerical scheme is required to be some fraction of the critical time step in order for the system to remain stable (Cundall et al., 1978). The explicit finite-difference scheme changes the assembly's geometry by updating current translational and angular velocities at the end of each time-step. The finite difference scheme, used to integrate the system's equations of motion, uses the subscript N will be used to represent the beginning of the current time step, N-1/2 will represent the results from a previous time step, and N+ 1/2 will denote the new result for the current time step.

The general form of the governing equation for each particle within the system can be expressed dynamically by the relationship

$$ma+c\,v+kx=F(t) \tag{1.3}$$

where m, c, and k are respectively the particles' mass, damping, and stiffness coefficients, and a, v, and x are the particles' acceleration, velocity, and displacement respectively. The first part of equation 1.3 is represented by Newton's Second Law which can be written as follows:

$$m_j(a_i)_N=(F_i)_N \tag{1.4}$$

$$I_j(\alpha_i)_N=(M_i)_N \tag{1.5}$$

where i is the x, y, z components as shown in Figure 1.1. In expressions 1.4 and 1.5, m and I represent the mass and moment of inertia of the sphere for all spheres j, and F_i and M_i are the net force and moment components, respectively at the beginning of the time-step, t_N.

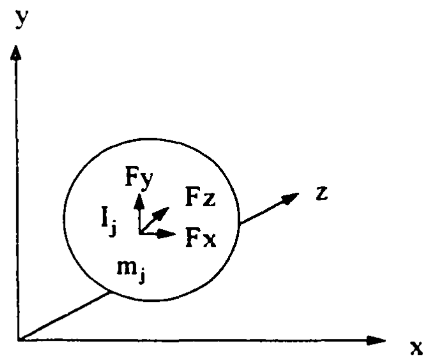


Figure 1.1 Cartesian coordinate system for spheres

Since the accelerations a_i and α_i are constant over the interval $t_{N-1/2}$ to $t_{N+1/2}$, then the sphere velocities can be calculated from

$$(v_i)_{N+1/2}=(v_i)_{N-1/2}+[(F_i)_N\Delta t\ /m] \tag{1.6}$$

$$(\omega_i)_{N+1/2}=(\omega_i)_{N-1/2}+[(M_i)_N\Delta t\ /I] \tag{1.7}$$

where v_i is the translational velocity and ω_i is the angular velocity (see Figure 1.2). The coordinates of each sphere are calculated at the end of the time step t_{N+1} with the relationship

$$(x_i)_{N+1}=(x_i)_N+((v_i)_{N+1/2})\Delta t \tag{1.8}$$

$$(\omega_i)_{N+1}=(\omega_i)_N+((\omega_i)_{N+1/2})\Delta t \tag{1.9}$$

The contact forces are calculated using an incremental force-displacement law of the form

$$(Fn)_{N+1}=(Fn)_N+(\Delta Fn)_N=(Fn)_N+k_n(\Delta n)_{N+1/2}\ \text{(contact normal forces)} \tag{1.10}$$

$(Fs)_{N+1}=(Fs)_N+(\Delta Fs)_N=(Fs)_N+k_s(\Delta s)_{N+1/2}$ (contact shear forces) (1.11)

where Fs and Fn are the normal and shear contact forces, k_n and k_s are the normal and shear stiffness and Δn and Δs are the normal and shear displacements, respectively. The normal and shear displacements are expressed using the Einstein summation convention as

$(\Delta n_i)_{N+1/2}=((v_{bi}-v_{ai})_{N+1/2})\, e_i\, \Delta t \quad i=x,y,z$ (1.12)

$(\Delta s_i)_{N+1/2}=[((v_{bi}-v_{ai})_{N+1/2})\, t_i - (\omega_{ai}R_a+\omega_{bi}R_b)_{N+1/2}]\Delta t$ (1.13)

where $(v_{bi}-v_{ai})$ is the relative velocity between two contacting particles *a* and *b*, e_i and t_i are the normal and tangential unit vectors, and R_a and R_b are the radiuses corresponding to the particles a and b respectively.

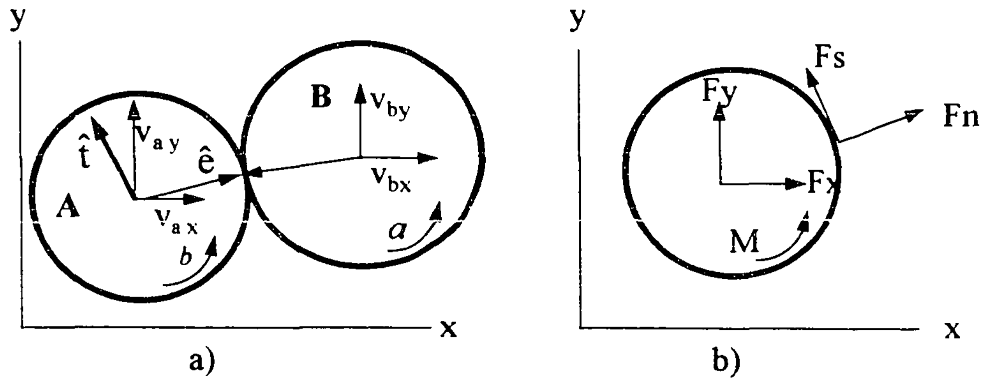


Figure 1.2 a)Contact vectors between spheres and b)Contact internal moments and forces

The Coulomb-type friction law is used to account for particles that slip once a threshold tangential force level is achieved. The friction expression for each contact is written as

$$|(Fs)|\,max = |\mu F_n| + cohesion \qquad (1.14)$$

where |(Fs)|max , Fn, and μ is the maximum shear force, the normal contact force and the coefficient of friction, respectively. If the maximum critical shear force has been exceeded by the actual shear force between the two particles, then the two particles slide and their interparticle shear force is reduced as if (Fs)max<(Fs) then

$$(Fs)_i = (Fs)_i * [(Fs)max/(Fs)] \quad i = x,y,z \qquad (1.15)$$

At the end of the contact calculations in the time-step, the sphere forces F_i and moments M_i for each sphere are computed by summing up all the contact components as

$$(F_i)^k_{N+1} = \sum_{n=1}^{n_k} ((Fn)_{N+1} e_i + (Fs)_{N+1}\, t_i) \qquad (1.16)$$

$$(M_i)^k_{N+1} = r^k \sum_{n=1}^{n_k} ((Fs_i)_{N+1}) \qquad (1.17)$$

where n_k refers to the contacts associated with the sphere k having the radius r^k . The forces and moments found here are used to update the translational and angular velocities in the next time-step, by integrating the equations of motion of the system.

The damping that is considered is a global damping, which acts on the absolute velocities and can be envisaged as dashpots which connect each sphere to a fixed frame of reference. The global damping coefficients are related to the mass and moment of inertia of each sphere through the coefficients of proportionality $\propto$

$$c_m = \propto m \qquad (1.18)$$

$$c_I = \propto I \qquad (1.19)$$

The coefficient of proportionality is selected arbitrarily in order to maintain stability within the system. Equations 1.4 and 1.5 can be modified to include the effect of damping as

$$m(a_i)_N = (F_i)_N - c_m(v_i)_N \quad i=x,y,z \qquad (1.20)$$

$$I(\alpha_i)_N = (M_i)_N - c_I(\omega)_N \qquad (1.21)$$

If the central difference scheme is used to integrate the above, whereby velocities are evaluated halfway through the time step, then

$$(v_i)_N = 1/2\,[(v_i)_{N+1/2} + (v_i)_{N-1/2}] \qquad (1.22)$$

$$(\omega_i)_N = 1/2\,[(\omega_i)_{N+1/2} + (\omega_i)_{N-1/2}] \qquad (1.23)$$

$$(a_i)_N = \frac{(v_i)_{N+1/2} - (v_i)_{N-1/2}}{\Delta t} \qquad (1.24)$$

$$(\alpha_i)_N = \frac{(\omega_i)_{N+1/2} - (\omega_i)_{N-1/2}}{\Delta t} \qquad (1.25)$$

This leads to the revised equations of motion in the form

$$(v_i)_{N+1/2} = \frac{(v_i)_{N-1/2}(1 - \propto \Delta t\ /\ 2) + (F_i)_N \Delta t\ /\ m}{(1 + \propto \Delta t\ /\ 2)} \qquad (1.26)$$

$$(\omega_i)_{N+1/2} = \frac{(\omega_i)_{N-1/2}(1 - \propto \Delta t\ /\ 2) + (M_i)_N \Delta t\ /\ m}{(1 + \propto \Delta t\ /\ 2)} \qquad (1.27)$$

If $\propto$ is zero, we will have the equation presented in 1.6 and 1.7 (see Fig 1.3).

The global stress tensors are calculated for the overall stress behavior represented by the average stress tensors,

$$\bar{\sigma}_{ij} = \frac{1}{V} \sum_{k=1}^{z} \left((F_n)_{N+1}^{k} e_j + (F_s)_{N+1}^{k} t_j \right) x^k e_i \tag{1.28}$$

where , V represents the total volume of the assembly, x^k is the sum of the two radii forming the contact k, and z is the total number of contacts in the assembly. For the derivation of equation 1.28, the reader is referred to Cundall et. al. (1982).

Cundall's algorithm, Trubal version 1.51 adopts these equations for the routines Motion and Ford which are described in the next chapter. Trubal for Parallel Machines (TPM), the modification of Cundall's Trubal, also adopts these equations in the routines Motion, Ford, and Globe. The fundamental mechanics of DEM just presented are validated in an actual experiment and are presented in order to give the unfamiliar reader a feel of its physical applicability.

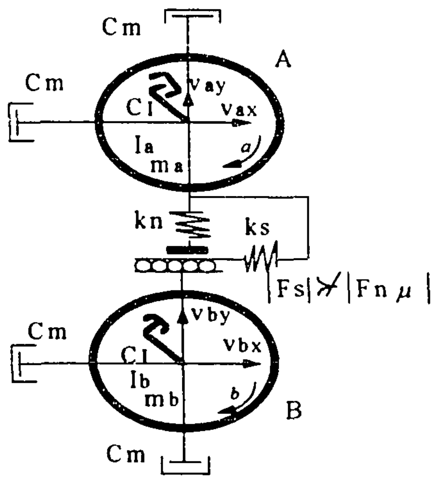
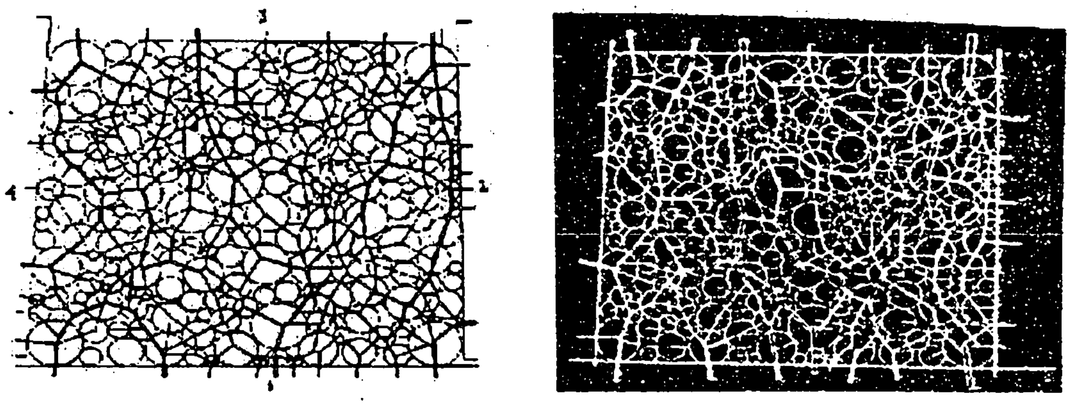


Figure 1.3 Schematic showing principle rheological elements of DEM (2D)

1.4 Experimental Validation of the DEM Model

Producing physical results from numerical computations of spherical bodies is difficult, especially since fictitious values, such as damping are included without any physical meaning. However, Cundall's program "Ball", the 2-d version of Trubal, qualitatively describes the microscopic effects for quasi-static problems on a global continuum level. The result presented in this section is reported in Cundall and Strack (1978) in more detail.

Program "Ball" was validated by simulating the test performed by de Josselin de Jong and Verruijt (1969) on photoelastic discs. An assembly of discs were placed between two plexiglass plates, that prevented the stack from buckling out of plane. It was loaded by an applied force perpendicular to four rigid beams encompassing it. When viewed in circularly polarized light, the photoelastically sensitive discs displayed a pattern of isochromatics. From these patterns, one can deduce the forces that are transmitted through the disc contact points. At each stage of the test De Josselin de Jong and Verruijt recorded the disc locations and contact forces. When the ratio between the horizontal force and the vertical force was 0.39, the deformation of the assembly appeared as shown in Figure 1.4 a. The discs center locations and radii were obtained by using a digitizer. The center coordinates of the discs, marked by cross-hairs in the original test, were located with reference to a grid in the background of the photograph. Similar results were obtained from a strain control simulation shown in Figure1.4 b for



a) ratio Fh/Fv=0.39 b) ratio Fh/Fv=0.33

Figure 1.4 a) Force vector plots obtained by De Josselin de Jong and Verruijt (1969) and b) Program "BALL" numerical reproduction of the 1969 experiment

comparison at the ratio of 0.33. The contact forces (branches) were observed between particles within the assembly in each model and there was good agreement. Although these results are good for the use of this model, there are other techniques that can be applied also. The following section presents and compares some of the other modelling techniques

1.5 DEM and Other Modeling Techniques in Continuum Mechanics

Since continuum mechanics is the classical approach used to solve geotechnical boundary value problems, a comparative study was done to highlight its relationship to the discrete element method. The three modelling techniques selected for comparison in this section are a Finite Element Method (*FEM*) model for granular systems, an *explicit DEM* numerical scheme adopted for this thesis, and an *implicit DEM* scheme called the Discontinuous Deformation Analysis (DDA). Of course, there are many other schemes for solving granular systems, but these three techniques represent the diversity in approaches.

For example, the *FEM* is used quite extensively to model almost every observed behaviour that is found in soils. Rodriguez -Ortiz (1974) is noted by Cundall et al. (1979c) to have the closest analysis to the DEM approach. In his approach, assemblies of discs are represented by the finite element method. A stiffness matrix is constructed that takes into account the geometrical arrangement of the particles and the current stiffness at each contact. Inverting the matrix allows incremental displacements to be computed from the last known forces, with an iteration procedure being necessary to deal with slip at contacts. Only one contact is allowed to slip at a time, and it is necessary to reform the stiffness matrix when the contacts join or break. It is important to note that in this method all particles interact with each other during each solution step.

With the *implicit DEM*, for example DDA, the block systems analysis can be closely compared to the finite element method. The discrete blocks are similar to the "elements" in the FEM, but the displacement compatibility condition between adjacent elements in FEM is replaced in DDA by contact relationships between blocks which are based on block kinematics and constitutive laws of discontinuities. The matrix produced from these equations is similar to the FEM with a much more sparse and less banded global coefficient matrix. The displacements and strains of the blocks are used as basic systems of unknowns and the matrix equation is solved by using a direct equation solver. The reader is referred to the paper of Shi [1988] for more details on this implicit approach.

His work was described here to provide the reader with an examples of an implicit discrete element analysis.

In contrast to the schemes just mentioned, the *explicit DEM* is a time-dependent scheme, where each time step is chosen to be so small that during a single time step, disturbances can not propagate from any particle beyond its immediate neighbor. The resultant forces on any particle are determined exclusively by its interaction with the other particles it contacts. This is the reason why the DEM can model the non-linear interaction of a large number of particles without the need for iterative techniques.

A comparison between the *explicit DEM* and the *implicit DEM* reveals that the *explicit DEM* absence of iterative techniques for non-linear behavior eliminates the possibility of converging on a path that is not physically possible. Chang and Acheampong (1993) points out that the advantage of the DEM approach compared with an implicit algorithm is that it requires less computational effort and computer memory. Cundall's model is known to be primarily dependent on artificial global damping to dissipate the energy within the assembly for stable results, and it requires many small time steps to reach a state of equilibrium. The one drawback is particles tending to spin at high speeds during the numerical simulation. Therefore, a high moment of inertia is therefore assigned to each particle to prevent this from happening.

On the other hand, since the implicit scheme doesn't require very small time steps to arrive at a certain stress state, the number of iterations that are required is much less than the number of time steps required by the DEM to achieve the same state. In Shi's DDA, the deformation energy is minimized within a block system to yield equilibrium directly and therefore does not require small time steps. The blocks are treated as linear elastic polygons. Since the block deformation and movement are determined by the minimization of energy, damping and a high moment of inertia is not necessary to achieve stable results. However, this numerical scheme requires that many cumbersome simultaneous equations be solved per time-step. Before the implementations of parallel computing to DEM models can be discussed, a brief description of computer terminology is provided in the next section.

1.6 Parallel Computing Terminology

Since this research is intended to have interdisciplinary appeal, basic computer science terminology will be defined in this section. When the term "massively parallel computer" is used, this signifies that hundreds to thousands of processors are incorporated within a network for a given application. If a machine consists of less processors than just mentioned, it is called a parallel machine. A massively parallel supercomputer is the state-of-the-art technology and can be accessed through national supercomputing sites, such as the Pittsburgh Supercomputing Center (PSC) and the National Center for Supercomputing Applications (NCSA).

The two major types of supercomputers found today are vector machines such as the Cray YMP and the massively parallel machines such as the Cray T3D and CM-5. Although vector processing is found in the Cray YMP, in the case of functional pipelines, the vector processing referred to in this paper is only related to the SIMD architecture. In short, vector processing occurs when arithmetic or logical operations are applied to multiple data as opposed to scalar processing which operates on one or a pair of data. The topology or layout of any parallel machine will be described in terms of its *multiple processor capability*, its *control network*, and its *interconnection network*. These classifications differentiate a parallel machine from a multiple processor serial machine.

The *multiple processor capability* can describe either many low performance processors linked together in a *homogeneous* system, a smaller number high performance processors linked together in a *homogeneous* system, or variable numbers of different processors linked together in a *heterogeneous* system. The first two cases will be discussed in this research for the CM-2 and the CM-5, respectively. In a *homogeneous* system, the same processors/nodes are used throughout the architecture, and in a *heterogeneous* system, different processors/nodes are networked together. In general, the memory is *distributed* or divided among the processors, so that communication links between processors enable memory to be addressed. There are some parallel machines with *shared memory*, in which case the same memory is accessed by all of the processors. Serial machines are an example of a *shared memory* system, although they may have more than one processor. The basic element of a parallel machine is the processor with its attached memory or the "node". A node can be a processor or processors with various functional capabilities. For example, a *transputer* node is a node that contains a processor, memory, and communication links on a single chip, whereas a CM-5 node (described later) consists of a RISC processor, 4 vector units, and its allocated memory.

The *control network* is defined as a single instruction multiple data machine (SIMD) or a multiple instruction multiple data machine (MIMD). Although, these architectures are described in detail later on, the major difference is that SIMD operates synchronously on the entire set of processors assigned to it for each instruction, and MIMD operates independently on each processor at different times.

Finally, the *interconnection network* is the physical wiring of the processors' connections to each other. Examples of this are the hypercube, binary tree, and 3D Torus, which are some of the networks that will be described further. The *communication latency* is a time measure of the communication overhead incurred between the processors/nodes, and the *memory latency* is the time required for the processor to access the memory.

1.7 Literature Survey

1.7.1 Ghaboussi Approach

The operation of massively parallel machines to model soil behavior has been proposed by Jamshid Ghaboussi et al. (1993). Ghaboussi proposed that the connection machine (CM-2) could be used to detect contacts between particles simultaneously (in parallel) with neural networks. Unlike sequential programming, neural networks are trained. The network then internally organizes itself to be able to detect the presented examples. Before the network can become operational for a discrete element algorithm, it has to be trained to detect contacts between particles. If the neural network is properly trained, it is able to give correct responses, or nearly correct responses when presented with partially incomplete stimuli. Its ability to self-organize itself or adjust its weights makes it possible for it to generalize certain conditions of contacts. This is a good approach, but detecting all of the existing contacts is not guaranteed. In order to simulate behavior in construction materials and soils, the arrangement or packing of particles can be quite complex, which means extensive training would be necessary to get close to the number of contacts that exist.

1.7.2 Kuraoka's Approach

Kuraoka (1994) modified Trubal and implemented it to the Intel iPSC/860 with 16 processors. In this case, an assembly of 2400 disc was simulated for the flow of sand in an expendable pattern casting process. It did not exploit the full capabilities of a massively parallel machine nor did it consider the three dimensional case, but the potential use of multiple processing machines was strongly supported in his work. His approach was applied to a Multiple Instruction Multiple Data (MIMD) architecture, which allowed individual processors to be assigned individual tasks. Although, this approach has inherent bottlenecks such as load balancing each processor with equal work or time loss due to synchronizing all of the processors, the efficiency achieved proved to be a valuable approach.

1.7.3 Hustrulid's Approach

Hustrulid (1995) adapted a two-dimensional discrete element model to a 64 node T805 transputer from Alta Technology. In this study, 625 particles were used to simulate particle flow on a MIMD architecture. Although, 64 nodes were available, only 32 nodes were applied to the problem due to the excessive communication overhead. From this report, there appeared to be a small increase in speedup once 25 or more nodes were applied to the problem.

1.7.4 O'Connor's Approach

O'Connor (1995) developed a scheme known as the Discrete Function Representation (DFR) which is used to model the surface geometry of complex 3D objects. His scheme of contact detection was noted to yield speedups of up to two orders of magnitude when the sequential application of DFR was compared with another sequential technique. In addition, a number of MIMD machines were tested for their ability to a handle parallel sorting algorithms and were used in simulations modeling spherical particles. Although a possible application of an SIMD architecture was mentioned, this architecture was not tested in this study. Also the full capability of a supercomputer was not exploited, but was discussed as future research. This approach proved to be resourceful in modeling complex figures for potentially large systems ranging into the thousands of elements.

1.8 Research Program

This research studies the application of the discrete element method to a massively parallel SIMD machine. (No significant advances are known to be reported for a 3-dimensional discrete element model on this platform.) Since the majority of researchers can benefit from adapting their codes to a less complicated SIMD architecture, two modifications of program Trubal 1.51 were developed. The first major development was to adapt Trubal version 1.51 to the CM-2 machine, in order to exploit its entire architecture. The second major development was then to adapt it to the CM-5, its successor, for improved performance.

In this thesis, Chapter Two describes the approach used to develop the first version of TPM. The architecture of the CM-2 is explained in detail and the data structure of the Trubal 1.51 is presented to show the segments that were improved. Chapter Three discusses the second version of TPM that is applied to the CM-5, and discusses how improved speeds were obtained. Chapter Four is dedicated to simulations using the triaxial experiment performed by Rowe (1962). Chapter Five concludes with other approaches that are being taken to extend this research to other platforms.

CHAPTER 2

TPM VERSION 1.0

This chapter describes the computer implementation of the program Trubal to the massively parallel architecture of the CM-2. Trubal's data structure is analyzed for its inherent parallelism. In addition, the topology of the CM-2 is studied in order to consider the best implementation of Trubal. As a result, the progressive development of the TPM algorithm is described after repeated triaxial simulations are tested. Monitoring TPM's performance in each case led to various modifications, which are also presented in this chapter as part of TPM version 1.0. The final revision of this algorithm became TPM 2.0 when it was ported to the CM-5, which is described in the subsequent chapter.

2.1 Introduction

The demand for simulating larger granular assemblies with shorter run-time, motivated the adaptation of a discrete model to the connection machine, CM-2, with 32,768 processors. Meegoda and Washington (1993) proposed that if a connection machine (CM-2) was adapted to Trubal version 1.51, it could achieve a speedup of two orders of magnitude by taking advantage of CM-2's entire resource. However, there were a few obstacles that did not allow this goal to be reached. First, the CM-2's entire system resources were not readily available. Second, the overhead in processor communication would not allow the overall performance to reach two orders of magnitude. Third, the program development was retarded by the discontinuation of this machine. Nevertheless, the basic methodology that was pursued in developing an algorithm for the CM-2 is described in this chapter. The evolution of the CM-2 algorithms are noted as TPM 1.0 and the performance of the last two revisions are evaluated. The first draft of TPM 1.0 closely followed the data structure of Trubal version 1.51 and with minor modifications, adapted it to parallel programming. Before describing the TPM algorithm, an overview of Trubal's version 1.51 is presented.

2.2 General Description of Trubal's Algorithm

While there are many routines in the Trubal algorithm, there are six major routines (Cycle, Motion, Rebox, Search, Bbtest and Ford), that perform all of the basic calculations required in each time-step. The operation of the each routine is briefly described here. Readers are referred to Cundall et. al. (1978) for more detail. Figure 2.1 shows the flow chart of the routines involved in each time-step, including the Gen Routine which generates particles for the simulations.

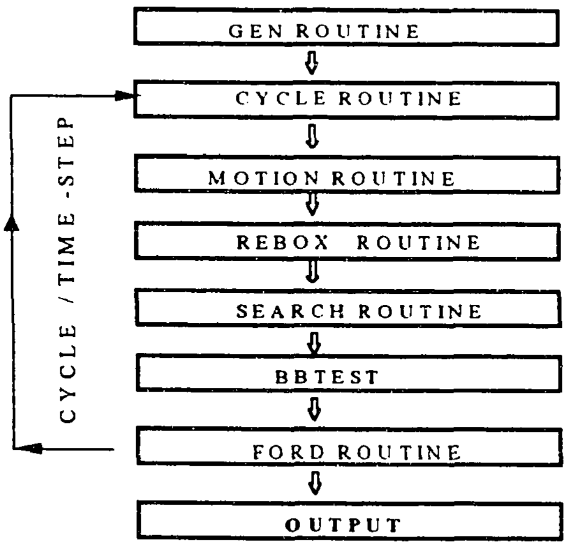


Figure 2.1 Flow chart for Trubal algorithm

2.2.1 Gen Routine

The Gen routine generates a number of spheres randomly using a random generating library found in most high level languages such as Fortran. Gen creates a random packing of spheres, by first considering a fixed free space and plotting points for the center of each sphere at random. After each sphere is located, contact detection is necessary to insure that there are no sphere overlaps. If a sphere is found overlapping another, it is deleted and another random point is selected. This is necessary to insure that there are no internal forces present prior to the packing of the entire assembly. When the requested number of spheres is generated, the boundaries are moved so that particles can make contact. In most cases, the boundaries are relaxed or expanded after the initial contraction in order to achieve this point of equilibrium.

2.2.2 Cycle Routine

The cycle routine updates all the global variable parameters, such as the current size of the boundaries and the global strain rates. It also resolves the global stresses and strains that currently exist in the assembly. One of its primary functions is to call the other subroutines that are required for each time-step.

2.2.3 Motion Routine

The motion routine is responsible for updating particle velocities, angular velocities, displacements and angular displacements, based on the current forces, moments, gravity effects and boundary conditions. Global damping (as described in Section 1.3) is implemented within this routine in order to provide stability to the entire system.

2.2.4 Rebox Routine

In order to reduce the number of contact checks between any two spheres, each sphere is centered within an imaginary cube and the entire assembly is partitioned into a specified number of boxes. This allows the number of contact checks to be confined within the various boxes, as opposed to checking each sphere with every other sphere within the assembly. The Rebox routine designates a memory address for each corner of the cube that encompasses a sphere (Fig 2.2). This address is checked by the Search routine and added to a memory listing described in a later section. In order for the boxing system to work efficiently, the size of the boxes is predetermined to be greater than the diameter of the largest sphere. It should be noted that contact detection is the most time intensive component of discrete element simulations, especially if the particles have complex shapes. Therefore, contact detection is only performed when a particle has moved a specified distance.

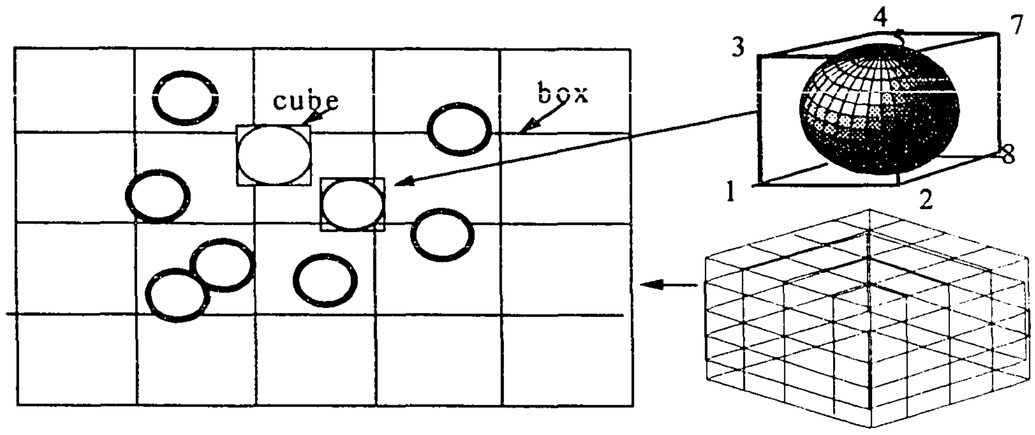


Figure 2.2 Spheres encompassed by cubes are mapped within boxes

2.2.5 Search Routine

The search routine allocates memory within the one dimensional data structure of Trubal (see Fig 2.3). The system, referred to as "leap frog" or "link and list", is a *dynamic memory* manager used to sort spheres within boxes with neighboring pointer addresses. Each sphere is listed along with a pointer, which denotes the memory location of the next sphere in the box . This system allows for systematic contact checking for each sphere located within the region of the box. If a sphere moves out of the box, it is deleted from the memory list and the memory space is reused for another sphere. The reader should note that a detailed representation of Trubal's data structure can be found in Cundall and Strack (1978).

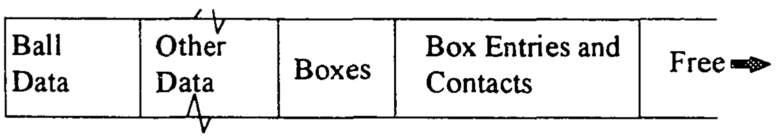


Figure 2.3 Trubal's data structure presented as a one dimensional array

2.2.6 Bbtest Routine

The BBTEST routine checks the distance between two spheres to determine whether the two spheres are in contact. For a spherical object, this is done simply by summing the radii together and subtracting it from the distances between the two centers. If a contact is formed between the two spheres or they were within a certain tolerance apart, a contact list is formed which is later analyzed in the Ford routine for possible internal particle forces.

2.2.7 Ford Routine

The Ford routine computes the shear and normal forces between two overlapping particles using their stiffness and relative angular and translational velocities (see Section 1.3). In addition, the resultant forces and moments are calculated for each particle based on the sum of its surrounding contacts with other particles. The global stress tensors are summed over all of the contacts existing within the assembly.

2.3 CM -2 Architecture

A parallel machine's architecture can be classified by its *multiple processing capabilities*, its *control systems* and its *interconnection network*. The CM-2's *multiple processing capabilities* attaches thousands of low performance processors together to work as a high performance homogenous architecture. The CM-2's *control system* is a Single Instruction Multiple Data Machine (SIMD) designed by the Thinking Machine Corporation. The SIMD architecture implies that each line of the code is executed in each processor before the next line can be read, therefore all operations are synchronized. Its 32,768 processors are divided into four primary partitions, all of which are controlled by a Unix-based Sun front end machine. The primary partitions, referred to as "sequencers", are not all dedicated to the user, but are assigned to jobs according to the system manager (see fig 2.4). Most commonly, one sequencer is timeshared while other sequencers run dedicated batch jobs. The Fortran 90 and C language are compatible high level languages used in operating parallel machines.

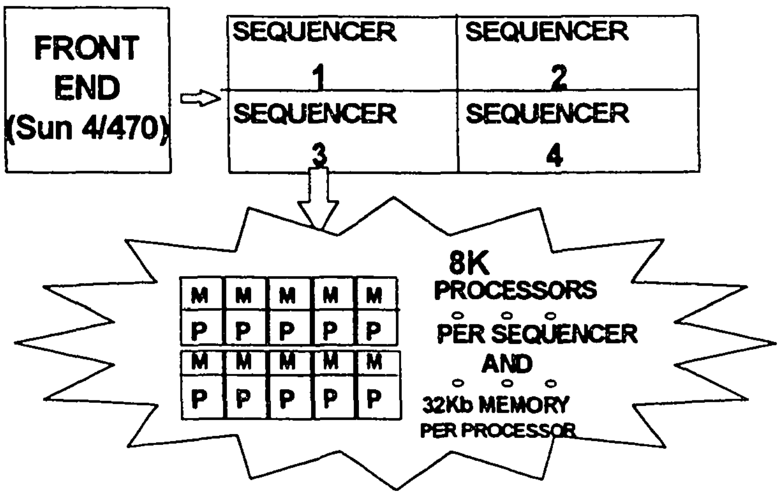


Figure 2.4 CM-2 Architecture

Even though all of the processors within the partition are operating, the number of processors that actually do useful calculations is dependent upon the layout of arrays used within the code. For example, the layout of 16 processors could be written as in Figure 2.5. This figure shows that when an array is declared, the machine automatically chooses the number and configuration of the processors required (i.e. 4x4 processors). Normally, the machine will select the smallest dimension for the configuration of processors, in order to shorten the distance of interprocessor communication. For example, "1x16" would not be selected since the greatest possible communication latency would be the along the second axis,"16". In Figure 2.5, the machine chose the 4x4 configuration for the best performance. In this system, the distributed memory allows each processor to operate on the assigned data in its attached memory. Although, the actual location of the processors and their memory remains unknown to the user, the processors can be visualized, in this case, as a square arrangement, with four processors in

```
Declaration Statement
real array(64,64)
cmf$layout array(news, news)

Possible Layout done by machine
Axis 0: 64(4 physical x 16 subgrid)
Axis 1: 64(4 physical x 16 subgrid)
```

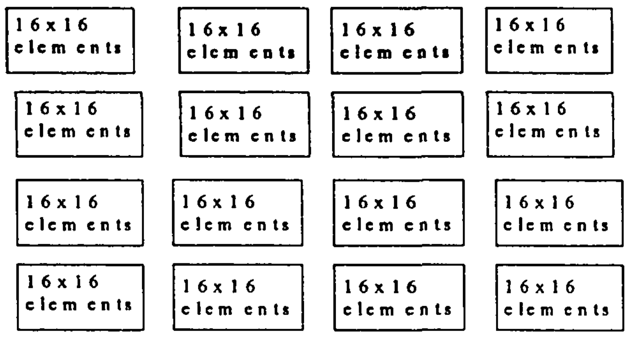


Figure 2.5 An array (64,64) layout over sixteen processor

each column and four in each row. The subgrid (as mentioned in Fig. 2.5) can be viewed as a 16x16 grid of elements inside of the memory of each processor. The "elements" refer to the data within each of the assigned arrays. If the machine allocated the array as described in the layout, then 256 elements of the array would be placed within each of the 16 processors (see Fig. 2.5). In this example, only 1 Kilobyte (Kb) of memory was required per processor; however the CM-2 allows for a maximum of 32Kbytes (256K bits) of memory for each processor, which is a physical limitation to the problem size that can be analyzed.

Finally, each of the processors can communicate with any of the other processors through the *interconnection network*. The *hypercube* network, denoted as a router in

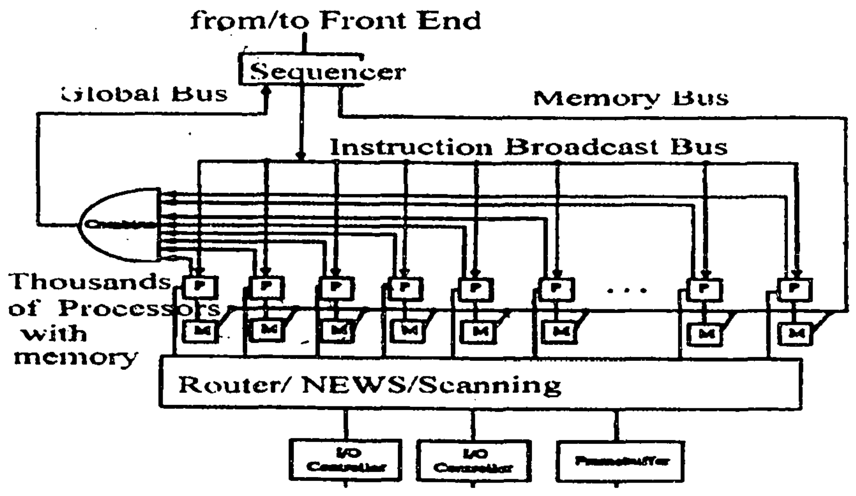
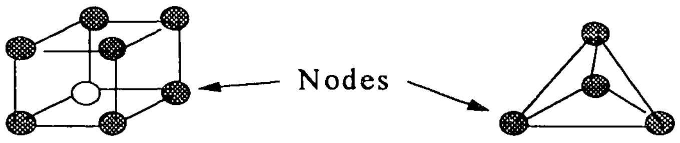


Figure 2.6 The CM-2 parallel processing unit

Figure 2.6, is the communication network used between the processors' memories. It is a very rapid interconnection network for data communication, because the *diameter* of the system is equal to its *node degree*. The *diameter* of the system is the shortest path between the two nodes that are the farthest apart, and the *node degree* is the number of channels or connections going into each node. The hypercube is a "n-cube" system, with an interconnection design that renders a relatively small diameter. For example, Figure 2.7a shows a 3-cube interconnection network, with three channels and eight nodes. In this case, the number of nodes, $N = 2^n$, so that the *node degree*, n, is equal to 3, therefore the diameter is also equal to three. In the case of the CM-2, each node is considered to be an individual processor along with its associated memory, so the entire system connected would require a 15-cube system for 32,768 (2^{15}) processors with a diameter of fifteen.



a) Hypercube - 3 -cube network b) Fully Connected network

Figure 2.7 Interconnect networks systems

The fastest interconnection network possible, is the fully connected network shown in Figure 2.7b, where the diameter of the system is one and any arbitrary node can communicate with any other node in one unit of time. However, this type of network is not practical for larger systems such as the CM-2, due to the number of connections that would have to be physically attached between every node. Considering the fact that the CM-2 can not implement a fully connected network, the hypercube network appears to be a feasible alternate for a relatively fast interconnect network.

As a result of the rapid growth in computer technology, the CM-2 machine is currently not available at supercomputer sites such as Pittsburgh Supercomputer Center (PSC) and the National Center for Supercomputing Applications (NCSA). Currently, the Pentium-100s and above are higher performing chips than the CM-2 processors used in this study. However, the parallelism of the algorithm still remains the governing factor as to whether the CM-2 can exceed the performance of a "high-end" desk top computer.

2.4 Data Structure of TPM Version 1.0

As mentioned earlier, the initial idea proposed by Meegoda and Washington (1993) for TPM version 1.0 was based on two criteria. The first criterion was that if each particle within the assembly was stored in its own processor along with the necessary data pertaining to all the other particles, each processor could perform its particle operations with its neighboring contacts. Hence, the latency of interprocessor communication would be reduced. The second criterion was that since dynamic memory (i.e. the dynamic memory mentioned in Section 2.2.5) is unsuitable for SIMD machines, then it was necessary to create a more *static memory arrangement*. Recall that the Search routine, described in Section 2.2.5, was responsible for the *dynamic memory management* of Trubal, which has been replaced by a *static memory arrangement* in TPM's reboxing routine. Therefore, Trubal's "link and list" was removed from the algorithm and was replaced with a large three dimensional array representing TPM's rebox routine, described later on in this section. The algorithm for TPM version 1.0 has the same flow chart as Trubal 1.51, except that the search routine was eliminated (see Fig 2.8). The following altered data structure of the TPM algorithm is described for each of the major routines.

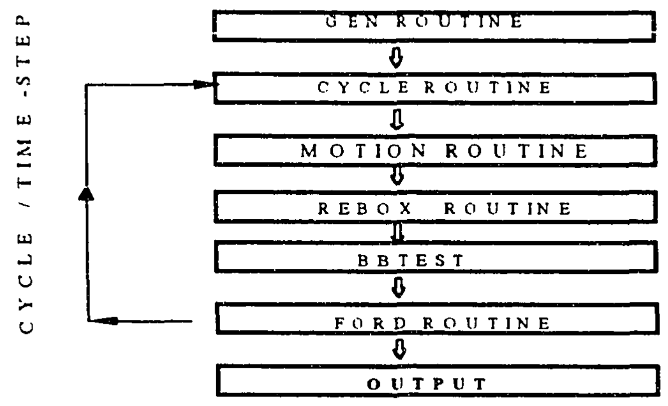


Figure 2.8 Flow chart for TPM version 1.0

First, the *gen* routine and the *cycle* routine are the same as in Trubal 1.51, except that *gen* simultaneously produces random sphere locations for the entire assembly with a random generating function. The *motion* routine uses a 2-D array, referred to as "sphere array", which places the information for each sphere into the memory of each processor as shown in Figure 2.9. By comparing Figure 2.9 to Figure 2.3 and Figure 2.5, the evolution of this scheme can be described. In Figure 2.3, all of the information required for Trubal 1.51 is stored within a one-dimensional array and it is accessed as *shared memory*. In Figure 2.5, an example of the data layout within a two dimensional array is shown, so that the parameters for each sphere in Figure 2.9 can be allocated within a common memory space. One axis of the sphere array was assigned to each particle in the assembly and the other axis was assigned to the various parameters pertaining to a particular particle. Since the arrays determine the data storage for each processor based on the array layout, a single array was used to store information pertaining to the sphere parameters.

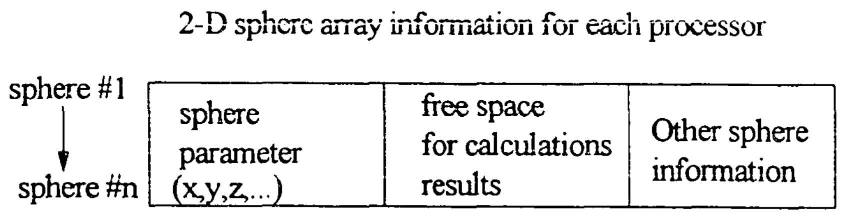

2-D sphere array information for each processor
sphere #1
sphere #n
sphere parameter (x,y,z,...)
free space for calculations results
Other sphere information

Figure 2.9 Data structure for the arrays of TPM version 1.0

The *rebox* routine, as described in Section 2.1.4, was handled by a set of processors that were assigned to a 3-D "box array". One axis of the array represented each particle within the assembly and another axis stored the box number for each of the eight corners encompassing the particle (see Fig 2.2). The third axis of the box array is the number of boxes in the assembly and it stores the sphere corners (refer to Figure 2.2) that map into that memory location relating to its box.

In order to perform this distribution of data within the array, a number of CM libraries were used. First, after the box number was calculated from the corners of the cube encompassing the sphere (see Cundall et. al. 1978), it produced repetitious box numbers. To explain TPM's reboxing scheme, a sample of two spheres encompassed by a cube is drawn within four boxes as noted in Figure 2.10. The 3-D boxing array would originally appear as shown in Figure 2.11, where all box numbers would appear in box #1.

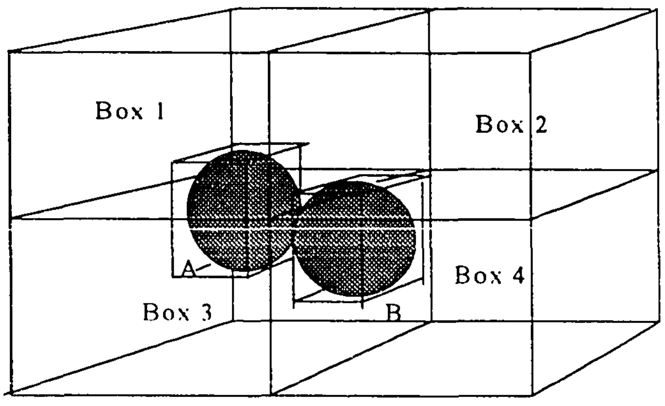


Figure 2.10 An example of two spheres within four boxes

A "spread" command is used to duplicate the box numbers for the eight corner points into the third axis. This means that the numbers that appear in Box#1 in Figure 2.11, would appear the same way in the remaing boxes (i.e. Box #2, Box#3,& Box#4). At this point, a "forall" command is used to select the box numbers that are residing in the correct

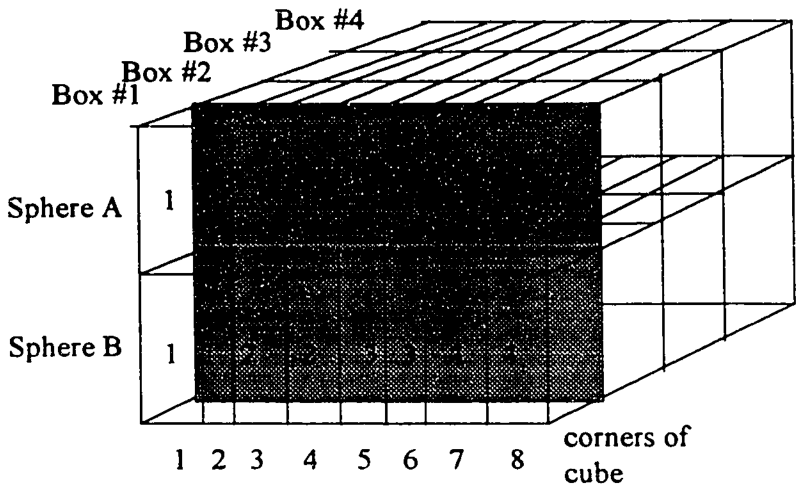


Figure 2.11 TPM's Box Array Initial Setup

memory location, and to replace the correct box numbers with the correct sphere names within each box. In the example presented in Figure 2.10, the sphere names are the letters "A" and "B", so that a distinction between boxes and spheres can be made; however in the actual algorithm, the spheres are assigned numbers. A "firstloc" and "where" command is then used to eliminate the duplicate sphere names that appear for the same box. First, the "firstloc" commands *flags* the first location of the sphere number within each box. Second, the "where" command places a *dummy value* wherever the *flag* is false. At this stage, TPM's box array appears as shown in Figure 2.12 without the dummy variable for clarity. Finally, the "pack" command consolidates the box numbers into a single column of elements for each box. At this point, each box has a list of spheres in which it can permutate all possible contacts. Then two arrays containing a list of all possible contacts is matched with the box list using a "forall" command and when the list matches, it then transfers the contact pairs into the contact vector arrays. In this

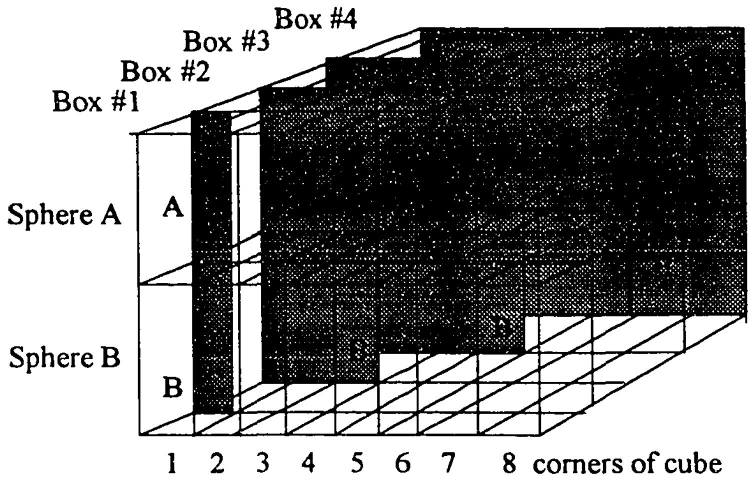


Figure 2.12 TPM's Box Array after "Spread" command and "Forall" command is used

example, the only permutation possible is between "A" and "B" , so in this case these contacts would be found in box#1 and box#3. The duplication of the same contact found here is eliminated as the algorithm flags all the contacts within a *contact map*. This map contains one of every possible contact that can be permutated within the algorithm. Once the *contact map* is flagged by the list of all possible contacts, its information is transfered to the two one-dimensional arrays shown in Figure 2.9.

The *bbtest* routine performs the same contact checking function as in Section 2.1.6, except that it is performed in parallel. The *bbtest* routine use the two single dimensional contact arrays (see Figure 2.13), because the CM-2 requires a one

One Dimensional Contact Array "X"

particle #A	Other particle names

One Dimensional Contact Array "Y"

particle #B	particle names that pair off with array "X"

Figure 2.13 One dimensional contact arrays for TPM 1.0

dimensional array as a vector or pointer, when it uses as an argument within another array. In this case, two spheres in contact were stored in two arrays so that when lines of instruction were invoked for the contact checking as well as the force/displacement calculations, only the information for each pair was used within the calculation.. In this fashion, the entire contact check and force/displacement operations are executed in parallel. The following example demonstrates how the difference in sphere center distances could be calculated between all the contacting spheres within the assembly in one line of instruction:

dist(:)=sphere(center, vector X(:))- sphere(center, vector Y(:))

where (:), implies all the elements within the array, *dist* is the distance between center of spheres, vector #1 and #2 represents the contact arrays with a list of spheres in possible contact, *center* represents the coordinates of the center of a sphere, and the *sphere* array is the global array with all of the various sphere information. The Trubal algorithm, on the other hand, repeats this line for each contact within the assembly. Finally, the *ford* routine uses the contacts that are flagged as overlapping, to perform the parallel force/displacement operations that are described in Section 2.1.7.

2.5 Advantages of TPM's Data Structure

The data structure described has many advantages for parallel processing. The sphere array, box array, and contact array are constructed so that parallel operations could be performed in the motioning of the particles, mapping of the particles and the detection of contacts respectively. TPM's motioning of spheres took advantage of parallelism, by displacing all spheres based on their current velocities and boundary conditions simultaneously, in contrast to Trubal's individual operation for each sphere. In this case, the two algorithms (Trubal and TPM) used the same lines of instructions (with exception to the names of the array), but the CM-2 performed these lines for the entire assembly without looping for each particle.

For example, in the original Trubal, the corner of each cube encompassing the spheres (8 points) was mapped into boxes, one at a time. This operation took several lines of instructions and two different routines. Recall that the *rebox* routine determined the coordinates of each point of a cube encompassing a sphere, and the *search* routine used a "link and list" system to store the point in the correct box location. However, in this modification of TPM, the several lines required for determining the coordinates were performed for all of the spheres simultaneously. Each corner point was located in the correct box by a few CM library commands. In addition, the link and list was eliminated in this version of TPM, because the contact listings were stored in the contact arrays.

2.6 Evaluation of Results of TPM Version 1.0

A triaxial simulation containing one hundred and fifty spheres was performed using TPM version 1.0 with the data structure denoted in Figure 2.9, and the results were compared to the VAX 8800 performance of the same simulation. The three dimensional simulation qualitatively modeled the consolidation behavior of a dry granular sample under cyclic loading conditions. Initially, the sample was randomly generated and then compacted to a state of equilibrium. Two cycles of loading were performed on the sample during the simulation, and then it was strained to failure. Failure in this case, was defined at the point where the strain continued to increase at the same level of stress. Although, an actual experiment was not performed, this fictitious simulation was used as a means of comparison between the performance of a parallel algorithm (TPM) and a serial algorithm (Trubal). Therefore, the input parameters were values that simply allowed for a stable simulation. Because a complete simulation was not performed on the CM-2, the results of this simulation are described in detail for TPM version 2.0, where an improved performance was achieved.

A complete simulation of this triaxial experiment lasted for 20,000 cycles on the VAX 8800 and it required twenty-five minutes to perform this task. However, after 200 cycles TPM version 1.0 had already used close to six minutes, which is only enough time to form two contacts in the initial compaction of the assembly. At that point, the simulation was terminated based on the fact that 20,000 cycles was desired to complete the simulation and it was assumed that TPM would require approximately 6 minutes per 200 cycles. Therefore, TPM's performance was not comparable with the VAX 8800's time. Table 2.1 shows the results of the system resource usage as well as routines with the highest time consumption.

The bottlenecks within the TPM algorithm became evident through the use of the CM debugging software, PRISM, which produces a performance evaluation as shown in Table 2.1. The front end machine denoted as (FE) is able to operate simultaneously

Table 2.1 CM-2 Resource usage viewed in PRISM

	Total time: 390.24 sec		
FE cpu (user)	278.651s		
FE cpu (system)	2.780s		
FE I/O	14.200s	CYCLE	7.332 s
FE Total	295.631s	MOTION	5.355 s
		REBOX	181.21s
CM cpu(user)	142.471s	BBTEST	22.374s
CM cpu (system)	39.938s	FORD	57.227s
Comm (Send/Get)	120.404s		
Comm (NEWS)	25.139s		
Comm (Reductions)	18.393s	150 particles -200 cycles	
Comm (FE<->CM)	1.392s		
CM not profiled			
CM I/O			
CM Total	347.737s		

with the connection machine denoted as (CM). The CM, in this case, governed the time that was required to complete these cycles, causing the "FE cpu user " time to idle for a longer period of time. This is an inherent result of the SIMD architecture which synchronizes the system at each level of instruction. Therefore, only the times for the CM need to be analyzed, based on its operation during each routine.

The *rebox* routine is clearly the most time-intensive, due to the fact that it is responsible for the contact detection. As mentioned earlier, the information assigned to the spheres in each processor is globally reduced to the arrays that map the spheres into boxes. This is seen in the "Comm (Reductions)" time usage, but also sorting of the spheres into the correct boxes requires a great deal of "Comm (Send/Get)" time, which is indicative of interprocessor communication. Even with the parallelism of the boxing routine as described earlier, the amount of sorting and consolidating of data elements between the processors was time-intensive for both processor communication as well as cpu usage. Evidently, the sorting and consolidating of the box information are best performed using the local memory of a processor, because the dynamic randomness of particle movement caused data elements to be placed sporadically within processor memory, loosing the advantage of parallelism.

The unpredicted behavior of this type of modeling, is reflected in the data flowing into the *ford* routine. The *ford* routine selects the contacts that were formed randomly and were *globally reduced* from larger arrays. These large arrays contained information required for each sphere that was found in contact. Once the data elements are transferred to their correct memory location, the parallel computation is then performed rather quickly.

The *bbtest* routine, which checks the distances between spheres, requires information from different processors within the sphere array (such as coordinates, incremental displacements, etc.). As a result, the interprocessor communication time is required for the *contact vector* array to point to the memory of the data that is needed and transfer this information to the processor that is performing that particular calculation. Again, the randomness of the vector array information can increase the communication time between the processors.

Regardless of the various operations just mentioned, the "CM cpu (system)" time consumption always requires time to perform its internal functions that are hidden from the user. The system may require different internal algorithms to perform basic memory management, library functions or network protocols for the processors' communication. The following example demonstrates that if data from one array is being used within a computation with another array, the system will create a buffer to copy the information of that array, before changing the data within that array.

```
A(:,:)=A(:,:)+B(:,:)
```

If array A and B are not within the same processors, then interprocessor communications protocols within the system will be required.

2.7 Discussion and Other Modifications

From studying the topology of any parallel architecture, it is understood that the best performance can be obtained if all calculations are local within each processor. Since the fastest access to data occurs locally within each processor's memory, the speed of the interconnection network can not exceed this. Of course, the second fastest data access is the communication between two neighboring processors/nodes that require just one unit of time. The development of TPM version 1.0 took advantage of the local memory speed, but failed to optimize its global communication, which led to a series of bottlenecks.

Due to the large memory requirement for the array layouts, scaling to a larger problem became inconceivable, therefore an attempt was made to allocate dynamic arrays which would have had the advantage of being created during the execution of the program and deleted when the array was unneeded. This modification, although very practical , proved to be unfruitful as far as the time consumption on the system for memory management. The CM-2 required additional libraries to construct and deconstruct these dynamic arrays which inevitably defeated the overall purpose.

Once it was recognized that using a single array (i.e. sphere array) did not guarantee that data would be entered into the desired processor, another modification was made assigning each parameter to an array with the same size and dimensions. This was the stipulation in the CM-2 architecture which guaranteed local memory between arrays. This was the last modification that lead to a better performance and to the next generation of TPM on the CM-5.

2.8 The Last Modification of TPM 1.0

TPM version 1.0 last modification included the use of separate arrays for each parameter of the particles (to localize memory) and considered all possible contacts within the assembly to eliminate the boxing routine. Hence, a better performance was achieved using the same simulation. Because the CM-2 was discontinued during the application of this data structure, the preliminary results for the simulation that was tested were used to extrapolate the time for a complete simulation. Meegoda and Washington (1994) estimated that the CM-2 could perform up to five times faster than the VAX 8800 when executed for a 20,000 cycle simulation. The complete simulation time for the CM-2 was approximated to finish in five minutes from the data collected after 200 cycles of actual run-time. This calculation was made assuming the work required by the system was the same throughout the simulation and that it would take an average of 3 seconds for every 200 cycles. Since 20,000 cycles were anticipated, that meant 300 seconds would be needed to complete the simulation on the CM-2. Because the matching time for the VAX 8800 was noted at twenty-five minutes, it was believed that an improved speedup of up to five folds could be achieved.

Since the performance of simulations are relatively architecture dependent, a comparison between the CM-2 and the CM-5 results may not be valid. However, the CM-5 results does suggest that there can be a slight variation of time per 200 cycle. When this same simulation was performed on the CM-5, the first 200 cycles took 2.5 seconds which was very close to the time just mentioned. However, the time increased to 3.7 seconds when the assembly reached its maximum consolidation with a coordination of six (six contacts per particle). The time for 200 cycles increased again to 3.9 seconds, when the triaxial axial load was applied. Therefore, the CM-2 speedup should be noted as an upper bound when one sequencer is used. Since the CM-2 machine was upgraded to the CM-5, it was deemed that any further investigation of TPM on the CM-2 would be overshadowed by the advancement of the CM-5 architecture. Table 2.2 summarizes the results of the VAX 8800 and the CM-2.

Table 2.2 Comparison of Timing between CM-2 and VAX 8800

Machine Architecture		Time Scale for 150 particles (20,000 cycle simulation)
CM-2 (TPM)	with one global sphere array and boxing routine	6 minutes per 200 cycles estimated 10 hours for 20,000 cycles
CM -2 (TPM)	with separate arrays for each parameter and all to all contact checking	3 seconds per 200 cycles estimated 5minutes for 20,000 cycles
VAX 8800 (Trubal 1.51)		25 minutes for 20,000 cycles

2.9 Conclusion

In conclusion, TPM version 1.0 has a number of parallel applications built into the algorithm, but the bottlenecks due to incompatible array sizes, cause an excessive amount of communication protocols. Also, in the earlier modifications of the TPM, the dynamic behavior of the assembly demanded extra cpu-time for selecting random data from different processors. Single Instruction Multiple Data machines perform best on data structures with static memory structure, which means that data is not shifted from its memory location and computations can be performed within local processor memory. However, in the earlier stages of TPM's development, excessive array reductions slowed down the data communication considerably. Inevitably, TPM 1.0 evolved into a workable algorithm introduced in the next chapter as TPM 2.0. TPM 1.0's last modification rendered a promising performance, although the results were not entirely conclusive.

This early approach proved to be a landmark in current attempts to parallelize discrete models on SIMD massively parallel supercomputers. The discontinuation of this machine had considerable effects on this research, however many of the ideas that are currently being developed were a result of this earlier project.

CHAPTER 3

TPM VERSION 2.0

This chapter describes the transition of the TPM algorithm from the CM-2 machine to the CM-5 machine. The CM-5 architecture advances are described and the exploitation of the system is discussed. TPM version 2.0 adaptation to the CM-5 is included with a complete simulation of a triaxial test, which is studied for accuracy and speedup.

3.1 Introduction

Since the architecture of the Connection Machine (CM-5) replaced the CM-2 machines, TPM version 2.0 was implemented to the CM-5, since 1) the CM-2 ceased to be available for this research and 2) the upgraded CM-5 became available for porting the code. To improve the performance of TPM 1.0, two modifications were made as mentioned in the last chapter. First, the boxing routine was removed entirely. This meant that the number of contact checks increased to the full size of the assembly, squaring the problem size. Secondly, a multiple number of contacts were clustered into the same processor, by using the same array size declarations and this was to eliminate the bottleneck of array reduction found in TPM version 1.0. The new memory arrangement robustly handled all of the calculations and contact detection simultaneously. One further advantage is that unlike all the other schemes, all particles were motioned, checked for contacts, and calculated for interparticle forces within their own processor without the need for interprocessor communication. However, this was not the case with the global calculations that required communication extend to all the processors.

In this chapter, the architecture of the CM-5 is first described and compared to the CM-2. Then the new TPM version 2.0 is discussed and it is shown how the squared problem size, due to an all to all contact pairing within the algorithm, accelerated the speed of the simulation. Finally, a similar simulation to the one described in Chapter 2 is rerun to validate the modifications to the algorithm.

3.2 CM-5 Architecture

The CM-5 architecture was the Thinking Machine Corporation's last generation of massively parallel machines. This company, although still successful in marketing literature and software for their existing machines, discontinued the manufacturing of supercomputers due to bankruptcy at the end of 1994. However, many other companies, such as Kendall Square Research (KSR), which manufactured very good high performance supercomputers suffered the same fate. Since the termination of these companies are not uncommon, the quality and performance of the machine's design should not be reflected, but rather other factors, such as limited demand and poor marketing. In fact, the CM-5

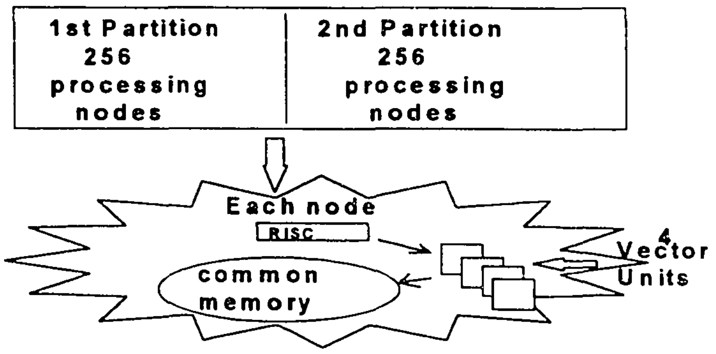

1st Partition 256 processing nodes
2nd Partition 256 processing nodes
Each node
RISC
common memory
4 Vector Units

Figure 3.1 CM-5 Architecture

was a great improvement over the CM-2 for a number of reasons, which can be seen within the three basic components of the architecture (the *multiple processing capability*, the *control system*, and the *interconnection network*).

The CM-5's *multiple processing capabilities* were designed differently from those of the CM-2. In contrast to the 32,768 processors found in the CM-2, there are only 512 nodes in the CM-5 (see Figure 3.1). Recall that each node for the CM-2 consisted of a low performance processor attached to its own memory (Figure 2.5). In the case of the CM-5, each node consists of a high performance RISC processor (performing at 5 Mflops/32Mhz) attached to four vector processors (performing at 128Mflops/16Mhz). The RISC processor is able to perform all of the operations that the front-end performed for the CM-2, only in this case it controls a single node instead of the entire machine. The RISC processor's performance can be comparable to an IBM workstation's performance since the Thinking Machine Corporation decided to use better performing IBM microprocessors than the one used in the CM-2. The vector unit's design shown in Figure 3.2 is able to perform computations (with the Arithmetic Logic Unit), as well

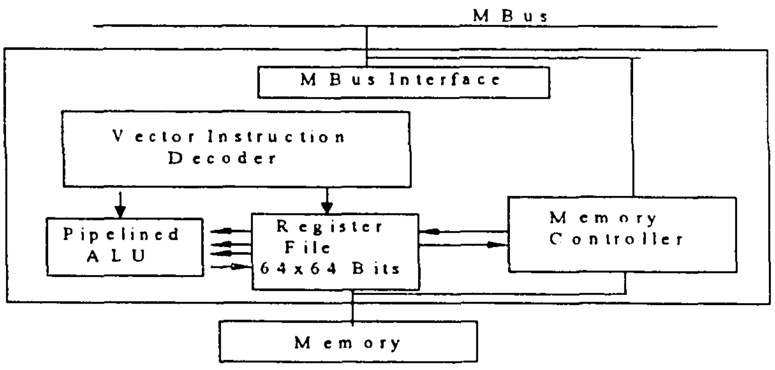


Figure 3.2 Vector unit functional architecture

as receive instructions (with the Vector Instruction Decoder) and manage memory (with the memory controller). Each vector unit is allocated 8Mb of memory and is able to share its memory with the other vector units within that node. This 32Mb of shared memory is an important factor in reducing the latency of data communication. The vector units are comparable with those of the CM-2 processing node because they must receive instructions before they can perform any calculations or memory management.

Secondly, the *control system* of the CM-5 is more advanced than that of the CM-2. All of the CM-5 nodes are interconnected by three major networks: the data network, the control network, and the diagnostic network (see Figure 3.3). The control network is

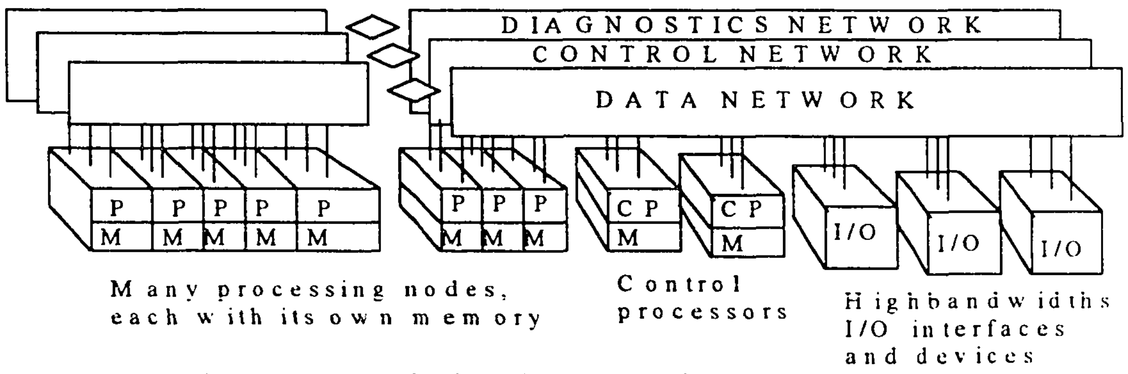

DIAGNOSTICS NETWORK
CONTROL NETWORK
DATA NETWORK
P
M
CP
M
I/O
Many processing nodes, each with its own memory
Control processors
High bandwidths I/O interfaces and devices

Figure 3.3 Organization of the Connection Machine (CM-5)

used for operations that involve all of the nodes at once, such as synchronization operations and broadcasting (SIMD). The data network is responsible for bulk data transfers where each item has a single source and destination. It is this feature of the data network that enables the architecture to perform as a Multiple Instruction Multiple Data (MIMD) machine. The diagnostic network is used by the system manager for maintaining the system in case of hardware failure or malfunction. The "control processor" is another RISC microprocessor which is also referred to as the "partition manager". When an SIMD code is submitted to the control processor, it copies the program into each node, and all of the nodes synchronously perform each operation one line at a time using the control network. When a MIMD code is submitted to the control processor, each node gets an address and performs that part of the code that is assigned to its address. In this case, it uses the data network for its interprocessor communication.

The *interconnection network* of this system is within the control network and the data network. The control network and the data network are a binary tree and fat tree interconnection network respectively. The diameter of the binary tree is D=2(h-1) where h=$\log_2[N+1]$ is the tree height and N is the number of nodes. In Figure 3.4, the tree height, h, is four and the diameter, D, is six. For the CM-5 design with 512 nodes, the tree height, h, is nine and the diameter of the system, D, is sixteen. Recall that the diameter of the CM-2 was fifteen for the entire architecture, which makes the speeds of

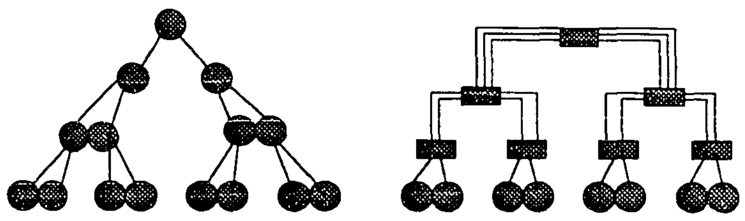

a) Binary Tree b)Fat Tree

Figure 3.4 Binary and Fat tree interconnection network for 15 node system

the interconnect network between the two machines quite comparable. The fat tree in comparison with the binary tree is designed to produce fewer bottlenecks in communication as messages are sent to higher levels of the tree. The binary tree will inherently be slower, because there are less physical channel connections for data flowing to the higher levels of the tree. Therefore the MIMD global communication is preferred over the SIMD global communication. Hence, in TPM version 2.0, the greatest time consumption was detected in the global communication.

. A comparison between the CM-2 and the CM-5 architecture reviewed in this thesis is given in Table 3.1. It is found that although the interconnection network has comparable speeds, the CM-5's hardware is more advanced. Since the CM-5 has a larger amount of shared memory within its node, less communication between processors is expected.

Table 3.1 Comparison between the CM-2 and the CM-5

Attributes	CM-2	CM-5
processor trademark	Weitek	RISC (IBM)
# of processors	32,678 procs	512 nodes
memory capacity	1 Gbyte	16Gbyte
interconnection network	hybercube	Binary tree & Fat tree
control system	SIMD	SIMD/MIMD
memory per processor/node	32 Kbytes/processor	32Mbytes/node
Diameter of interconnection network	15	16
speed	5.6 Gigaflops for entire system	128 Megaflops per node

3.3 TPM Version 2.0

TPM version 2.0 was based on Trubal version 1.51, but eliminated the *rebox* and *search* routines. It also renamed the global communication portion of the *ford* routine as the *globe* routine. As a result, the flow chart shown in figure 3.5 depicts all of the routines

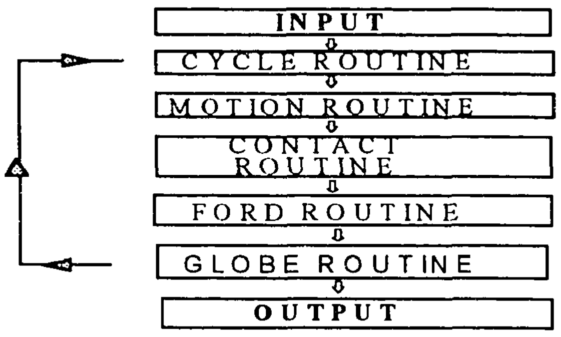


Figure 3.5 Flow chart for TPM version 2.0

that are involved in a cycle or time-step of TPM version 2.0. The *contact* routine is the equivalent of bbtest, and the *globe* routine handles the global calculations required for the forces and moments of each particle as well as the global stress tensors.

The idea used in TPM version 2.0, is to permutate every possible contact existing within the assembly, without having to use do-loops or having to create a dynamic memory arrangement within the algorithm. Therefore, within the square of the problem size,n, an nxn matrix will produce twice the number of contact possibilities in the lower diagonal and the upper diagonal. The calculations required for the global stresses from the contacts within the assembly use just one of these diagonals, in order to avoid the duplication of contacts. However, the force and moment summations require the full row or column of a matrix depending on whether the matrix or it's transpose is calculated. For example, if each array represents a sphere parameter for the matrix of the entire assembly, and, for example, V represents the velocity and V^T represent the transpose, so then

$$V = \begin{bmatrix} v11 & v12 & \dots & v1n \\ v21 & v22 & \dots & v2n \\ v31 & v32 & \dots & v3n \\ vn1 & vn2 & \dots & vnn \end{bmatrix} \text{ and } V^T = \begin{bmatrix} v11 & v21 & \dots & vn1 \\ v12 & v22 & \dots & vn2 \\ v13 & v23 & \dots & vn3 \\ v1n & v2n & \dots & vnn \end{bmatrix} \qquad (3.1)$$

where *n* represents the number of spheres in the assembly . V_{ij} is the velocity for the ith sphere and the j represents the velocity of the jth sphere found in the corresponding location of V_{ij}^T. The *motion* routine computes equation 1.26 for both V and V^T. This equation can now be written as follows,

$$[V] = \frac{[V](1 - \alpha \Delta t / 2) + [F]\Delta t / [m]}{(1 + \alpha \Delta t / 2)} \qquad (3.2)$$

and,

$$[V^T] = \frac{[V^T](1 - \alpha \Delta t / 2) + [F^T]\Delta t / [m^T]}{(1 + \alpha \Delta t / 2)} \qquad (3.3)$$

where *F* and *m* are the force and mass matrices for the entire assembly, and F^T and m^T are the transposes of those matrices respectively. Because the architecture is able to broadcast scalar values intrinsically, they can be used within the equation without having to be replicated in a matrix for parallel computation. Table 3.2, depicts how TPM version

2.0 implements the scheme of squaring the problem size to adapt to the CM-5. In the table, *Mtrans* represents the transpose matrix of the moment values found in *M*, and the same is true for the force, *F*, velocity, *V*, and so forth.

Table 3.2 Declaration statements and psuedocode flow chart for TPM version 2.0.

```
 cmf$layout M(n,n),Mtrans(n,n)
 cmf$layout F(n,n),Ftrans(n,n)
 cmf$layout V(n,n),Vtrans(n,n)
 cmf$layout X(n,n),Xtrans(n,n)
 cmf$layout (for all parameters)
get all contact pair parameters in corresponding array locations
for all numbers of cycles do
  motion each sphere using updated coordinates
  check for contacts between coordinates of A and B
  Calculate overlapping contact pairs for all forces and moments
  sum up all global stress tensors and sum forces/moments
enddo
print output
stop
```

During the operation of the *motion* routine, extra computations are required for the transpose of each matrix, which increases the amount of work required for the processors. Even extra memory is required to create the transpose matrices for these calculations, which leads to an inefficiency within the algorithm. However since there is no data dependency, all of the operations are performed in parallel with a significant speedup of time.

The *contact checking* routine is able to check for contacts without boxing since the matrices are already arranged in a permutation. Since all of the parameter arrays have the same array sizes, each corresponding pair of spheres is computed by the same processor within a shared memory. After the contacts are flagged within the matrix, there is no further need to use transpose matrices. Now the matrices represent contact pair values corresponding to its location within the matrix. Table 3.3 demonstrates how the connection machine facilitates this operation using the *where* command with a logical

Table 3.3 Pseudocode for *Contact* Routine in TPM version 2.0

```
CONTACT ROUTINE
check all coordinates for periodic space
distx=x-x'
disty=y-y'
distz=z-z'
dist=squareroot($(distx)^2$ + $(disty)^2$ + $(distz)^2$ )
radius sum=rad+rad'
where(dist-radius sum.lt.0)contact=.TRUE.
return
```

array. Assuming *contact* is a logical array size equal to the size of the other parameter arrays, all of the processors containing physical contacts will be flagged by an element of *contact.* Parallel contact detection in TPM has a major advantage over other discrete model algorithms, because it eliminates "link and lists" and various permutation schemes.

The *ford* routine performs calculations for all of the elements in the matrix, regardless of whether or not a physical contact exists. Since SIMD architecture inherently requires that all processors assigned to the front end must perform each instruction simultaneously, there are no idling processors. If a processor is not given any data, it continues to perform the operation on *padded* data produced by the machine. Therefore, erroneous results produced by non-contacting spheres are eliminated in the *globe* routine by contact flagging. In order to make a proper performance evaluation, a test was performed using a subset of the matrix, calculating only the flagged elements in the *contact* array. Instead of an accelerated time performance as anticipated, a slower time was achieved. The architecture required extra time to eliminate the results that were not flagged and replace them with *padded* elements to be ignored. However, when another array was created to match the same number of contacts that existed, the reduce array size had a better performance. Hence, when an array is very large, more work is required from each processor to select computations for certain data elements. Although creating a smaller array from the number of contacts improves computation time, TPM version 1.0 proved that transferring data to a smaller array (global reduction) causes a major bottleneck for algorithms requiring dynamic memory.

Table 3.4 Psuedocode for the Globe routine in TPM version 2.0

```
GLOBE ROUTINE
Set up all flags for contacts in upper triangle of matrix (upcon)
Set up flags for contacts using entire matrix (contact)
Flag the diagonal as false for both (contact) and (upcon) matrix
globalstress=sum(stress, upcon=.true.)
Force(1,:)=sum(interparticle force, contact=.true.)
Moment(1,:)=sum(interparticle moment, contact=.true)
Force prime=Spread(Force(1,:),dim=2)
Moment prime = Spread(Moment(1,:), dim=2)
return
```

In the *globe* routine, global *reduction* and *broadcasting* uses the *contact* array to select the correct results at the end of each time-step. Table 3.3 demonstrates the advantage of having a squared matrix data structure. When global stress tensors' calculations are made, only the upper triangular portion of the matrix is needed to avoid duplication of the stresses. The *sum* function is the intrinsic library function that uses the flag (denoted as *upcon* in table 3.3) across the *control* network to globally reduce the various contact stresses into the nine stress tensors for the global assembly. The *sum* function is also able to use the flag (denoted as *contact* in table 3.3) for the entire matrix, since each sphere is represented with regard to every other sphere by each row of the matrix. Since the diagonal of the matrix represents the sphere in contact with itself, it is automatically eliminated from all calculations. The *spread* function is a global *broadcast* intrinsic library function that also uses the control network to regenerate the transpose of the force and moment matrices required for the next time-step in the *motion* routine.

The *globe* routine is clearly the source for any bottlenecks in TPM version 2.0, since it introduces global reduction and broadcasting among all of the processors and it flags certain elements for computations. Speed is even further retarded by the time needed to serially print out results that are layed out in parallel on the machine. The bottlenecks as just described were found to be inherent bottlenecks of massively parallel machines with unshared memory capacity. Some of the difficulties that were discovered in the two versions of TPM were a result of architecture design and are gradually being dealt with by the manufacturers of these machines.

3.4 Experimental Verification and Performance Evaluation

The TPM algorithm was validated by simulating a triaxial test that had previously been simulated in the original Trubal and TPM version 1.0. It should be noted that the data structure in TPM was modified to obtain a more efficient performance on the CM-5 platform. In this case, a similar test was conducted with Trubal running on one of the host processors of the CM-5, and with TPM operating on thirty-two nodes of the same machine. The objective of this test was to see if a discrete element model could obtain a speedup due to the new data structure and if it could be scaled up to a larger problem. A triaxial simulation was conducted with the input parameters shown in Figure 3.6. As stated in the last chapter, these values were selected on an arbitrary basis with the premise that the stability of the simulation would be achieved. The complete simulation involves two stages.

The first stage involves generating random locations for one hundred and fifty uniform spheres. Please note that once the locations were generated, they were stored into an input file and the same coordinates for each of the spheres were used in both TPM and Trubal simulations. The assembly was allowed to compact under isotropic conditions until the average particle had six different contacts. The stresses within the assembly reached very high values at this stage of the simulation as noted in Figure 3.6. Afterwards, the assembly was relaxed by reducing all of the boundary's strain rates to zero. This allowed the assembly to dissipate the internal stresses as it approached a state of equilibrium and it also reduced the amount of particle overlap as the particles rearranged themselves to handle the stresses. It took 5200 cycles to reach a point of equilibrium so that the compacted sample could then be used to simulate a triaxial test.

The second stage involved emulating the response curve of a typical triaxial test. Please note that each cycle represented one time step within the time integration of the

DEM. The vertical stain rate was introduced at the boundary while the lateral strain rate remained constant. The sample was vertically strained to failure at 10,600 cycles. In this case, failure was depicted as a reduction in the stress that is needed to strain the sample. Afterwards, the sample was unloaded (13,825 cycles) and then reloaded a second time under the same stress conditions and a similar stress path history was obtained. The simulation was stopped at 18,600 cycles after the sample had failed again. Figure 3.6 shows the stress vs. strain curve of the simulation (sigma 11 vs. strain 11), where the

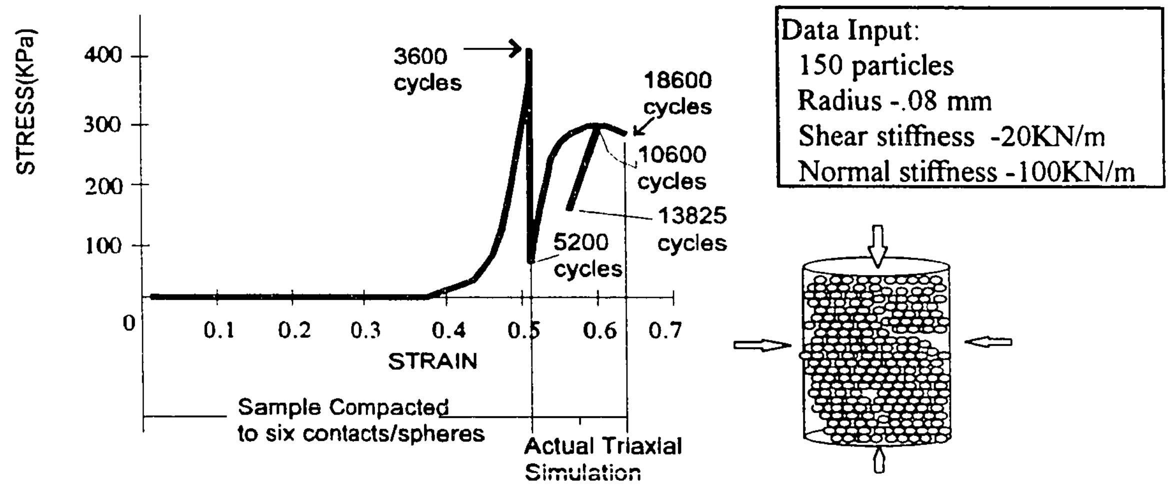


Figure 3.6 A comparison between TRUBAL and TPM of a simulated triaxial test

sample was strained in a vertical direction and the corresponding vertical stresses were recorded. The two curves proved to be identical as expected, showing that the altered data structure has no effect on the results. TPM performed this simulation two times faster with a 32 node parallel data structure, than it did with Trubal's serial algorithm on a single processor of the CM-5's RISC processor.

The performance evaluation of TPM was based on the speedup,

$$Sp(N) = T^*(N)/Tp(N) \qquad (3.4)$$

where $T^*(N)$ is the CPU time for the best serial version of the algorithm running on a single processor, Tp(N) is the CPU time taken by the parallel algorithm with p processors, and N is the problem size or number of elements.

3.5 Conclusion

The data structure that was used in this current version has a disadvantage of not always matching the problem size with the architecture memory configuration required for peak performance. If the size of an array does not match the machine configuration, the next largest size that matches the machine is created. When this happens on SIMD platforms, extra processing power is used that is not needed. TPM's overall communication was retarded between processors due to this type of problem. As a result, a lower efficiency was obtained which means that without further modification, TPM is not scalable to a larger problem. As shown in equation 3.5, the efficiency formula

$$Ep=Sp(N)/p \qquad (3.5)$$

is directly dependent upon the size of the problem (N) and inversely dependent upon the number of processors (p).

CHAPTER 4

SIMULATIONS OF ROWE'S EXPERIMENT

In this chapter, the results from TPM version 2.0 simulations of Rowe's steel sphere experiment are described. The first simulation was to 403 spheres which was identical to that used by Cundall (1979a). The second was a complete simulation with spheres as reported in Rowe (1962). Several indications validating the TPM algorithm are discussed.

4.1 Introduction

As stated before, the main thrust of this research is to develop an algorithm which can simulate three-dimensional geotechnical problems significantly faster than the currently available speed of computation. However, from the previous chapters, it is clear that a dynamic memory model does not perform efficiently on a SIMD architecture. The overhead in communication due to global reductions, broadcast and random sorting can hinder the overall performance of each time-step, which is further retarded by the number of time-steps required to complete the simulation. In this chapter, three advantages of TPM converting a dynamic memory model to a static memory arrangement are discussed using a simulated triaxial experiment. First, it will be shown that even when the size of the assembly is increased by a factor of four, the TPM algorithm can maintain its improved performance. Second, a speedup of nine fold was obtained when the entire CM-5 architecture is exploited. Third, TPM's sample size limitations can be shown to accomodate well over a thousand particles.

Rowe's (1962) triaxial test known as the "uniform spheres in face-centered cubic packing" experiment was simulated with Trubal and TPM to validate the performance and accuracy of the TPM algorithm. In doing so, the global behavior of a granular type sample under triaxial loading was clearly effected by the particle arrangement in the assembly. This was demonstrated simulating a triaxial test using an assembly of 1/4 in. diameter steel balls. Since the discrete element method is based on modeling the behavior of discrete element bodies within an assembly, Rowe's test was an ideal experiment for Cundall's program Trubal (Cundall et. al.,1979). In order to clearly validate the usefulness of TPM version 2.0, a simulation of Rowe's model was performed using the TPM algorithm and the same assembly size and parameters as Cundall. Since Cundall's assembly was smaller than Rowe's, the full size assembly was also simulated by TPM in order to test the accuracy of the results and the overall performance.

4.2 Rowe's Physical Laboratory Test

The laboratory model used by Rowe consisted of an octagonal shaped packing of "large" and "small" layers (see fig 4.1). The 1,672 sphere sample consisted of 13 large layers with 76 spheres in each row and 12 small layers with 57 spheres of each row placed alternately on top of one another. The large layers at the top and bottom of the sample were in contact with the axial loading that was applied to the sample. The rubber membrane which encompassed the sample created the confining pressure.

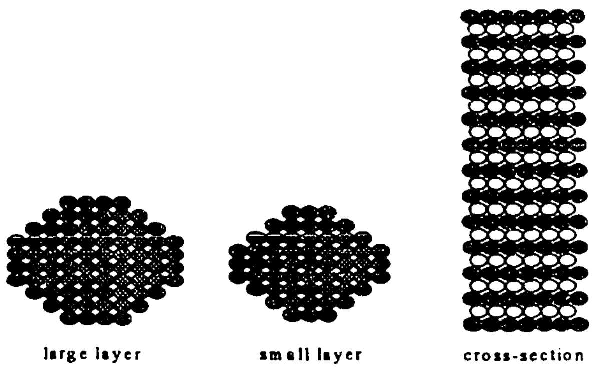


Figure 4.1 Layers of spheres used in physical test by Rowe (1962)

4.3 Cundall's Numerical Simulation

Cundall's numerical sample of this model was slightly smaller in size, because of the limitation of the computer's memory capacity at that time. Hence, the model's large layer was reduced from 76 spheres to 37 spheres and the small layer was reduced from 57 spheres to 24 spheres as illustrated in figure 4.2. In this case, seven large layers and six small layers placed alternately on top of one another with the cross-section as

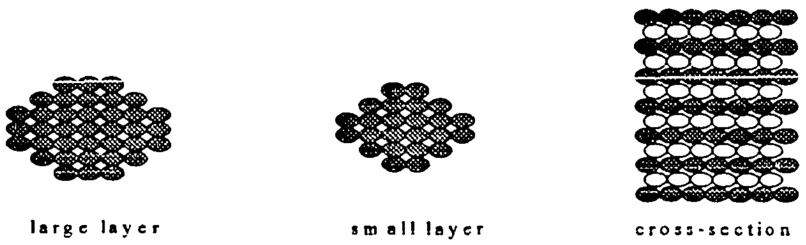


Figure 4.2 Layers of spheres in the numerical test.

shown. Table 4.1 shows the parameters used in the Trubal simulation where the contact stiffnesses were chosen so that the elastic deformations would be small compared to the distortions arising from the slip between particles. The end platens used in the Rowe model to apply the load were simulated, by fixing the velocity of the top and bottom boundary particles in the z-direction. During the compaction phase, the particle velocity is set to zero so that the assembly can achieve a state of equilibrium. The rubber membrane is approximated to form of an *ideal* membrane (shown as dotted lines in figure 4.3) and is assumed to stretch between the particles. Since the membrane is only in contact with

Table 4.1 Parameters of material properties used in Cundall's numerical simulation

Density of each sphere:	2000
shear contact stiffness:	1.5×10^9
normal contact stiffness:	1.5×10^9
friction angle	: 7 degrees
cohesion	: 0
radius of each sphere	: 20
confining pressure($_2$)	:5×10^4

the boundary particles, the confining pressure is only applied individually to the particles on the outer perimeter of the larger layers. These forces are computed using the exposed lengths Lx, Ly and Lz, defined in figure 4.3 and can be written as follows:

$$Fx=Lx*Lz*\sigma_2 \quad (4.1)$$

$$Fy=Ly*Lz*\sigma_2.. \quad (4.2)$$

where σ_2 is defined as the confining pressure. None of the forces in the z- direction were considered for the boundary particles in the larger layer. It was noted that the fixed forces on the boundary were only valid for small strains and displacements, since the actual forces will vary as the geometry changes. In order to make a comparison to

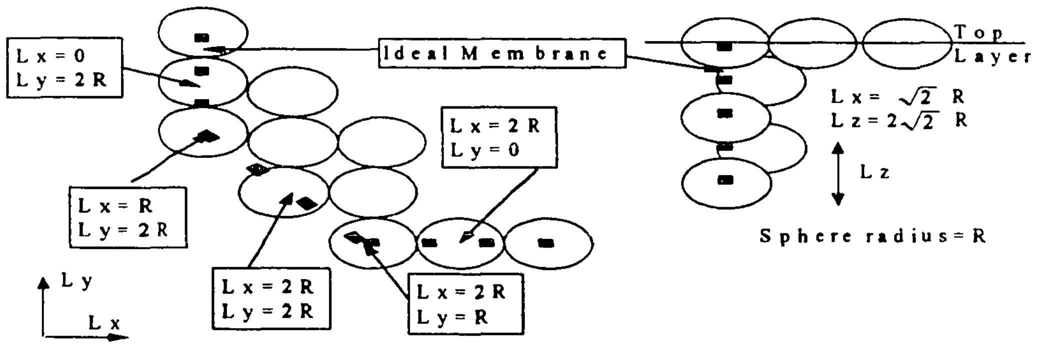


Figure 4.3 Exposed lengths for plane through large layer in x-y and x-z direction

Rowe's results, the graph of the axial strain, e_1, vs. the stress ratio, R, used in Rowe's paper was studied. The stress ratio is defined as follows:

$$R = \frac{\sigma_1'}{\sigma_2'} = \frac{F_1}{A * \sigma_2} \tag{4.3}$$

where A is the area of the octagonal shape formed by the assembly, F_1 is the measured platen force, and σ_2 is the confining pressure on the boundary particles ($\sigma_2 = 5*10^4$). The axial strain, e_1, is defined as follows:

$$e_1 = \frac{2 * \delta}{\Delta h} \tag{4.4}$$

where δ is the measured displacement at either boundary layer, and Δh is the distance between the centers of the top and bottom layers. The factor of 2 appears due to the movement of the top and bottom layers. Figure 4.4 shows (Cundall et. al. 1979) Rowes experimental test and Cundall's numerical simulation. Rowe's test was re-plotted in this

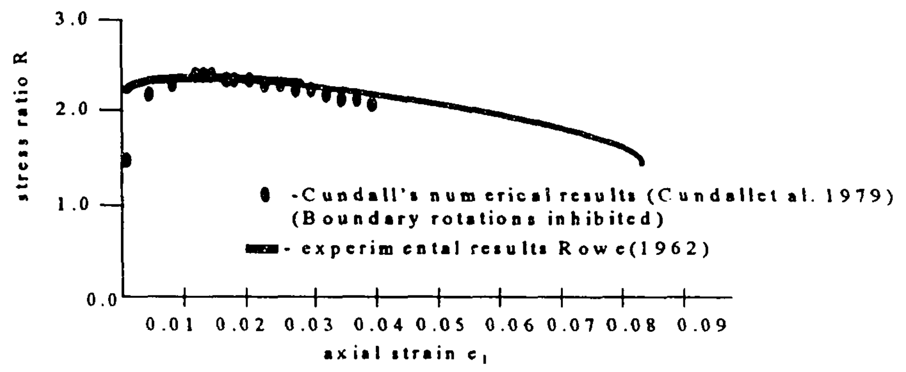


Figure 4.4 Comparison between Rowe's results and Cundall's numerical results

figure on an extended horizontal axis for clarity. Because the rubber membrane produced a confining pressure, Cundall's numerical results were improved when the rotations of particles in contact with the membrane were fixed. Therefore, all of the simulations reported in this chapter restrict rotation when particles contact the membrane.

4.4 TPM's Simulation of the Rowe's Model

A resimulation of Cundall's 403 sphere model was performed on the control processor of the CM-5 and then TPM's version 2.0 was tested for the same simulation on 32, 64, 128, 256, and 512 nodes respectively. Also, a number of full scale model simulations of 1672 spheres were tested on a different number of nodes and the results of this simulation were compared with the results of the 403 sphere simulations. The input parameters for TPM's algorithm were the same parameters used by Cundall. However, when Cundall's

Rayleigh damping values of λ_{min}=0.05 and f_{min}=0.5 for the fraction of critical damping and the modal frequency respectively, were applied to TPM version 2.0 the simulation

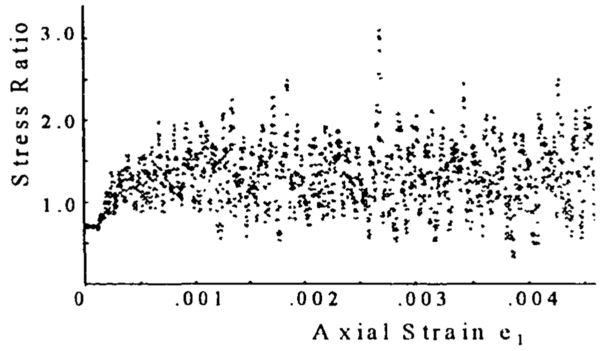


Figure 4.5 Instability in TPM version 2.0 simulation due to improper damping

became unstable as shown in figure 4.5. It was found by increasing the damping values to one, for both λ_{min} and f_{min}, stable results were achieved as shown in figure 4.6. In Figure 4.6, results from the TPM algorithm, for both the 403 and 1672 particle simulation, were superimposed onto the results of Figure 4.4, to show the agreement. Both Cundall

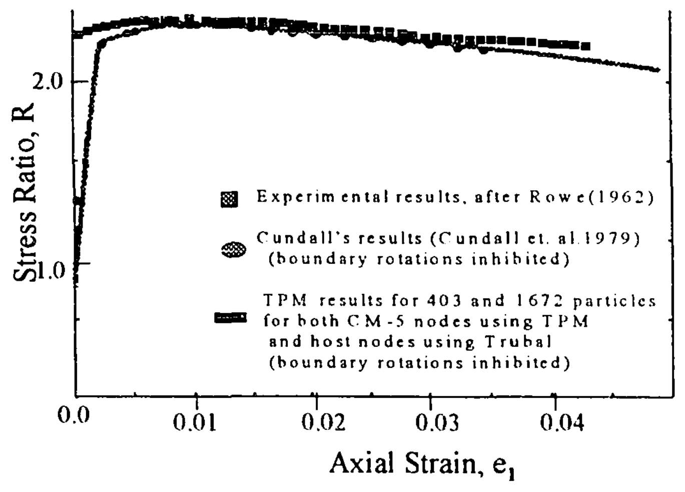


Figure 4.6 Results presented by Rowe, Cundall, and TPM version 2.0

TPM results begin at a lower R value, because that is the equilibrium of the assembly before it is strained for the triaxial test.

4.5 Discussion

The speedup of the simulation was calculated and plotted for the 403 and 1672 sphere simulation in order to analyze the performance of TPM. The 403 sphere simulation time was recorded from the control processor, 32, 64, 128, 256, and 512 nodes. After the speedups and efficiencies were calculated based on equation 3.1 and 3.2 respectively, the speedups were plotted. Then the cpu times for the 1672 sphere simulation were recorded, but due to the memory consumption, only the control processor, 256 nodes, and the 512 nodes could perform this simulation. Table 4.2 shows the times from each simulation along with their corresponding speedups and efficiencies.

Table 4.2 Speedup chart for the 403 and 1672 sphere simulation

	control processor	32 nodes	64 nodes	128 nodes	256 nodes	512 nodes
403 spheres						
speedup	1	1.6	2.8	4.4	6.5	7.9
efficiency	100%	5%	4.4%	3.4%	2.5%	1.5%
cpu time	2.3 hrs	35 min	20min	12 min	8 min	7 min
1672 spheres						
speedup	1	-----	-----	------	5	8.7
efficiency	100%	-----	-----	------	2.0%	1.7%
cpu time	5.4 hrs	-----	-----	------	67.1 min	37.1 min

Figure 4.7 shows the curves of actual speedup of the Rowe simulation along with the ideal speedup. The ideal speedup is based on the premise that the code is completely

parallel and there are no overheads in communication processes. Since processors are doing the same work, a speedup becomes a multiple of the number of processors used. In the case of TPM version 2.0, global communication was a major bottleneck within the algorithm producing a lower efficiency as the number of processors increased. Figure 4.7 shows the speedup curve for the 403 spheres simulation as noted in Table 4.2.

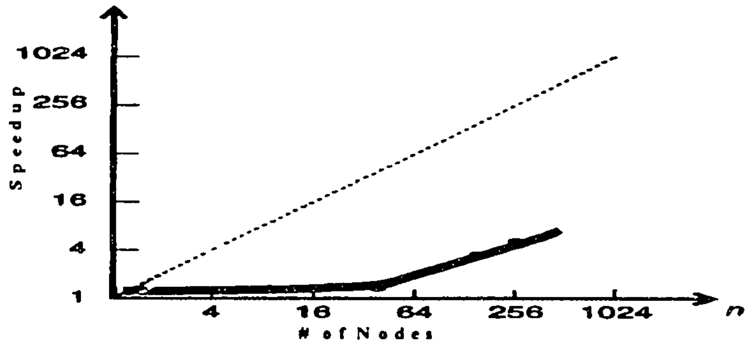


Figure 4.7 Speedup plot for TPM algorithm simulating the 403 sphere Rowe model

4.6 Conclusion

In conclusion, three basic advantages were demonstrated in the results of this chapter. First, TPM can exploit a SIMD machine architecture with its static memory arrangement and obtain a speedup of up to nine times for a 1672 particle simulation. With Rowe's model, the damping constants and equivalent confining pressures had to be increased and reduced respectively, in order to achieve the correct results. Secondly, a drastic increase in the problem size had a negligible effect on the speedup, instead of decreasing the speedup as expected. In the case presented, the problem size was increased by a factor of four, however Table 4.2 shows small differences in the speedup between the two problem sizes. Lastly, the size of assembly can exceed over a thousand particles, even though TPM's memory requirement restricts very large problem sizes. This restriction was overcome during the 1672 particle simulation by repeatedly using arrays that were the square of the problem size (i.e. as in Table 3.2) for different parameters.

CHAPTER 5

FUTURE RESEARCH AND CONCLUSION

This chapter discusses the future of parallel computing and its application to geotechnical engineering problems. As computer technology develops, parallel computing is beginning to play a role of increasing significance. This implies that the inherent parallelism within the discrete element model should become of increasing importance to the geotechnical engineering profession. This chapter first summarizes the findings of earlier chapters and then later describes the implication of this work with regard to future heterogeneous platforms.

5.1 Conclusion

This thesis first algorithm (TPM version 1.0) exploited the architecture of the CM-2 as a parallel supercomputing application to geotechnical engineering problems. In this version, Trubal version 1.51 was modified by eliminating the "link and list " memory management, replacing it with a parallel data structure that treated each particle separately within each processor. At first, this version did not achieve a speedup due to problems with global sorting and broadcasting of data. TPM version 2.0 on the other hand, achieved a significant speedup by creating a static memory data structure throughout the algorithm.

The faster TPM version 2.0 assigns each processor a multiple number of contacts with each sphere paired with every other sphere in the assembly and each pair kept within the same processor. As a result of this static memory arrangement, the assembly size that was needed was the square of the simulated model assembly. The extra memory that was required then placed limitations on the size of the problem that could be simulated.

Three basic advantages were obtained from the results presented in this thesis. First, TPM can exploit a SIMD machine architecture with its static memory arrangement and obtain a speedup of up to nine times for a 1672 particle simulation. Secondly, a drastic increase in the problem size does not decrease the overall speedup as expected. In the case presented, when the problem size was increased by a factor of four, only a small difference in the speedup was noted between the two problem sizes. Lastly, the size of assembly can exceed a thousand particles, even though TPM's memory requirement restricts very large problem sizes from being simulated. This restriction was overcome during the 1672 particle simulation by repeatedly using arrays that were the square of the problem size for different parameters. As a result, memory consumption is reduced and a simulation using over a thousand spheres can be handled effectively with this technique.

Excessive global communications found in the global operations appeared to be the inherent weakness of TPM version 2.0, however it is able to handle rapid data parallel contact detection which is found to be most time intensive in discrete models. Therefore, the contact detection portion of the algorithm is favorable for a proposed heterogeneous platform which is currently being tested.

5.2 Future Research

As previously mentioned, the three basic components that were considered for parallel systems were the *multiple processing capabilities*, the *interconnection network*, and the *control systems*. Up to this point, only the simplest type of control system (SIMD) was considered, with a homogeneous system of processors. SIMD inherent ability to handle data parallel operations for discrete models has been demonstrated, however its inherent weakness was found in global operations.

The other control system is the MIMD architecture which is slightly more complicated but has some distinctive advantages for a discrete model. Its inherent strength is its handling of dynamic communication between processors. However, scaling up this dynamic communication for larger problems can increase the cost of the communication overhead. Of course, SIMD does not have this type of problem. Separating these two control systems within an architecture does not necessarily lead to the best performance in larger discrete models. For this reason, heterogeneous platforms should be noted as a possible configuration for handling a dynamic data structure that can cause constraints within SIMD and MIMD topologies. In this case, a heterogeneous platform is suggested only for large systems where communication between various platforms is negligible. Since SIMD has already been reviewed in detail, some aspects of MIMD should be highlighted to motivate the discussion of heterogenous platforms.

5.2.1 The MIMD Approach

A MIMD architecture implies that processors are able to handle computations independently of each other. This is not the case with SIMD architectures or conventional serial machines. For this reason, programing a MIMD machine is slightly more complicated than programming a SIMD machine. If the algorithm is not correctly synchronized by the programmer, processors can hang up waiting for messages that are not there. In addition, the complexity of programming the algorithm increases in proportion to the size of the problem. For this reason, scaling the problem size of an MIMD machine to include more processors is difficult to program. For example, Kurioaka's (1994) and Hustrihild (1995) considered dividing the discrete element model geometrically. The basic idea of this approach is to divide the assembly space into regions. However, in this scheme each processor is assigned the computations within each region and the regions are defined differently (see Figure 5.1). There are two common drawbacks to this MIMD approach.

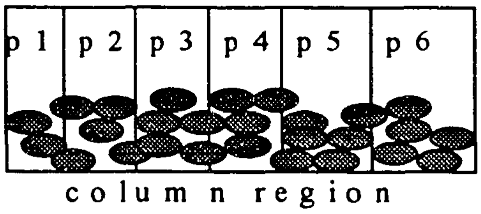


Figure 5.1 Clustering schemes of particles within multiple processors

First is the sporadic migration of particles and the formation of contacts across the regions which increase the overhead communication. Second is the load balancing of processors due to these same migration of particles and the deformation of the assembly space. In order to load balance the processors, each processor must be given the same workload otherwise poor efficiency is created within the algorithm. Both Kurioka and Hustrihild chose column shape regions to minimize these two problems because 1)a one dimensional arrangement of columns simplifies the interprocessor communication between shared boundaries and 2) in some cases the load balancing among processors favors columns due to gravity effects. As a result, the performance of these algorithms is governed by the interprocessor communication and the load balancing. The proposed research takes advantage of this time by combining the platforms of SIMD and MIMD.

5.2.2 Combining Control Systems

One approach that is currently being undertaken can complement the methods that have just been presented. By considering data dependencies at the routine level and even at the variable level, the control systems of various architectures can be optimized. In this approach, the problem is analyzed by grouping particles as in the previous section, as well as studying each routine for its need for shared memory. This approach was tested on the CM-5 which functions as a MIMD and SIMD control system as described in section 3.2. TPM version 2.0 was modified to perform independently on each of the CM-5 nodes and each node represented a column shaped region as described in Figure 5.1. Recall that the SIMD control system of the CM-5 functions by using the control processor as the front end and the nodes as processing elements receiving the same instruction. However, in this application, data parallel operations are able to be performed by using the RISC processor within each node as a front end and the four vector units as processing elements. For this reason, TPM version 2.0 data parallel calculations were applied to each node as a SIMD operation. The MIMD control system was applied to any routine requiring global communication between the column regions or nodes as described in Table 5.1. Note that the contact detection required two separate routines. The majority of the contact detection was handled locally within the nodes or regions, however migrating particles and particles which overlap the regions required global communication to detect neighboring contacts. The complexity of handling the data structure of this combined control system has not yet been resolved. The development of such an algorithm has produced numerous bugs that are difficult to locate for large assemblies.

Table 5.1 Control system optimization at routine level for the CM-5

ROUTINE FUNCTION	CONTROL SYSTEM
MOTIONING PARTICLES	MIMD
CONTACT DETECTION (LOCAL)	SIMD
CONTACT DETECTION (GLOBAL)	MIMD
INTERPARTICLE CALCULATIONS	MIMD
GLOBAL STRESS CALCULATIONS	MIMD

5.2.3 Heterogeneous Platform

The approach described in Section 5.2.2 is not limited to one architecture and is also being pursued on a heterogeneous platform so that each task is optimized on the architecture best suited for it. This approach is designed for discrete models with particles ranging into the thousands for each column region. As mentioned in Section 5.2.1, any MIMD machine can handle a problem within its memory limitations, without the aide of an SIMD control system. The idea proposed here is that while the MIMD control systems are handling the communication between regions, the SIMD control systems can be detecting contacts within the region. It is believed that the speed required for both operations to be performed simultaneously will be comparable.

Because an assembly size is being designed to exceed the memory capacity of a CM-5 node, a parallel virtual machine (PVM) is being tested. This virtual machine consists of the CM-5, the Cray T3D, and a SUN workstation acting as a front end. In this scheme, the CM-5 is performing the SIMD operations of the algorithm using TPM version 2.0, while the Cray T3D performs the MIMD operations of the algorithm and the interconnection network between the two is performed by Oak Ridge National Laboratory's (ORNL) PVM version 3.3. Based on the assumption that MIMD is better suited for global communication and SIMD is better suited for contact detection of bodies/elements, this algorithm is being proposed as shown in Table 5.2. In this case, the contact detection which consumes most of the cpu-time is being optimized by using the CM-5's SIMD control system. The Cray T3D and PVM architectures are described briefly in the following section.

Table 5.2 Control System Optimization at Routine Level of a Heterogeneous Platform

ROUTINE FUNCTION	ARCHITECTURE	PROCESSOR	CONTROL SYSTEM
MOTIONING PARTICLES	CRAY T3D	DEC ALPHA	MIMD
CONTACT DETECTION (LOCAL)	CM-5	RISC	SIMD
CONTACT DETECTION (GLOBAL)	CRAY T3D	DEC ALPHA	MIMD
INTERPARTICLE CALCULATIONS	CRAY T3D	DEC ALPHA	SIMD
GLOBAL STRESS CALCULATIONS	CRAY T3D	DEC ALPHA	MIMD

5.3 Cray T3D and PVM

The Cray T3D being tested has a MIMD control system with 512 DEC alpha nodes multiple processing capability and a 3D Torus for its interconnection network. Each DEC Alpha runs at 150 Mhz and is attached to 64 Mbytes per processor. The 3D Torus, which allows for a communication rate of 300Mbytes per second, has a relatively high bandwidth between processors compared to those machines previously listed (see Fig 5.2).

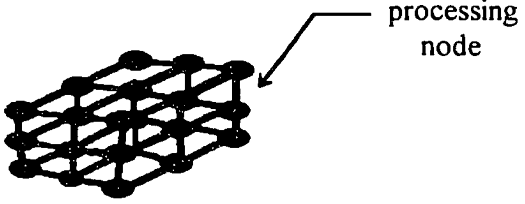


Figure 5.2 3D Torus Communication Network

PVM can scale to a larger problem by networking many machines together. It allows a heterogeneous collection of unix computers to be linked together under a *master* host. PVM, a product of a collaborative venture between DOE and several universities (Geist et al., 1994), can use a regular workstation containing the source file as a *master* host. By installing the basic PVM software on various platforms, the *master* host (SUN workstation) operates all of its *slave* host as a single high-performance parallel machine. At this level, many massively parallel systems (such as the Cray T3D and the CM-5) can be supported on this system. The routines within the host program allow for the initiation and termination of tasks across the network as well as communication and synchronization between tasks. The communication constructs include those for sending and receiving data structures as well as high-level primitives such as broadcast, barrier synchronization, and global sums. The level of this application is feasible, only if the size of the proposed model simulation is very large, because then the communication overhead time will become less significant in the overall performance.

5.4 Implementation of the Proposed Heterogeneous Platform

The Sun Sparc Workstation is the slowest machine so it is designated as the host of the virtual system. It is required to start or *spawn* the Cray T3D and the CM-5 programs, as well as terminate them (see Fig 5.3). The workstation also retrieves results that are necessary for printing, thereby reducing the overhead in the other architectures that would have performed this operation. Since the Cray T3D interconnection network is very fast,

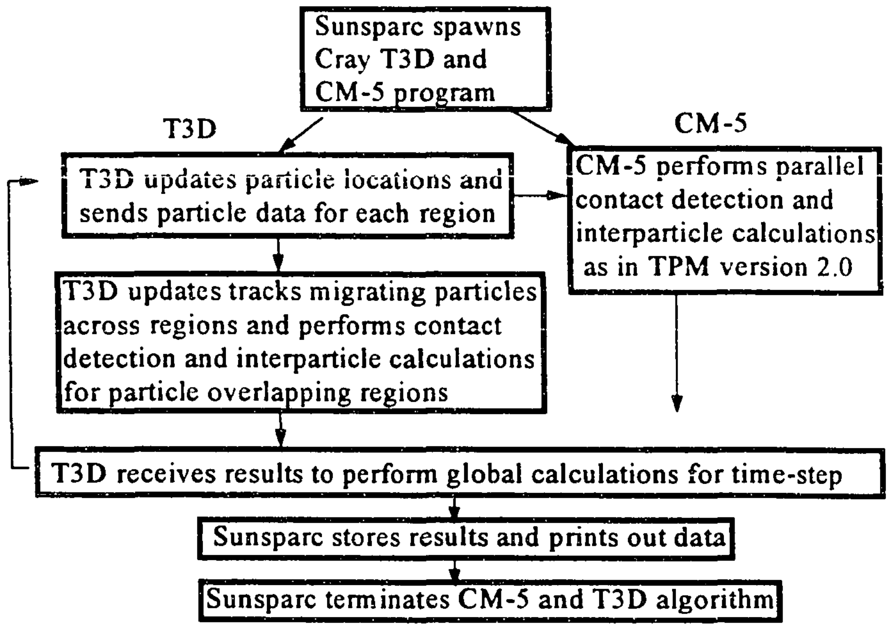

Sunsparc spawns Cray T3D and CM-5 program
T3D
CM-5
T3D updates particle locations and sends particle data for each region
CM-5 performs parallel contact detection and interparticle calculations as in TPM version 2.0
T3D updates tracks migrating particles across regions and performs contact detection and interparticle calculations for particle overlapping regions
T3D receives results to perform global calculations for time-step
Sunsparc stores results and prints out data
Sunsparc terminates CM-5 and T3D algorithm

Figure 5.3 TPM's proposed heterogeneous algorithm using PVM

it is selected for tracking migrating particle as well as calculating particle contacts that overlap their regions. Because TPM version 2.0 handles contact detection rapidly in parallel, it is being adapted to the CM-5 algorithm and also performs the interparticle calculations. All information is returned to the Cray T3D so that global stresses can be computed. It should be noted that the communication speed between the Cray T3D and CM-5 is governed by the internet, which is relatively slower than the interconnection network of these architectures. Therefore, the problem selected should be of proportionate size to balance out the latency required for global and local communication required for each region.

By developing an algorithm of this type, geotechnical problems can be solved for large systems. Also, other platforms can be incorporated to this scheme to meet the computational demands that are required.

APPENDIX

TPM VERSION 2.0 (ROWE'S MODEL)

```
C Include file - Trbcom.inc for TPM version 2.0
*
*  SETUP/NEXT
*
         LOGICAL LOGFLG,ERROR,GMVFLG,hisflg,hertz,twod
     integer, parameter ::cor=403
     integer, parameter ::pos=cor
     double precision, array(3):: xmax
         integer nerr
     integer i,j,k,l
         integer lunr,lunw,MODE,nfob
         integer NVARB,NTYPM,NTYPS,NBOX
         real OVLAP,PI,DEGRAD
         real ALPHA,BETA,fff,fob
         real GAIN,SERVEM
         character*4 hed

*
*   GEN VARIABLES
*
         integer geno,nreq,n
     integer skips
     real eax
     integer m1a, m2a
         integer NBALL,NTOT,NCYC
     logical genflag
*      real, array(3,5)::random

*
*    CON VARIALBLES
*
    integer memory
    real, array(cor,3)::con
    real, array(cor,2)::con1
    real, array(cor)::rrr
    real, array(cor)::shft
    logical, array(pos,pos)::mas,mas1,mas2

*
*    SPHERE VARIABLES
*
          real, array(pos,pos)::s1x
          real, array(pos,pos)::s1y
          real, array(pos,pos)::s1z
          real, array(pos,pos)::s2x
          real, array(pos,pos)::s2y
          real, array(pos,pos)::s2z
          real, array(pos,pos)::v1x
          real, array(pos,pos)::v1y
          real, array(pos,pos)::v1z
          real, array(pos,pos)::v2x
          real, array(pos,pos)::v2y
          real, array(pos,pos)::v2z
          real, array(pos,pos)::tv1x
    real, array(pos,pos)::tv1y
    real, array(pos,pos)::tv1z
    real, array(pos,pos)::tv2x
    real, array(pos,pos)::tv2y
    real, array(pos,pos)::tv2z
    real, array(pos,pos)::rd1
    real, array(pos,pos)::rd2
    real, array(pos,pos)::x1x
    real, array(pos,pos)::x1y
    real, array(pos,pos)::x1z
    real, array(pos,pos)::x2x
    real, array(pos,pos)::x2y
    real, array(pos,pos)::x2z
    real, array(pos,pos)::xd1x
    real, array(pos,pos)::xd1y
    real, array(pos,pos)::xd1z
    real, array(pos,pos)::xd2x
    real, array(pos,pos)::xd2y
    real, array(pos,pos)::xd2z
real, array(pos,pos)::t1x
real, array(pos,pos)::t1y
real, array(pos,pos)::t1z
real, array(pos,pos)::t2x
real, array(pos,pos)::t2y
real, array(pos,pos)::t2z
    real, array(pos,pos)::td1x
    real, array(pos,pos)::td1y
    real, array(pos,pos)::td1z
    real, array(pos,pos)::td2x
    real, array(pos,pos)::td2y
    real, array(pos,pos)::td2z
real, array(pos,pos)::f1x
real, array(pos,pos)::f1y
real, array(pos,pos)::f1z
real, array(pos,pos)::f2x
real, array(pos,pos)::f2y
real, array(pos,pos)::f2z
    real, array(pos,pos)::m1x
    real, array(pos,pos)::m1y
    real, array(pos,pos)::m1z
    real, array(pos,pos)::m2x
    real, array(pos,pos)::m2y
    real, array(pos,pos)::m2z
real, array(pos,pos)::xxx
real, array(pos,pos)::xyy
real, array(pos,pos)::xzz
real, array(pos,pos)::xsx
real, array(pos,pos)::xsy
real, array(pos,pos)::xsz
real, array(pos,pos)::rdf
real, array(pos,pos)::dif
```

```
*  MOTION VARIABLES
*
        real cn1,cn2,gt1,gt2,gt3,tm,tmi
     real kkn

        real, array(pos,pos)::dt1x
        real, array(pos,pos)::dt1y
        real, array(pos,pos)::dt1z
        real, array(pos,pos)::dt2x
        real, array(pos,pos)::dt2y
        real, array(pos,pos)::dt2z
        real, array(pos,pos)::xt1x
        real, array(pos,pos)::xt1y
        real, array(pos,pos)::xt1z
        real, array(pos,pos)::xt2x
        real, array(pos,pos)::xt2y
        real, array(pos,pos)::xt2z
*
*   CONTACT CHECKING VARIABLES
*
      integer rmax
      real xmb

*
*    FORD VARIABLES
*

      real, array(pos,pos):: snx
      real, array(pos,pos):: sny
      real, array(pos,pos):: snz

      real, array(pos,pos):: fn1
         real, array(pos,pos):: udm

         real, array(pos,pos):: fmx
         real, array(pos,pos):: fmy
         real, array(pos,pos):: fmz

         real, array(pos,pos):: rnx
         real, array(pos,pos):: rny
         real, array(pos,pos):: rnz

         real, array(pos,pos):: fxx
         real, array(pos,pos):: rtx

      real, array(pos,pos):: tpx
      real, array(pos,pos):: tpy
      real, array(pos,pos):: tpz

      real, array(pos,pos):: frx
      real, array(pos,pos):: fry
      real, array(pos,pos):: frz

      real, array(pos,pos):: fsx
     real, array(pos,pos):: fsy
     real, array(pos,pos):: fsz

        real, array(pos,pos):: t11
        real, array(pos,pos):: t12
        real, array(pos,pos):: t13
        real, array(pos,pos):: t21
        real, array(pos,pos):: t22
        real, array(pos,pos):: t23
        real, array(pos,pos):: t31
        real, array(pos,pos):: t32
        real, array(pos,pos):: t33

        integer, array(cor)::ityps
        integer, array(3)::NX

     integer port,x,y,o,nxy
     integer mct,ct,np,bn
     integer mp,ctt,tcm,vc
*      logical, array(maxpos)::fixed

        real, array(3)::WINDL,WINDU
*
*    BBTEST ARRAYS
*
        double precision xshear

*
*    INITP ARRAYS
*
     real TDEL,FRAC,AKSS,AKNN,AMUU,COHH,AKS1
        real, array(5)::AMASS,AMOI,DENS
        real, array(5)::shear,poiss,r
        real, array(5,5)::AKN,AKS,ccn,ccs,AMU,COH
        real, array(3,3)::EDGRID,EDUSER,SSAMPL,EDSERV

*
*    CYCLE VARIABLES
*
      real AT2
      integer ncont,nmm,crf,num1
*       real, array(maxpos)::iac
*          logical, array(6,maxpos)::SKIP
*       logical, array(6)::skip
          real, array(3)::DEL,GRAV

*
*    PRINT VARIABLES
*
      real volb,volg,fnav
      real ovlrat,coord,temp1
      real, array(cor):: svolb
          real, array(pos,pos):: sdd1,sdd2
          real, array(pos,pos):: dd1,dd2
```

```
*
*   RUNSERV
*
    integer nserv
    real, array(10)::SRVVAL
    integer, array(10)::ICDSRV
*
*   ROWE VARIABLES
*
    logical, array(pos,pos)::walp,layp,twalp,tlayp
    real, array(pos,pos)::f11x
    real, array(pos,pos)::f11y
    real, array(pos,pos)::f22x
    real, array(pos,pos)::f22y
**************************************************
*
*   COMMON SECTION
*
**************************************************
*****  SETUP COMMON

    COMMON /WIDTH/XMAX

*****  GEN COMMON

*     common /gencom/ gapmin,genflag,rbtest
    common /mot/ cn1,cn2,gt1,gt2,gt3,tm,tmi,
   .         kkn
    common/trccc/trc,trm,ier,trc1,trm1,eax
    common/numm/nmm
    common/jjf/jj
         COMMON /TRBCOM/
    ^    hertz,NERR,ERROR,lunr,lunw,lunh,
    ^    NVARB,NTYPM,NTYPS,num1,
    ^    OVLAP,PI,TOL,RMAX,
    ^    TDEL,FRAC,ALPHA,BETA,degrad,
    ^    NBALL,NTOT,NCYC,CRF,
    ^    NCONT,SLIDE,LOGFLG,hisflg,twod,
    ^    MODE,GMVFLG,xshear,GAIN,SERVEM,NSERV

         COMMON /PCOM1/
    ^    akss,aknn,amuu,cohh,aks1,radi,gt

         COMMON /PCOM/
    ^    AMASS,AMOI,DENS,AKN,AKS,
    ^    shear,poiss,ccn,ccs,
    ^    AMU,COH,NX,WINDL,WINDU,
    ^    EDGRID,eduser,SSAMPL,edserv,del


      COMMON /SRV/ ICDSRV,SRVVAL
      common /htzfrd/ ccnh,ccsh
         COMMON/need/m1a,nreq,nbox
         COMMON/generate/nreqd,nitypm,nitypm
```

```
        COMMON/parallel/R,con,
   ^    GRAV

     common /stasav/ xcen,rot,xmsav
        common /trb1/ hed(20)
*
*   SPHERE VARIABLES
*
     common/sphh/ s1x,s1y,s1z,s2x,s2y,s2z,
    . v1x,v1y,v1z,v2x,v2y,v2z,tv1x,tv1y,tv1z,
    . tv2x,tv2y,tv2z,rd1,rd2,x1x,x1y,x1z,
    . x2x,x2y,x2z,xd1x,xd1y,xd1z,xd2x,xd2y,
    . xd2z,t1x,t1y,t1z,t2x,t2y,t2z,f1x,f1y,
    . f1z,f2x,f2y,f2z,m1x,m1y,m1z,m2x,m2y,m2z,
    . xxx,xyy,xzz,xsx,rdf,dif,rrr,f11x,f11y,
    . f22x,f22y
     common/mott/dt1x,dt1y,dt1z,dt2x,dt2y,dt2z,
    . xt1x,xt1y,xt1z,xt2x,xt2y,xt2z
     common/fdd/ snx,sny,snz,fn1,fxx,fmx,t11,
    . t12,t13,t21,t22,t23,t31,t32,t33,frx,fry,frz,
    . tpx,tpy,tpz
     common/conn/mas,mas1,walp,layp,twalp,tlayp
************************************************
*
*  CMLAYOUT
*
************************************************
*
*    Setup/Next arrays
*
cmf$layout DEL(:news),WINDL(:news)
cmf$layout WINDU(:news),NX(:news)
cmf$layout xmax(:news)

*
*   GEN VARIABLES
*
*cmf$layout random(:news,:news)
*cmf$layout nreqd(:news),nityp(:news),nitypm(:news)

*
*    rebox arrays
*
CMF$ LAYOUT con(:news,:news)
cmf$ layout mas(:news,:news)
cmf$ layout mas1(:news,:news)
cmf$ layout mas2(:news,:news)
*cmf$ layout SPH(:serial,:block=4:pdesc=112,:block=1:pdesc=12, &
*cmf$ :block=1:pdesc=3)
*cmf$ layout fixed(:news)
cmf$ layout ITYPS(:news)

cmf$layout AMOI(:news),DENS(:news)
cmf$layout R(:news),AMASS(:news)
```

```
cmf$layout shear(:news),poiss(:news)
cmf$layout ccn(:serial,:news),ccs(:serial,:news)
cmf$layout EDGRID(:serial,:news),EDUSER(:serial,:news)
cmf$layout AKN(:serial,:news),AKS(:serial,:news)
cmf$layout AMU(:serial,:news),COH(:serial,:news)


*     cycle arrays

*cmf$layout iac(:news)

* cmf$layout SKIP(:serial,:news)
* cmf$layout skip(:news)
*
*      motion arrays
*
cmf$layout GRAV(:news)
*
*     FORD VARIABLES
*
cmf$layout SSAMPL(:serial,:news)

*
*     Periodic Boundary
*
cmf$layout svolb(:news)

*     for hertz
*     RUNSERV  VARIABLES
*
cmf$layout EDSERV(:serial,:news)
cmf$layout ICDSRV(:news),SRVVAL(:news)
*
*     SPHERE VARIABLES
*
cmf$ layout s1x(:news,:news)
cmf$ layout s1y(:news,:news)
cmf$ layout s1z(:news,:news)
cmf$ layout s2x(:news,:news)
cmf$ layout s2y(:news,:news)
cmf$ layout s2z(:news,:news)
cmf$ layout v1x(:news,:news)
cmf$ layout v1y(:news,:news)
cmf$ layout v1z(:news,:news)
cmf$ layout v2x(:news,:news)
cmf$ layout v2y(:news,:news)
cmf$ layout v2z(:news,:news)
cmf$ layout tv1x(:news,:news)
cmf$ layout tv1y(:news,:news)
cmf$ layout tv1z(:news,:news)
cmf$ layout tv2x(:news,:news)
cmf$ layout tv2y(:news,:news)
cmf$ layout tv2z(:news,:news)
cmf$ layout rd1(:news,:news)
```

```
cmf$ layout rd2(:news,:news)
cmf$ layout x1x(:news,:news)
cmf$ layout x1y(:news,:news)
cmf$ layout x1z(:news,:news)
cmf$ layout x2x(:news,:news)
cmf$ layout x2y(:news,:news)
cmf$ layout x2z(:news,:news)
cmf$ layout xd1x(:news,:news)
cmf$ layout xd1y(:news,:news)
cmf$ layout xd1z(:news,:news)
cmf$ layout xd2x(:news,:news)
cmf$ layout xd2y(:news,:news)
cmf$ layout xd2z(:news,:news)
cmf$ layout t1x(:news,:news)
cmf$ layout t1y(:news,:news)
cmf$ layout t1z(:news,:news)
cmf$ layout t2x(:news,:news)
cmf$ layout t2y(:news,:news)
cmf$ layout t2z(:news,:news)
cmf$ layout td1x(:news,:news)
cmf$ layout td1y(:news,:news)
cmf$ layout td1z(:news,:news)
cmf$ layout td2x(:news,:news)
cmf$ layout td2y(:news,:news)
cmf$ layout td2z(:news,:news)
cmf$ layout f1x(:news,:news)
cmf$ layout f1y(:news,:news)
cmf$ layout f1z(:news,:news)
cmf$ layout f2x(:news,:news)
cmf$ layout f2y(:news,:news)
cmf$ layout f2z(:news,:news)
cmf$ layout m1x(:news,:news)
cmf$ layout m1y(:news,:news)
cmf$ layout m1z(:news,:news)
cmf$ layout m2x(:news,:news)
cmf$ layout m2y(:news,:news)
cmf$ layout m2z(:news,:news)
cmf$ layout xxx(:news,:news)
cmf$ layout xyy(:news,:news)
cmf$ layout xzz(:news,:news)
cmf$ layout xsx(:news,:news)
cmf$ layout xsy(:news,:news)
cmf$ layout xsz(:news,:news)
cmf$ layout rdf(:news,:news)
cmf$ layout dif(:news,:news)

*
* MOTION PARTITION
*
cmf$ layout dt1x(:news,:news)
cmf$ layout dt1y(:news,:news)
cmf$ layout dt1z(:news,:news)
cmf$ layout dt2x(:news,:news)
cmf$ layout dt2y(:news,:news)
```

```
cmf$ layout dt2z(:news,:news)
cmf$ layout xt1x(:news,:news)
cmf$ layout xt1y(:news,:news)
cmf$ layout xt1z(:news,:news)
cmf$ layout xt2x(:news,:news)
cmf$ layout xt2y(:news,:news)
cmf$ layout xt2z(:news,:news)
*
* FD MEMORY PARTITION
*
cmf$ layout snx(:news,:news)
cmf$ layout sny(:news,:news)
cmf$ layout snz(:news,:news)

cmf$ layout fn1(:news,:news)
cmf$ layout udm(:news,:news)

cmf$ layout fmx(:news,:news)
cmf$ layout fmy(:news,:news)
cmf$ layout fmz(:news,:news)

cmf$ layout mx(:news,:news)
cmf$ layout my(:news,:news)
cmf$ layout mz(:news,:news)

cmf$ layout fxx(:news,:news)
cmf$ layout rtx(:news,:news)

cmf$ layout tpx(:news,:news)
cmf$ layout tpy(:news,:news)
cmf$ layout tpz(:news,:news)

cmf$ layout frx(:news,:news)
cmf$ layout fry(:news,:news)
cmf$ layout frz(:news,:news)

cmf$ layout fsx(:news,:news)
cmf$ layout fsy(:news,:news)
cmf$ layout fsz(:news,:news)

cmf$ layout t11(:news,:news)
cmf$ layout t12(:news,:news)
cmf$ layout t13(:news,:news)
cmf$ layout t21(:news,:news)
cmf$ layout t22(:news,:news)
cmf$ layout t23(:news,:news)
cmf$ layout t31(:news,:news)
cmf$ layout t32(:news,:news)
cmf$ layout t33(:news,:news)
```

```
      Program TPM version 2.0
c
c  Started by Peter Cundall in 1979(trubal), and modified by
c  David W. Washington in 1993(Trubal for Parallel Machines)
c  Model of spheres in periodic space for 64-bit processor.
c
      logical supout
      character*4 word
      include 'tpm.inc'
      include 'matcom.inc'
      COMMON /PROCOM/SUPOUT
      LUNR=2
      LUNW=3
      LOGFLG = .TRUE.
      open (lunr,file='tpm.dat',status='old')
      open (4,file='thg.dat',status='old')
      open (8,file='thg1.dat',status='old')
      open (9,file='thg2.dat',status='old')
      open (lunw,file='tpm.out',status='old')
      open (6,file='tpm.tes',status='old')
    5 CALL SETUP
      IF (.NOT. ERROR) GOTO 10
             GOTO 5
   10 CALL NEXT
      if(error)print*,'we got problems'
      END

      SUBROUTINE SETUP
C
C  TO START NEW PROBLEM, OR DO A RESTART
C
      save
      INCLUDE 'tpm.inc'
      include '/usr/include/cm/CMF_defs.h'
      LOGICAL REST,supout
      character*1 icom,icom1
      character*4 prompt
      INCLUDE 'matcom.inc'
      common /procom/ supout
      DIMENSION W(3),ICOM(13),ICOM1(5)
      DATA ICOM /'S','T','A',' ','R','E','S',' ','S','T','O',' ',iterm/
      DATA ICOM1 /'L','O','G',' ',iterm/
      DATA PROMPT /'trb>'/
      REST=.FALSE.
      call cmf_describe_array(x1x)
C-------------- CONSTANTS ----------------------------
      m1a = 0
      eax = 0
      nmm=0
      geno = 1
      nityp = 0
      nitypm = 0
      nreqd = 0
      nreq = 0
      TOL   = 3.5
      OVLAP = -10.0
      NVARB = 40
      NTYPM = 5
      NTYPS = 5
      PI    = 4.0*ATAN(1.0)
      DEGRAD= PI/180.0
C--------------- INITIALISE ARRAYS AND VARIABLES ---------
*
*     SPHERE VARIABLES
*
      s1x=0.0
      s1y=0.0
      s1z=0.0
      s2x=0.0
      s2y=0.0
      s2z=0.0
      v1x=0.0
      v1y=0.0
      v1z=0.0
      v2x=0.0
      v2y=0.0
      v2z=0.0
      tv1x=0.0
      tv1y=0.0
      tv1z=0.0
      tv2x=0.0
      tv2y=0.0
      tv2z=0.0
      rd1=0.0
      rd2=0.0
      x1x=0.0
      x1y=0.0
      x1z=0.0
      x2x=0.0
      x2y=0.0
      x2z=0.0
      xd1x=0.0
      xd1y=0.0
      xd1z=0.0
      xd2x=0.0
      xd2y=0.0
      xd2z=0.0
      t1x=0.0
      t1y=0.0
      t1z=0.0
      t2x=0.0
      t2y=0.0
      t2z=0.0
      td1x=0.0
      td1y=0.0
      td1z=0.0
      td2x=0.0
      td2y=0.0
```

```
    td2z=0.0
    f1x=0.0
    f1y=0.0
    f1z=0.0
    f2x=0.0
    f2y=0.0
    f2z=0.0
    m1x=0.0
    m1y=0.0
    m1z=0.0
    m2x=0.0
    m2y=0.0
    m2z=0.0
    xxx=0.0
    xyy=0.0
    xzz=0.0
    xsx=0.0
    xsy=0.0
    xsz=0.0
    rdf=50.0
    dif=0.0
*
* MOTION PARTITION
*
    dt1x=0.0
    dt1y=0.0
    dt1z=0.0
    dt2x=0.0
    dt2y=0.0
    dt2z=0.0
    xt1x=0.0
    xt1y=0.0
    xt1z=0.0
    xt2x=0.0
    xt2y=0.0
    xt2z=0.0

*
* FD MEMORY PARTITION
*

    snx=0.0
    sny=0.0
    snz=0.0
    fn1=0.0

    con=0.0

    R     = 0.0
    AMASS = 0.0
    AMOI  = 0.0
    shear = 0.0
    poiss = 0.0
    DENS  = 0.0
```

```
      AKN   = 0.0
      AKS   = 0.0
      AMU   = 0.0
      COH   = 0.0
      ccn   = 0.0
      ccs   = 0.0
      TDEL  = 0.0
      FRAC  = 0.05
      ALPHA = 0.0
      BETA  = 0.0
      GRAV  = 0.0
      NBALL = 0
      NTOT  = 0
      NERR  = 0
      ERROR = .FALSE.
      mas1=.false.
      GENFLAG =.FALSE.
      twod  = .false.
      hertz = .false.
      GMVFLG = .FALSE.
      EDGRID = 0.0
      eduser = 0.0
      SSAMPL = 0.0
      edserv = 0.0
      xshear = 0.0
      GAIN  = 0.0
      SERVS0 = 0.0
      SERVEM = 0.0
      supout = .false.
C--------------- READ PARAMETERS, ETC --------------------
   38 WRITE(lunw,613)
      IF (.NOT. LOGFLG) write(lunw,700) PROMPT
      READ (lunr,500) LINE
      CALL TIDY
      CALL MATCH (ICOM,1,JUMP)
      IF ((.NOT.MISS) .AND. (.NOT.BAD)) GOTO 45
      WRITE(lunw,601)
      GOTO 38
   45 IF (JUMP .EQ. 2) GOTO 200
      IF (JUMP .EQ.3) STOP
      DO 50 I=1,3
      W(I)=RVAR(I+1)
      NPBAD = I
      IF(W(I).LE.0.0) GOTO 1010
   50 CONTINUE
      CALL MATCH (ICOM1,8,JUMP)
      IF (.NOT. BAD) LOGFLG  = .TRUE.
      MISS = .FALSE.
      IF(LOGFLG) WRITE(lunw,600)
      IF(LOGFLG) WRITE(lunw,610)
      NBOX=IVAR(5)
      NBALLM=IVAR(6)
      NWALLM=IVAR(7)
      IF (MISS) GOTO 1010
      NPBAD = 4
      IF(NBOX.LE.0) GOTO 1010
      IF(LOGFLG) WRITE (lunw,603)
      IF(LOGFLG) WRITE (lunw,604) NBOX,(W(I),I=1,3)
      IF(LOGFLG) WRITE (lunw,605) NBALLM,NWALLM
      DDD=(W(1)*W(2)*W(3)/FLOAT(NBOX))**(1.0/3.0)
      DEL(1) = DDD
      DEL(2) = DDD
      DEL(3) = DDD
      NBOX=1
      DO 55 I=1,3
      NX(I)=W(I)/DDD+0.5
      NBOX=NBOX*NX(I)
   55 XMAX(I)=FLOAT(NX(I))*DDD
      IF(LOGFLG) WRITE (lunw,606)
      IF(LOGFLG) WRITE (lunw,604) NBOX,(XMAX(I),I=1,3)
      IF(LOGFLG) WRITE (lunw,607) (NX(I),I=1,3),DDD
      WINDL=0.0
      WINDU=XMAX
      WRITE(lunw,614)
      IF (.NOT. LOGFLG) write(lunw,700) PROMPT
      READ (lunr,502) HED
      IF (LOGFLG) WRITE(lunw,611) HED
C
      GOTO 300
C---------------- RESTART RUN ---------------------
  200 CONTINUE
      IF (ERROR) GOTO 1000
      REST = .TRUE.
  300 CONTINUE
  340 IF(REST) GOTO 400
      sph=0.0
  400 CONTINUE
 1000 CONTINUE
      RETURN
 1010 NERR = 2
      ERROR = .TRUE.
      RETURN
  500 FORMAT(80A1)
  502 FORMAT(20A4)
  503 FORMAT(I3)
  600 FORMAT(30X,'PROGRAM TPM for 3D'
     .     /30X,'---------------------------'/)
  601 FORMAT(30X,'*** FIRST COMMAND MUST BE START OR RESTART ***')
  602 FORMAT(30X,'*** BAD, OR OMITTED PARAMETERS ***')
  603 FORMAT(28X,'REQUESTED PARAMETERS:')
  604 FORMAT(30X,'BOXES          ',I5/
     .        30X,'WIDTH          ',F7.1/
     .        30X,'HEIGHT         ',F7.1/
     .        30X,'THICKNESS      ',F7.1)
  605 FORMAT(30X,'MAX. PARTICLES ',I5/
     .        30X,'MAX. WALLS     ',I5/)
  606 FORMAT(28X,'PARAMETERS USED:')
  607 FORMAT(30X,'NX(1)          ',I5/
     .    30X,'NX(2)          ',I5/
     .    30X,'NX(3)          ',I5/
     .    30X,'BOX DIMENSION   ',F7.3/)
 608 FORMAT(30X,'THIS IS A RESTART RUN')
 610 FORMAT(/30X,'THIS IS A START RUN'/)
 611 FORMAT(/30X,'HEADING: ',20A4)
 613 FORMAT(' START OR RESTART?')
 614 FORMAT(' HEADING?')
 615 FORMAT(' FILENAME?')
 616 FORMAT(' >',5A4)
 700 format (1x,a4)
     END
     SUBROUTINE NEXT
C
C  TO INTERPRET COMMAND LINES (3-D PROGRAM)
C
     save
     INCLUDE 'tpm.inc'
     INCLUDE 'matcom.inc'
     INCLUDE '/usr/include/cm/CMF_defs.h'
     INCLUDE '/usr/include/cm/cmssl-cmf.h'
     character*1 icom,icom1,icom2
     character*4 prompt
     DIMENSION ICOM(165),ICOM1(8),icom2(5),ivp(6)
     DATA ICOM
     . /'L','O','G',' ','C','Y','C',' ','P','L','O',' ',
     .  'C','R','E',' ','N','E','W',' ','I','S','E',' ',
     .  'R','S','E',' ','G','R','I',' ','S','T','O',' ',
     .  'W','I','N',' ','T','Y','P',' ','G','R','A',' ',
     .  'F','I','X',' ','S','H','E',' ','N','O','R',' ',
     .  'D','E','N',' ',
     .  'R','A','D',' ','F','R','I',' ','C','O','H',' ',
     .  'S','E','L',' ','D','A','M',' ','F','R','A',' ',
     .  'R','E','M',' ','G','E','N',' ','P','R','I',' ',
     .  'L','O','C',' ','Z','E','R',' ','R','E','S',' ',
     .  'G','O','B',' ',
     .  'F','M','M',' ','V','M','U',' ','S','A','V',' ',
     .  'G','A','I',' ',
     .  'H','I','S',' ','H','E','R',' ','2','-','D',' ',
     .  'F','R','O',' ','W','A','L',' ',
     .  'M','O','D',' ','B','O','N',' ','P','R','O',' ',
     .  'R','O','W',' ','H','Z','V',' ',iterm/
      DATA ICOM1
     . /'O','N',' ','O','F','F',' ',iterm/
      data icom2 /'A','L','L',' ',iterm/
C
      DATA PROMPT /'trb>'/
      IAL=1
      nrec=0
C---------------- READ NEXT LINE ------------------
    5 IF (ERROR) GOTO 1000
    6 IF (.NOT. LOGFLG) write(lunw,1501) PROMPT
      READ(lunr,1500) LINE
    7 IF(LOGFLG) WRITE(lunw,1601) LINE
```

```
   GOTO 20
 15 WRITE(lunw,1603)
   GOTO 5
C
 20 CALL TIDY
   CALL MATCH (ICOM,1,JUMP)
   IF (MISS) GOTO 5
   IF (.NOT. BAD) GOTO 25
   NERR = 1
   ERROR = .TRUE.
   GOTO 1000
C
 25 GOTO ( 40, 60, 80,100,
  .    120,140,160,180,
  .    200,220,240,260,
  .    280,300,320,340,
  .    360,380,400,420,
  .    440,460,480,500,
  .    520,540,560,580,
  .    600,620,640,660,
  .    680,700,720,740,
  .    630,750,760,770,
  .    780,800,820), JUMP
C--------------- LOG FLAG ---------------------
 40 CALL MATCH (ICOM1,2,JUMP)
   IF (MISS) GOTO 1010
   IF (BAD ) GOTO 1020
   GOTO (42,44), JUMP
 42 LOGFLG = .TRUE.
   GOTO 5
 44 LOGFLG = .FALSE.
   GOTO 5
C--------------- CYCLE THROUGH MOTION & FORD ---------------
 60 NCYC=IVAR(2)
   CALL INITP
   IF (ERROR) GOTO 1000
   CALL CYCLE
   WRITE (lunw,1612) NTOT
*    IF(NTOT.EQ.600)SEC=.TRUE.
*    IF(NTOT.EQ.800)SEC=.FALSE.
   GOTO 5
C--------------- DO A PLOT ------------------------------

 80  CALL APLOT
    GOTO 5
C--------------- CREATE A NEW BALL ---------------------
 100  GOTO 5
C--------------- NEW PROBLEM ------------------
 120 CALL SETUP
   GOTO 5
C--------------- SET MEMORY DIRECTLY ----------
 140 GOTO 5
 160 GOTO 5
C--------------- Grid strain-rates ---------------------
```

```
 180 RV1 = RVAR(2)
   RV2 = RVAR(3)
   RV3 = RVAR(4)
   RV4 = RVAR(5)
   RV5 = RVAR(6)
   RV6 = RVAR(7)
   IF (MISS) GOTO 1010
   eduser(1,1) = RV1
   eduser(2,2) = RV2
   eduser(3,3) = RV3
   eduser(1,2) = RV4
c**** see below *****
ccc     eduser(2,3) = RV5
ccc     eduser(3,1) = RV6
   GMVFLG = .FALSE.
   IF (RV1 .NE. 0.0 .OR. RV2 .NE. 0.0) GMVFLG = .TRUE.
   IF (RV3 .NE. 0.0 .OR. RV4 .NE. 0.0) GMVFLG = .TRUE.
   IF (RV5 .NE. 0.0 .OR. RV6 .NE. 0.0) GMVFLG = .TRUE.
c***** note: only E12 allowed at present ******
ccc     eduser(2,1) = eduser(1,2)
ccc     eduser(3,2) = eduser(2,3)
ccc     eduser(1,3) = eduser(3,1)
   GOTO 5
C--------------- STOP ----------------------------------
 200 CONTINUE

   NREC=0
   GOTO 1000
C--------------- CHANGE WINDOW ----------------------
 220 WINDL(1)=RVAR(2)
   WINDU(1)=RVAR(3)
   WINDL(2)=RVAR(4)
   WINDU(2)=RVAR(5)
   WINDL(3)=RVAR(6)
   WINDU(3)=RVAR(7)
   GOTO 5
C-------------------------------------------------------
 240 GOTO 15
C--------------- GRAVITY --------------------------------
 260 DO 262 I=1,3
 262 GRAV(I)=RVAR(I+1)
   IF (MISS) GOTO 1010
   GOTO 5
C--------------- SET OR RESET FIX BITS -----------------
 280 GOTO 5
C--------------- SHEAR STIFFNESS ----------------------
 300 ITYP1=IVAR(3)
   ITYP2=IVAR(4)
   IF (MISS) GOTO 1010
   AKS(ITYP1,ITYP2)=RVAR(2)
   AKS(ITYP2,ITYP2)=RVAR(2)
   AKSS=RVAR(2)
   GOTO 5
C--------------- NORMAL STIFFNESS ---------------------
```

```
 320 ITYP1=IVAR(3)
     ITYP2=IVAR(4)
     IF (MISS) GOTO 1010
     AKN(ITYP1,ITYP2)=RVAR(2)
     AKNN=RVAR(2)
     GOTO 5
C-------------- DENSITY ----------------------------
 340 ITYP=IVAR(3)
     IF (MISS) GOTO 1010
     ddd = rvar(2)
     DENS(ITYP)=ddd
     GOTO 5
C-------------- RADIUS -----------------------------
 360 ITYP=IVAR(3)
     IF (MISS) GOTO 1010
     RV2 = RVAR(2)
     IF (2.0*RV2 .LT. AMIN1(DEL(1),DEL(2),DEL(3))) GOTO 362
          NERR = 9
          ERROR = .TRUE.
          GOTO 1000
 362 R(ITYP) = RV2
     GOTO 5
C-------------- COEFFICIENT OF FRICTION ---------------
 380 ITYP1=IVAR(3)
     ITYP2=IVAR(4)
     AMU(ITYP1,ITYP2)=RVAR(2)
     AMU(ITYP2,ITYP2)=RVAR(2)
     AMUU=RVAR(2)
     GOTO 5
C-------------- COHESION -----------------------------
 400 ITYP1=IVAR(3)
     ITYP2=IVAR(4)
     COH(ITYP1,ITYP2)=RVAR(2)
     COH(ITYP2,ITYP2)=RVAR(2)
     COHH=RVAR(2)
     GOTO 5
C-------------- SELECT PLOT OPTIONS -------------------
 420 GOTO 5
C-------------- DAMPING CONSTANTS ---------------------
 440 RV2 = RVAR(3)
     RV1 = RVAR(2)
     IF (MISS) GOTO 1010
     ALPHA = 2.0 * PI * RV2 * RV1
     BETA = RV1 / (2.0 * PI * RV2)
     IF(IVAR(4).EQ.0) GOTO 445
     ALPHA=0.0
     WRITE(lunw,1608)
 445 IF(IVAR(5).EQ.0) GOTO 5
     BETA=0.0
     WRITE(lunw,1609)
     GOTO 5
C-------------- FRACTION OF CRITICAL TIME-STEP ----------
 460 FRAC=RVAR(2)
     IF (MISS) GOTO 1010
```

```
   GOTO 5
C------------- REMOTE COMMAND INPUT -------------
 480 NREC=1
   GOTO 5
C------------- AUTOMATIC PARTICLE GENERATION ---------
C              (GENERATE COMMAND)
 500 CALL test1
   GOTO 5
C------------- PRINTOUT ---------------------------
 520 CALL PRINT
   GOTO 5
C------------- LOCAL COMMAND INPUT --------------
 540 NREC=0
   GOTO 5
C------------- SET BALL VELOCITIES TO ZERO ----------
 560 GOTO 5
C------------- RESET RADIUS VECTOR ------------------
 580 GOTO 5
C------------- COMMAND ITERATION ---------------
 600 GOTO 5
C------------- FORCE MULTIPLIER -----------------
 620 GOTO 5
C---SET ANGULAR VELOCITIES OF ALL BALLS TO BE ZERO--------
 630 GOTO 5
C------------- VELOCITY MULTIPLIER --------------
 640 GOTO 5
C------------- SAVE PROBLEM ------------------
 660 GOTO 5
C------------- servo gain -------------
 680 RV1 = RVAR(2)
    RV2 = RVAR(3)
    IF (MISS) GOTO 1010
    GAIN = RV1
    SERVEM = RV2
    GOTO 5
c------------- history command ---
 700 goto 5

c------------- set Hertz contact parameters ---
 720 hertz = .true.
    ityp  = ivar(4)
    shear(ityp) = rvar(2)
    poiss(ityp) = rvar(3)
    goto 5
c------------- 2-d MODE ---
 740 twod = .true.
    goto 5
c------------- WALL command ---
 750 goto 5
c------------- MODE (servo control) ---
 760 iv = ivar(2)
    if (miss) goto 1010
    if (bad)  goto 1020
    mode = iv
   goto 5
c--------------- BOND command ---
 770 goto 5
c--------------- PROBE command ---
 780 goto 5
C--------------- ROWE command  ------
 800 call rowe
   goto 5
C---------HOLD Z VELOCITY ZERO FOR ROWE----
 820 NUM=0.0
   NUM=RVAR(2)
   forall(i=1:37)v1x(i,1)=-num
   forall(i=366:403)v1x(i,1)=num
   v1x=spread(v1x(:,1),dim=2,ncopies=cor)
   v2x=spread(v1x(:,1),dim=1,ncopies=cor)
   GOTO 5
c
 1000 CONTINUE
   RETURN
C--- missing parameter ---
 1010 NERR = 2
   ERROR = .TRUE.
   GOTO 1000
C--- bad parameter ---
 1020 NERR = 3
   ERROR = .TRUE.
   GOTO 1000
C--- memory overflow ---
 1030 NERR = 4
   ERROR = .TRUE.
   GOTO 1000
 1500 FORMAT(80A1)
 1501 format (1x,a4)
 1601 FORMAT(' >',80A1)
 1603 FORMAT(' COMMAND NOT AVAILABLE')
 1604 FORMAT(30X,'*** NO MORE MEMORY FOR NEW PARTICLES ***')
 1607 FORMAT(30X,'*** ADDRESS OUT OF RANGE ***')
 1608 FORMAT(30X,'MASS DAMPING TERM SET TO ZERO')
 1609 FORMAT(30X,'STIFFNESS DAMPING TERM SET TO ZERO')
 1612 FORMAT(30X,'CURRENT CYCLE COUNT =',I6)
 1613 FORMAT(I3)
 1615 FORMAT(30X,'MUST SPECIFY EDIT FREQUENCY FIRST')
   END
   subroutine gen
   return
   end
   block data ranset
   logical iflag
   common /ran/ iflag,iold
   data iflag /.false./
   end

   SUBROUTINE INITP
C
```

```
C  TO PERFORM CERTAIN OPERATIONS PRIOR TO CYCLING
C
    save
    INCLUDE 'tpm.inc'
    INCLUDE '/usr/include/cm/CMF_defs.h'
    INCLUDE '/usr/include/cm/cmssl-cmf.h'
    logical supout
    common /procom/ supout
    data ak13,ak23 /0.3333333,0.6666667/
C
    AMIN=1.0E20
    DO 10 I=1,NTYPS
    IF(R(I).LE.0.0) GOTO 10
    IF(DENS(I).LE.0.0) GOTO 10
    AMASS(I)=4.0*PI*DENS(I)*(R(I)**3)/3.0
    AMOI(I)=0.4*AMASS(I)*R(I)**2
    AMIN=AMIN1(AMASS(I),AMIN)
  10 CONTINUE
    RMAX=25
    IF(AMIN.LT.1.0E15) GOTO 20
  15 NERR=5
    ERROR = .TRUE.
    GOTO 1000
  20 AKMAX=0.0
    DO 30 I=1,NTYPM
    DO 30 J=1,NTYPM
  30 AKMAX=AMAX1(akn(i,j),AKS(I,J),AKMAX)
    IF(AKMAX.LE.0.0) GOTO 15
    TDEL=FRAC*2.0*SQRT(AMIN/AKMAX)
    if (.not. supout) WRITE(lunw,601) TDEL
    if (.not.supout) write(7,*) tdel
C--- keep boxes clean ---
    IF (NBALL .EQ. 0) GOTO 100
    genflag=.true.
    genflag=.false.
c----- set up gather/scatter routines for globe

c--- apply user strain-rate ---
 100 forall(j=1:3,i=1:3)edgrid(i,j) = eduser(i,j)
c--- set up Hertz parameters ---
    if (hertz) then
          do 140 i = 1,ntyps
             do 130 j = 1,ntyps
                   if (r(i).gt.0.0 .and. r(j).gt.0.0) then
c--- note: we take elas prop of ball I for both
              rbar = 2.0 * r(i) * r(j) / (r(i) + r(j))
              ccn(i,j) = 2.0*sqrt(2.0*rbar)*shear(i) /
    .              (3.0 * (1.0 - poiss(i)))
              ccs(i,j) = 2.0 * shear(i)**ak23
    .              * (3.0*(1.0-poiss(i))*rbar)**ak13
    .              / (2.0 - poiss(i))
                   endif
 130       continue
 140    continue
```

```
      endif
c---
 1000 RETURN
  601 FORMAT(30X,'TIME-STEP = ',1P,E12.4)
      END

      SUBROUTINE CYCLE
C
C  TO CYCLE THROUGH MAIN CALCULATION LOOP
C
      save
      INCLUDE 'tpm.inc'
      include '/usr/include/cm/CMF_defs.h'
      parameter (nsvar=10)
      double precision dxs
      LOGICAL SKIP,SERVO
      include 'matcom.inc'
      common /iccom/ icont
C
      IF(NCYC.EQ.0) GOTO 1000
c--- initialise strain accumulators ---
*      eax  = 0.0
      erad = 0.0
      evol = 0.0
C--------------- check command line for servo parameters -----
      nserv = 0
      icdsrv = 0
      srvval = 0.0
      CALL SETSRV(SERVO)
      IF(ERROR) GOTO 1000
      gt1=GRAV(1)*TDEL
      gt2=GRAV(2)*TDEL
      gt3=GRAV(3)*TDEL
      AT2=ALPHA*TDEL/2.0
      cn1=1.0-AT2
      cn2=1.0/(1.0+AT2)
C
      call cm_timer_clear(1)
      call cm_timer_start(1)
      DO 200 N=1,NCYC
      NTOT=NTOT+1
C--------------- update periodic space ---------------
      IF (servo .or. GMVFLG) then
c*** note: only good for diagonal terms *****
            XMAX(1) = XMAX(1) + EDGRID(1,1) * TDEL * xmax(1)
            XMAX(2) = XMAX(2) + EDGRID(2,2) * TDEL * xmax(2)
            XMAX(3) = XMAX(3) + EDGRID(3,3) * TDEL * xmax(3)
            DEL(1) = XMAX(1) / float(NX(1))
            DEL(2) = XMAX(2) / float(NX(2))
            DEL(3) = XMAX(3) / float(NX(3))
c--- (1-2) shear ---
            xshear = xshear + (   edgrid(1,2)
     .          +xshear * (edgrid(1,1)- edgrid(2,2)) ) * tdel
      endif
```

```
C--------------- SCAN ALL BALLS ----------------------
*    IF(NBALL) 55,55,20
*   20 ITYPS=SPH(typ,:,:)
     tm=TDEL/AMASS(1)
     tmi=TDEL/AMOI(1)
     INDX=1
*    IAC=sph(31,:)
*    DO 30 J=1,6
*    SKIP(J)=IAND(IAC,INDX).NE.0
*   30 INDX=INDX*2
*    skip.eq.false
     SSAMPL = 0.0
     nmm=0
*    NCONT = 0
     SLIDE = 0.0
     CALL MOTION
     CALL CHECK2
     IF (NMM .NE. 0) SLIDE = SLIDE / FLOAT(NMM)
C--------------- SERVO CONTROL ---------------
  60 IF (SERVO) CALL RUNSRV
c--- write out strain and stress ---
c--- let 1 direction be "axial" ---
     eax  = eax   + edgrid(1,1) * tdel
     erad = erad  + 0.5 * (edgrid(2,2) + edgrid(3,3)) * tdel
     evol = evol  + (edgrid(1,1)+edgrid(2,2)+edgrid(3,3)) * tdel
     samvol = xmax(1) * xmax(2) * xmax(3)
     s11 = ssampl(1,1) / samvol
     s22 = ssampl(2,2) / samvol
     s33 = ssampl(3,3) / samvol
     write(6,*)ntot,eax,s11
     CALL PLAT
C
     IF(ERROR) GOTO 1000
 200 CONTINUE
     call cm_timer_stop(1)
     call cm_timer_print(1)
1000 RETURN
1010 NERR = 2
     ERROR = .TRUE.
     RETURN
1020 NERR = 3
     ERROR = .TRUE.
     RETURN
1030 nerr = 13
     error = .true.
     return
2000 format (2i10)
     END


     SUBROUTINE MOTION
C
C   LAW OF MOTION (2- and 3-D)
C
```

```
      save
      INCLUDE 'tpm.inc'
      DATA XRES,THRES /1.0,0.01/

*       AMASS(I)=4.0*PI*DENS(I)*(R(I)**3)/3.0
*       AMOI(I)=0.4*AMASS(I)*R(I)**2
*       tm=TDEL/AMASS(1)
*       tmi=TDEL/AMOI(1)
       t11=0.0
       t12=0.0
       t21=0.0
       t22=0.0
       t31=0.0
       t32=0.0
       t11=4.0*PI*1000*(rd1**3)/3.0
       t12=4.0*PI*1000*(rd2**3)/3.0
       t21=TDEL/t11
       t22=TDEL/t12
       t31=TDEL/(0.4*t11*rd1**2)
       t32=TDEL/(0.4*t12*rd2**2)

c
c 3-D calculation
c ---------------
c
C--- Linear motion ---

       where(.not.layp)
        v1x=(v1x*cn1+f1x*t21+gt1)*cn2
        v1y=(v1y*cn1+f1y*t21+gt1)*cn2
        v1z=(v1z*cn1+f1z*t21+gt1)*cn2
       endwhere
       where(.not.tlayp)
        v2x=(v2x*cn1+f2x*t22+gt1)*cn2
        v2y=(v2y*cn1+f2y*t22+gt1)*cn2
        v2z=(v2z*cn1+f2z*t22+gt1)*cn2
       endwhere

       dt1x = v1x
       dt1y = v1y
       dt1z = v1z
       dt2x = v2x
       dt2y = v2y
       dt2z = v2z
*
*
*

C   (add in grid motion)
       dt1x =dt1x + edgrid(1,1) * (x1x + xd1x)
       . +edgrid(1,2) * (x1y+ xd1y)
       . +edgrid(1,3) * (x1z +xd1z)

       dt1y = dt1y + edgrid(2,1) * (x1x+xd1x)
```

```
  . + edgrid(2,2) * (x1y+ xd1y)
  . + edgrid(2,3) * (x1z +xd1z)

   dt1z = dt1z + edgrid(3,1) * (x1x +xd1x)
  . + edgrid(3,2) * (x1y+ xd1y)
  . + edgrid(3,3) * (x1z+ xd1z)

   dt2x =dt2x + edgrid(1,1) * (x2x + xd2x)
  . +edgrid(1,2) * (x2y+ xd2y)
  . +edgrid(1,3) * (x2z +xd2z)

   dt2y = dt2y + edgrid(2,1) * (x2x+xd2x)
  . + edgrid(2,2) * (x2y+ xd2y)
  . + edgrid(2,3) * (x2z +xd2z)

   dt2z = dt2z + edgrid(3,1) * (x2x +xd2x)
  . + edgrid(3,2) * (x2y+ xd2y)
  . + edgrid(3,3) * (x2z+ xd2z)

    xd1x = xd1x + dt1x *TDEL
    xd1y = xd1y + dt1y *TDEL
    xd1z = xd1z + dt1z *TDEL

    xd2x = xd2x + dt2x *TDEL
    xd2y = xd2y + dt2y *TDEL
    xd2z = xd2z + dt2z *TDEL

   where(ABS( xd1x ).GE.XRES)
    x1x = x1x + xd1x
    xd1x =0.0
   endwhere
   where(ABS( xd1y ).GE.XRES)
          x1y = x1y + xd1y
          xd1y =0.0
   endwhere
   where(ABS( xd1z ).GE.XRES)
          x1z = x1z + xd1z
          xd1z =0.0
   endwhere
   where(ABS( xd2x ).GE.XRES)
          x2x = x2x + xd2x
          xd2x =0.0
   endwhere
   where(ABS( xd2y ).GE.XRES)
          x2y = x2y + xd2y
          xd2y =0.0
   endwhere
   where(ABS( xd2z ).GE.XRES)
          x2z = x2z + xd2z
          xd2z =0.0
   endwhere

C         (periodic space)
    xt1x =  x1x  +  xd1x
```

```
      xt1y = x1y  + xd1y
      xt1z = x1z  + xd1z
      xt2x = x2x  + xd2x
      xt2y = x2y  + xd2y
      xt2z = x2z  + xd2z

      where( xt1x .LT. 0.0)
           x1x =  x1x + XMAX(1)
      endwhere
      where( xt1y .LT. 0.0)
           x1y =  x1y + XMAX(2)
      endwhere
      where( xt1z .LT. 0.0)
           x1z =  x1z + XMAX(3)
      endwhere
      where( xt1x.GE.XMAX(1))
           x1x =  x1x - XMAX(1)
      endwhere
      where( xt1y.GE.XMAX(2))
           x1y =  x1y - XMAX(2)
      endwhere
      where( xt1z.GE.XMAX(3))
           x1z =  x1z - XMAX(3)
      endwhere

      where( xt2x .LT. 0.0)
           x2x =  x2x + XMAX(1)
      endwhere
      where( xt2y .LT. 0.0)
           x2y =  x2y + XMAX(2)
      endwhere
      where( xt2z .LT. 0.0)
           x2z =  x2z + XMAX(3)
      endwhere
      where( xt2x.GE.XMAX(1))
           x2x =  x2x - XMAX(1)
      endwhere
      where( xt2y.GE.XMAX(2))
           x2y =  x2y - XMAX(2)
      endwhere
      where( xt2z.GE.XMAX(3))
           x2z =  x2z - XMAX(3)
      endwhere
C--- spins ---
*     IF(SKIP(I+3)) GOTO 115
      where(.not.walp)
100    tv1x =( tv1x *cn1+ m1x *t31)*cn2
       tv1y =( tv1y *cn1+ m1y *t31)*cn2
       tv1z =( tv1z *cn1+ m1z *t31)*cn2
      endwhere
      where(.not.twalp)
       tv2x =( tv2x *cn1+ m2x *t32)*cn2
       tv2y =( tv2y *cn1+ m2y *t32)*cn2
       tv2z =( tv2z *cn1+ m2z *t32)*cn2
```

```
    endwhere
     td1x =  td1x + tv1x *TDEL
     td1y =  td1y + tv1y *TDEL
     td1z =  td1z + tv1z *TDEL

     td2x =  td2x + tv2x *TDEL
     td2y =  td2y + tv2y *TDEL
     td2z =  td2z + tv2z *TDEL

    where( td1x .GE.THRES.or.- td1x .GE.THRES)
          t1x = t1x + td1x
          td1x =0.0
    endwhere
    where( td1y .GE.THRES.or.- td1y .GE.THRES)
          t1y = t1y + td1y
          td1y =0.0
    endwhere
    where( td1z .GE.THRES.or.- td1z .GE.THRES)
          t1z = t1z + td1z
          td1z =0.0
    endwhere

    where( td2x .GE.THRES.or.- td2x .GE.THRES)
          t2x = t2x + td2x
          td2x =0.0
    endwhere
    where( td2y .GE.THRES.or.- td2y .GE.THRES)
          t2y = t2y + td2y
          td2y =0.0
    endwhere
    where( td2z .GE.THRES.or.- td2z .GE.THRES)
          t2z = t2z + td2z
          td2z =0.0
    endwhere
C--- reset force sums and moment sums ---
    f1x=f11x
    f1y=f11y
    f1z=0.0
    f2x=f22x
    f2y=f22y
    f2z=0.0

     m1x  = 0.0
     m1y  = 0.0
     m1z  = 0.0
     m2x  = 0.0
     m2y  = 0.0
     m2z  = 0.0

C

200  RETURN
     END
```

```
      SUBROUTINE BBTEST
      return
      end

      SUBROUTINE CHECK2
C
C  TO TEST FOR CONTACT BETWEEN BALLS
C
      save
      INCLUDE 'tpm.inc'
      include '/usr/include/cm/CMF_defs.h'

*
*****************************************************************
*
*  we do checks for all contacts
*
*****************************************************************
*
      s1x=0.0
      s1y=0.0
      s1z=0.0
      s2x=0.0
      s2y=0.0
      s2z=0.0
      xxx=0.0
      xyy=0.0
      xzz=0.0
      xsx=0.0
      xsy=0.0
      xsz=0.0
      rdf=50.0
      dif=0.0

       s1x = x1x + xd1x
       s1y = x1y + xd1y
       s1z = x1z + xd1z

       s2x = x2x + xd2x
       s2y = x2y + xd2y
       s2z = x2z + xd2z
C
C---------------- BALL-TO-BALL CONTACT --------------------

        xxx = s1x -s2x
        xyy = s1y -s2y
        xzz = s1z -s2z
        xsx = xxx
        xsy = xyy
        xsz = xzz
       xmb = xmax(1)
       WHERE (ABS(xxx) .gt. RMAX)
            xsx = SIGN(xmb, xxx)
            xsx =  xxx -  xsx
      ENDWHERE
      xmb = xmax(2)
      WHERE (ABS(xyy) .gt. RMAX)
           xsy = SIGN(xmb, xyy)
           xsy =  xyy -  xsy
           xsx =  xxx - dmod(xshear,xmax(2))* xsy
      ENDWHERE
      xmb = xmax(1)
      WHERE(ABS(xsx).gt.RMAX)
           xsx =  xxx - sign(xmb, xsx)
      ENDWHERE
      xmb = xmax(3)
      WHERE (ABS(xzz) .gt. RMAX)
           xsz = SIGN(xmb,  xzz)
           xsz = xzz -  xsz
      ENDWHERE
       xxx =xsx
       xyy =xsy
       xzz =xsz
       xsx = xsx * xsx
       xsy = xsy * xsy
       xsz = xsz * xsz
       xsx= xsx+ xsy+ xsz
       dif=SQRT(xsx)
      where(rd1.ne.0.0)
       rdf = dif-rd1-rd2
      endwhere
      nmm=(count(rdf.le.0.0)-cor)/2
      if(nmm.eq.0)goto 200

      call ford
      call globe
200   return
      end

      SUBROUTINE FORD
C
C  TO TEST FOR CONTACT BETWEEN BALLS
C
      save
      INCLUDE 'tpm.inc'
      include '/usr/include/cm/CMF_defs.h'

*
*  initializing global constants
*

      kkn=AKNN*TDEL
      AKS1=AKSS*TDEL
*
* removes shear and normal forces in broken contacts
*
      udm=0.0
      fmx=0.0
      fmy=0.0
      fmz=0.0
      rnx=0.0
      rny=0.0
      rnz=0.0
      fxx=0.0
      rtx=0.0
      tpx=0.0
      tpy=0.0
      tpz=0.0
      frx=0.0
      fry=0.0
      frz=0.0
      fsx=0.0
      fsy=0.0
      fsz=0.0
      t11=0.0
      t12=0.0
      t13=0.0
      t21=0.0
      t22=0.0
      t23=0.0
      t31=0.0
      t32=0.0
      t33=0.0

      where(rdf.gt.0.0)
         snx=0.0
         sny=0.0
         snz=0.0
         fn1=0.0
      endwhere

      fmx = v1x - v2x
      fmy = v1y - v2y
      fmz = v1z - v2z

*
*****************************************************
*
*  Begin Calculations for FORD
*
***************************************************
*
C  (allow for grid motion)

       fmx= fmx+edgrid(1,1)* xxx
       fmx= fmx+edgrid(1,2)* xyy
       fmx= fmx+edgrid(1,3)* xzz

       fmy= fmy+edgrid(2,1)* xxx
       fmy= fmy+edgrid(2,2)* xyy
       fmy= fmy+edgrid(2,3)* xzz
      fmz= fmz+edgrid(3,1)* xxx
      fmz= fmz+edgrid(3,2)* xyy
      fmz= fmz+edgrid(3,3)* xzz

c
c--- end of regular linear contact section ---
C
      udm =0.0

      rnx = xxx / dif
      rny = xyy / dif
      rnz = xzz / dif

      udm = fmx * rnx+fmy * rny+fmz * rnz
C
      tpx = udm * rnx
      tpy = udm * rny
      tpz = udm * rnz

      frx = fmx - tpx
      fry = fmy - tpy
      frz = fmz - tpz
C
      fsx= xyy*frz-xzz*fry
      fsy= xzz*frx-xxx*frz
      fsz= xxx*fry-xyy*frx

c  we are now dealing with the shear forces


      t11=tdel/xsx
      t21=snx
      t22=sny
      t23=snz

      fsx = fsx * t11
      fsy = fsy * t11
      fsz = fsz * t11

      t31 = snx
      t32 = sny
      t33 = snz

      do 10 n=1,2
      snx = fsy* t23- fsz* t22
      sny = fsz* t21- fsx* t23
      snz = fsx* t22- fsy* t21

      snx = snx+ t31
      sny = sny+ t32
      snz = snz+ t33

      t21 =0.5*(t31 + snx)
      t22 =0.5*(t32 + sny)
      t23 =0.5*(t33 + snz)
10    continue


C we are dealing with angular velocities now

      t21 =rd1* tv1x+rd2*tv2x
      t22 =rd1* tv1y+rd2*tv2y
      t23 =rd1* tv1z+rd2*tv2z

      t31= t22* rnz- t23* rny
      t32= t23* rnx- t21* rnz
      t33= t21* rny- t22* rnx

C---NORMAL FORCE---
      fn1 = fn1-udm*kkn
      where( fn1.LT.0.0)
           snx=0.0
           sny=0.0
           snz=0.0
           fn1=0.0
           rdf=50.0
      endwhere

      nmm=(count(rdf.le.0.0)-cor)/2

C---SHEAR FORCE---
      fmx= 0.0
      t31 = frx - t31
      t32 = fry - t32
      t33 = frz - t33
      snx = snx - t31 * AKS1
      sny = sny - t32 * AKS1
      snz = snz - t33 * AKS1
      fmx = snx*snx+sny*sny+snz*snz
      fmx = (fmx)**0.5

C---CHECK FOR SLIP---
      fxx =0.0
      fxx = AMUU * fn1 + COHH

      where( fxx .eq. 0.0)
           snx =0.0
           sny=0.0
           snz=0.0
      endwhere
      where( fmx .GT. fxx .and. fmx.eq.0.0)
           rtx =0.0
      endwhere
      where( fmx .GT. fxx .and. fmx .ne.0.0)
           rtx= fxx / fmx
      endwhere
      where( fmx .GT. fxx )
```

```
           snx = snx * rtx
           sny = sny * rtx
           snz = snz * rtx
      endwhere
C---RESOLVE FORCES TO GLOBAL DIRECTIONS---

          m1x= sny* rnz- snz* rny
          m1y= snz* rnx- snx* rnz
          m1z= snx* rny- sny* rnx

       f1x = fn1* rnx+ snx
       f1y = fn1* rny+ sny
       f1z = fn1* rnz+ snz

       fsx  =  f1x * (rd1+rd2)
       fsy  =  f1y * (rd1+rd2)
       fsz  =  f1z * (rd1+rd2)

C     (save stress tensor)


       t11=-(fsx* rnx)
       t12=-(fsx* rny)
       t13=-(fsx* rnz)

       t21=-(fsy* rnx)
       t22=-(fsy* rny)
       t23=-(fsy* rnz)

       t31=-(fsz* rnx)
       t32=-(fsz* rny)
       t33=-(fsz* rnz)

200   return
       end

       SUBROUTINE GLOBE
C
C  TO TEST FOR CONTACT BETWEEN BALLS
C
       save
       INCLUDE 'tpm.inc'
       include '/usr/include/cm/CMF_defs.h'
       INCLUDE '/usr/include/cm/cmssl-cmf.h'

       slide=count(fxx.eq.0.0.and.rdf.le.0.0)
       slide=slide+count(fmx.GT.fxx.and.rdf.le.0.0)
       slide=slide/2
       shft=0
       mas=.false.
       mas2=.false.
       where(rdf.le.0.0)mas=.true.
       where(rdf.le.0.0)mas2=.true.
       forall(i=1:cor,j=1:cor,i.ge.j)mas(i,j)=.false.
      forall(i=1:cor,j=1:cor,i.eq.j)mas2(i,j)=.false.
*
*******************************************************
*
* Global Stress Tensors
*
*******************************************************
*
c-----sum up the stress stress tensor


      ssampl(1,1)=sum(t11,mask=mas)
      ssampl(1,2)=sum(t12,mask=mas)
      ssampl(1,3)=sum(t13,mask=mas)
      ssampl(2,1)=sum(t21,mask=mas)
      ssampl(2,2)=sum(t22,mask=mas)
      ssampl(2,3)=sum(t23,mask=mas)
      ssampl(3,1)=sum(t31,mask=mas)
      ssampl(3,2)=sum(t32,mask=mas)
      ssampl(3,3)=sum(t33,mask=mas)

*      CALL CMF_SCAN_ADD( f1x, f1x, 0, 2, cmf_upward,
*     & cmf_inclusive, CMF_none, mas2)
*      CALL CMF_SCAN_ADD( f1y, f1y, 0, 2, cmf_upward,
*     & cmf_inclusive, CMF_none, mas2)
*      CALL CMF_SCAN_ADD( f1z, f1z, 0, 2, cmf_upward,
*     & cmf_inclusive, CMF_none, mas2)

      f1x(:,1)=sum(f1x,dim=2,mask=mas2)
      f1y(:,1)=sum(f1y,dim=2,mask=mas2)
      f1z(:,1)=sum(f1z,dim=2,mask=mas2)

      f1x=spread(f1x(:,1),dim=2,ncopies=cor)
      f1y=spread(f1y(:,1),dim=2,ncopies=cor)
      f1z=spread(f1z(:,1),dim=2,ncopies=cor)
      f2x=spread(f1x(:,1),dim=1,ncopies=cor)
      f2y=spread(f1y(:,1),dim=1,ncopies=cor)
      f2z=spread(f1z(:,1),dim=1,ncopies=cor)

      m1x=rd1*m1x
      m1y=rd1*m1y
      m1z=rd1*m1z

*      CALL CMF_SCAN_ADD( m1x, m1x, 0, 2, cmf_upward,
*     & cmf_inclusive, CMF_none, mas2)
*      CALL CMF_SCAN_ADD( m1y, m1y, 0, 2, cmf_upward,
*     & cmf_inclusive, CMF_none, mas2)
*      CALL CMF_SCAN_ADD( m1z, m1z, 0, 2, cmf_upward,
*     & cmf_inclusive, CMF_none, mas2)
*      mas2=lastloc(mas2,dim=2)


      m1x(:,1)=sum(m1x,dim=2,mask=mas2)
      m1y(:,1)=sum(m1y,dim=2,mask=mas2)
      m1z(:,1)=sum(m1z,dim=2,mask=mas2)
      m1x=spread(m1x(:,1),dim=2,ncopies=cor)
      m1y=spread(m1y(:,1),dim=2,ncopies=cor)
      m1z=spread(m1z(:,1),dim=2,ncopies=cor)
      m2x=spread(m1x(:,1),dim=1,ncopies=cor)
      m2y=spread(m1y(:,1),dim=1,ncopies=cor)
      m2z=spread(m1z(:,1),dim=1,ncopies=cor)
      MEMORY = CMF_AVAILABLE_MEMORY( )
 200  continue
      RETURN
      END

      SUBROUTINE PRINT
C     GENERAL PRINT ROUTINE ... 3-D PROGRAM
C
      save
      INCLUDE 'matcom.inc'
      INCLUDE 'tpm.inc'
      include '/usr/include/cm/CMF_defs.h'
      LOGICAL SFLG, ALLFLG, GAPFLG
      character*1 icom,icom1
      DIMENSION ICOM(57),ICOM1(9),iiFIX(6)
      DATA ICOM
     . /'M','A','P',' ','E','N','T',' ','C','O','N',' ',
     .  'B','A','L',' ','T','Y','P',' ','L','O','A',' ',
     .  'G','R','I',' ','A','N','D',' ','S','T','R',' ',
     .  'I','N','F',' ','P','A','R',' ','C','H','I',' ',
     .  'W','A','L',' ','B','O','N',' ',iterm /
      data icom1 /'A','L','L',' ','G','A','P',' ',iterm/
c
      NARG = 2
    1 CALL MATCH (ICOM,NARG,JUMP)
      IF (.NOT. MISS) GOTO 5
      IF(NARG.NE.1) GO TO 1010
      WRITE(lunw,1600)
      GO TO 1010
    5 IF (.NOT. BAD) GOTO 20
            NERR = 3
            ERROR = .TRUE.
            GOTO 1010
   20 GOTO (100,120,140,160,
     .    180,200,220,1000,
     .    240,260,280,300,
     .    320,340),JUMP
C
C---------------- MEMORY MAP -------------------
  100 GOTO 1000
C---------------- PRINTOUT OF ENTRIES ---------------------
  120 continue
  130 continue
  132 continue
      GOTO 1000
C---------------- PRINTOUT OF CONTACTS --------------------
  140 if(nmm.eq.0)goto 1000
```

```
      WRITE(lunw,1608)
      do 150 i=1,cor
       do 150 j=1,cor
            if(mas(i,j))then
             WRITE(lunw,1609)i,j,snx(i,j),
     .   sny(i,j),snz(i,j),fn1(i,j),rdf(i,j)
            endif
 150 continue
      GOTO 1000
C--------------- PARTICLE PRINTOUT -------------------
 160 fff=0.0
      fob=0.0
      nfob=0
      do 175 i=1,pos
       fff=sqrt((f1x(i,1))**2+(f1y(i,1))**2
     . +(f1z(i,1))**2)
       fob=fob+fff
       nfob=nfob+1
 175 continue
      if(nfob.gt.0)then
       fob=fob/float(nfob)
       write(lunw,1620)fob,nfob
      endif

c     WRITE(lunw,1604)
c     do 170 i=1,pos
c      WRITE(lunw,1605)i,s1x(i,1),s1y(i,1),s1z(i,1),
c    . v1x(i,1),v1y(i,1),v1z(i,1),t1x(i,1)+td1x(i,1),
c    . t1y(i,1)+td1y(i,1),t1z(i,1)+td1z(i,1)
c 170 continue
c 172 WRITE(lunw,1606)
c     do 175 i=1,pos
c     WRITE(lunw,1607)i,f1x(i,1),f1y(i,1),f1z(i,1),
c    . m1x(i,1),m1y(i,1),m1z(i,1),n,rd1(i,1)
c 175 continue

 180 GOTO 1000
C---------------- LOADS ON PLATTENS -----------------
 200 GOTO 1000
C------------- current grid size ---------------
 220 WRITE (lunw,1613) (XMAX(I) , I=1,3), (EDGRID(I,I), I=1,3),
     .              edgrid(1,2)
      write (lunw,1621) xshear
           write(7,*) (xmax(i),i=1,3)
      GOTO 1000
C-------------- measured stress tensor -----------
 240 WRITE (lunw,1614)
      VOL = XMAX(1) * XMAX(2) * XMAX(3)
      DO 245 I = 1,3
           S1 = SSAMPL(I,1) / VOL
           S2 = SSAMPL(I,2) / VOL
           S3 = SSAMPL(I,3) / VOL
              WRITE (lunw,1615) S1, S2, S3
           write(7,*) s1,s2,s3
 245 CONTINUE
cc     if (twod) then
          s0  = (ssampl(1,1)+ssampl(2,2)+ssampl(3,3)) / (3.0*vol)
          ss0  = 0.5 * (ssampl(1,1) + ssampl(2,2)) / vol
          if (ss0 .eq. 0.0) goto 1000
          ss12 = 0.5 * (ssampl(1,2) + ssampl(2,1))
          sdev = sqrt( 0.25*(ssampl(2,2)-ssampl(1,1))**2
     .              +ss12**2 ) / vol
          thet = atan2 ( ss12 , 0.5*(ssampl(2,2)-ssampl(1,1)) )
          thet = thet / degrad
          write (lunw,1622) s0,ss0,sdev,thet
cc     endif
    GOTO 1000
C--------------- general information -------------
 260 IF (COR .EQ. 0) GOTO 1000
C         (compute total ball volume)
    SVOLB = 0.0
    VOLB = 0.0
    TEMP1 = 4.0 * PI / 3.0
*      ITYPS = sph(25,:)
*      RAD   = R(ITYPS)
    SVOLB  = TEMP1 * rrr** 3
    VOLB = sum(SVOLB,dim=1)
    VOLG = XMAX(1) * XMAX(2) * XMAX(3)
C         (scan contacts to find overlap volume)
    if(nmm.eq.0)goto 265
    VOLOV = 0.0
    fnav  = 0.0
    DD1 = 0.0
    DD2 = 0.0
    SDD1 = 0.0
    SDD2 = 0.0
    where(mas)
    SDD1 = (xsx + rd1*rd1-rd2*rd2) / (2.0 * dif)
    SDD2 = (xsx + rd2*rd2-rd1*rd1) / (2.0 * dif)

    DD1 = rd1 - SDD1
    DD2 = rd2 - SDD2

    SDD1 = PI * (DD1*DD1 * (rd1- DD1 / 3.0)
   . +DD2*DD2 * (rd2- DD2 / 3.0))
    endwhere
    VOLOV=sum(SDD1,mask=mas)
    fnav =SUM(fn1,mask=mas)
 C
 265   if(nmm.eq.0)then
      fnav  = 0.0
      volov = 0.0
     endif
     POROS = (VOLG - VOLB + VOLOV) / VOLG
     OVLRAT = VOLOV / VOLG
     COORD = 2.0 * FLOAT(NMM) / FLOAT(COR)
     RCONS = FLOAT(NMM) * (3.0 - 2.0 * SLIDE) / (6.0 * COR)
     if (nmm .gt. 0) fnav = fnav / float(nmm)
```

```
      WRITE (lunw,1618) POROS, NMM, COORD, RCONS, SLIDE, OVLRAT,
     .          fnav
       write(7,*) poros*volg
      GOTO 1000
C-------------- stress partitions --------------
 280 GOTO 1000
C-------------- contact histogram --------------
 300 goto 1000
c-------------- walls -------------------------
 320 goto 1000
c-------------- bond -------------------------
 340 goto 1000
c---
 1000 WRITE(lunw,1610)
      NARG = NARG + 1
      IF (NARG .LE. 40) GOTO 1
 1010 RETURN
C
 1600 FORMAT(30X,'PRINT WHAT ?')
 1601 FORMAT(30X,'PRINTOUT OF ENTRIES, BY BOX ...'
     .   //' -----BOX------      ENTRIES ....'
     .    /' NUMBER ADDRESS')
 1604 FORMAT(30X,'DATA ON PARTICLES ...'/' ADDRESS',4X,
     . 'U(1)',5X,'U(2)',
     . 5X,'U(3)',4X,'UDOT(1)',4X,'UDOT(2)',4X,'UDOT(3)',
     . 1X,' THETA(1)',
     . ' THETA(2) THETA(3)')
 1605 FORMAT(1X,I6,3F9.3,1P,3E11.3,0P,3F9.3)
 1606 FORMAT(/' ADDRESS',3X,'FSUM(1)',4X,'FSUM(2)',4X,'FSUM(3)',
     .  4X,'MSUM(1)',4X,'MSUM(2)',4X,'MSUM(3)',' STYPE MTYPE')
 1607 FORMAT(1X,I6,6E11.3,I6,F5.2)
 1608 FORMAT(30X,'CONTACT DATA ...'/'     BALL1 BALL2)'
     . ,6X,'FS(1)',6X,'FS(2)',6X,'FS(3)',9X,'FN'
     . ,6X,'XC(1)',6X,'XC(2)',6X,'XC(3)',6X,'GAP')
 1609 FORMAT(1X,3X,2I6,1P,5E11.3)
 1610 FORMAT(' ')
 1611 FORMAT(30X,'MEMORY MAP ...'/
     . ' M1 =',I6,' M1A =',I6,' M2 =',I6,' M2A =',I6,
     . ' M3 =',I6,' M3A =',I6,' M4 =',I6,' M5 =',I6/)
 1612 FORMAT(1X,3(1P,2E11.3,2X))
 1613 FORMAT (' Current grid size -',1P,3E13.5/' Rates -',1P,4E11.3)
 1614 FORMAT (' Measured stress tensor ...')
 1615 FORMAT (1X,1P,3E11.3)
 1616 FORMAT (1X,I3,I8,2I6,' ERROR - ZM is zero')
 1617 FORMAT ('    (overlap is larger than tolerance)')
 1618 FORMAT
     . ('  Porosity  Contacts  Coord-num  Cons-rat   Sliding',
     . 2x,'Ovlap-rat',5x,'av. Fn'
     . / 1X,F11.3,I11,3F11.3,F11.4,1p,e11.3)
 1619 FORMAT (1X,I3,I8,2I6,1P,2E11.3)
 1620 format (' average fob, num:',1p,e11.3,i6)
 1621 format (' 1-2 shear strain = ',1p,e11.3)
 1622 format ('    sig-0    sum/2     s-dev     theta'
     .       /1x,1p,3e11.3,0p,f10.3)
```

```
1630 format (1x,1p,3e11.3)
     END

     SUBROUTINE SETSRV(SERVO)
     save
     INCLUDE 'tpm.inc'
     parameter (nsvar=10)
     LOGICAL SERVO
     character*1 icom
     include 'matcom.inc'
     DIMENSION ICOM(30)
     DATA ICOM /'S','0',' ','S','1','M','S','2',' ','S','3','3',' ',
     .         'R','I','N',' ','S','1','1',' ','S','2','2',' ',
     .         'R','O','T',' ',
     .         iterm/
     SERVO = .FALSE.
     NSERV = 0
     NPAR = 3
   5 CALL MATCH(ICOM,NPAR,JUMP)
             IF(MISS) GOTO 8
             IF(BAD)  GOTO 1020
     if (nserv+1 .gt. nsvar) then
             nerr = 12
             error = .true.
     endif
     NSERV = NSERV + 1
     ICDSRV(NSERV) = JUMP
     SRVVAL(NSERV) = RVAR(NPAR+1)
             IF(MISS) GOTO 1010
     if (jump .eq.4) srvval(nserv) = cos (2.0 * srvval(nserv)
     .                       * degrad)
     SERVO = .TRUE.
     NPAR = NPAR+2
     GOTO 5
1010 SERVO = .FALSE.
     NERR = 2
     ERROR = .TRUE.
     RETURN
1020 NERR = 3
     ERROR = .TRUE.
     RETURN
   8 MISS = .FALSE.
     RETURN
     END
     SUBROUTINE RUNSRV
     save
     logical velflg
     parameter (nsvar=10)
     include 'matcom.inc'
     INCLUDE 'tpm.inc'
     dimension detr(3,3),etr(3,3)
c--- flag to request direct velocity control ---
     velflg = mode .eq. 1
c
```

```
    do 5 j = 1,3
    '    do 4 i = 1,3
           detr (i,j) = 0.0
  4   continue
  5 continue
c
   samvol = xmax(1) * xmax(2) * xmax(3)
   DO 100 I=1,NSERV
   GOTO(10,20,30,40,50,60,70), ICDSRV(I)
C
C  SERVO CONTROL FOR MEAN STRESS
C
  10   S0MES = (SSAMPL(1,1) + SSAMPL(2,2) + SSAMPL(3,3)) /
   .      (3.0 * samvol)
         S0REQ=SRVVAL(I)
         DED = ( GAIN * (S0REQ - S0MES) ) / 3.0
         if (velflg) then
            if (abs(ded) .gt. servem) ded = sign(servem,ded)
            edgrid(1,1) = ded
            edgrid(2,2) = ded
            edgrid(3,3) = ded
            eduser(1,1) = ded
            eduser(2,2) = ded
            eduser(3,3) = ded
         else
            detr(1,1) = detr(1,1) + DED
            detr(2,2) = detr(2,2) + DED
            detr(3,3) = detr(3,3) + DED
         endif
    GOTO 100
C
C  SERVO CONTROL FOR (S11+S22)/2
C
  20 S12MES = (SSAMPL(1,1)+SSAMPL(2,2)) /
   .     (2.0 * samvol)
    S12REQ = SRVVAL(I)
    DED12  = 0.5 * GAIN * (S12REQ - S12MES)
    detr(1,1) = detr(1,1) + DED12
    detr(2,2) = detr(2,2) + DED12
    GOTO 100
C
C  SERVO CONTROL FOR S33
C
  30 continue
    DED      = GAIN * (srvval(i) - ssampl(3,3)/samvol)
    if (velflg) then
          if (abs(ded) .gt. servem) ded = sign(servem,ded)
          edgrid(3,3) = ded
          eduser(3,3) = ded
    else
          detr(3,3) = detr(3,3) + DED
    endif
    goto 100
c
```

```
c servo control for ring-shear simulation (keeps constant angle of p.s.)
c
  40 S33MES = SSAMPL(3,3) / samvol
     S33REQ = 0.5 * (ssampl(1,1)+ssampl(2,2)
     .        +(ssampl(1,1)-ssampl(2,2)) * SRVVAL(I) ) / samvol
     DED33  = GAIN * (S33REQ - S33MES)
     detr(3,3) = detr(3,3) + DED33
     goto 100
C
C control for s11
c
  50 continue
     DED     = GAIN * (srvval(i) - ssampl(1,1)/samvol)
     if (velflg) then
          if (abs(ded) .gt. servem) ded = sign(servem,ded)
          edgrid(1,1) = ded
          eduser(1,1) = ded
     else
          detr(1,1) = detr(1,1) + DED
     endif
     goto 100
c
c control for s22
c
  60 continue
     DED     = GAIN * (srvval(i) - ssampl(2,2)/samvol)
     if (velflg) then
          if (abs(ded) .gt. servem) ded = sign(servem,ded)
          edgrid(2,2) = ded
          eduser(2,2) = ded
     else
          detr(2,2) = detr(2,2) + DED
     endif
     goto 100
c
c
c control for rotation of pr. axes: deviator is kept at given value;
c   applied strain-rate is normal to stress vector, with magnitude
c   of max. strain rate and sign taken from given deviator.
c
  70 continue
     s11m = ssampl(1,1) / samvol
     s22m = ssampl(2,2) / samvol
     s33m = ssampl(3,3) / samvol
     s0   = (s11m + s22m + s33m) / 3.0
     s12m = 0.5 * (ssampl(1,2) + ssampl(2,1)) / samvol
     sdev = sqrt(0.25*(s22m-s11m)**2 + s12m*s12m)
     sn   = s12m / sdev
     cs   = 0.5 * (s22m -s11m) / sdev
     if (velflg) then
          udr1  = gain * (abs(srvval(i)) - sdev)
          udc  = sign (servem,srvval(i))
          bb   = gain * (s22m - s33m)
c*** temp ***
```

```
          udv = gain * (-1.38e-5 - s0)
c************
          aa  = udrl * cs - udc * sn
          edgrid(1,1) = (udv - 2.0*aa - bb) / 2.0
          edgrid(2,2) = (udv + 2.0*aa - bb) / 2.0
          edgrid(3,3) = bb
          edgrid(1,2) = 2.0 * (udrl * sn + udc * cs)
          eduser(1,1) = edgrid(1,1)
          eduser(2,2) = edgrid(2,2)
          eduser(3,3) = edgrid(3,3)
          eduser(1,2) = edgrid(1,2)
    else
          edx  = 0.5 * (edserv(2,2) - edserv(1,1))
          edy  = edserv(1,2)
c--- resolve strainrates into circum. component ---
          edc  = edy * cs - edx * sn
c--- increment for circum. adjsutment ---
          dec  = sign(0.1 * servem,srvval(i)) - edc
c--- resolve back ---
          dex  = -dec * sn
          dey  =  dec * cs
          detr(1,2) = detr(1,2) + dey
          detr(2,2) = detr(2,2) + dex
          detr(1,1) = detr(1,1) - dex
c--- increments for radial adjustments ---
          ded  = gain * (abs(srvval(i)) - sdev)
          detr(1,2) = detr(1,2) + ded * sn
          detr(2,2) = detr(2,2) + ded * cs
          detr(1,1) = detr(1,1) - ded * cs
c--- servo to keep 33-stress = 22-stress, for ring shear device ---
          ded  = gain * (s22m - s33m)
          detr(3,3) = detr(3,3) + ded
    endif
    goto 100
c
  100 CONTINUE
    if (velflg) return
c
c test for strain-rate limit
c and add in to user-given strain rate
c
    do 200 j = 1,3
          do 150 i = 1,3
            edserv(i,j) = edserv(i,j) + detr(i,j)
            if (abs(edserv(i,j)) .gt. servem) then
                  edserv(i,j) = sign(servem, edserv(i,j))
            endif
            edgrid(i,j) = eduser(i,j) + edserv(i,j)
  150   continue
  200 continue
c---
    RETURN
    END
```

```
      subroutine tidy
C
C TO ELIMINATE BLANKS, ETC. FROM INPUT
C LINE AND MAKE INDEX TO LOCATION OF PARAMETERS
C
      save
      logical sep
      include 'matcom.inc'
      BAD=.FALSE.
      NCHAR = 0
      L1=1
      L2=1
      NPAR=0
      DO 5 I=1,40
    5 LPNT(I)=0
      GOTO 20
C---NOW WITHIN A STRING---
   10 IF(SEP(LINE(L2))) GOTO 30
   15 LINE(L1)=LINE(L2)
      L1=L1+1
      L2=L2+1
      IF(L2.LE.80) GOTO 10
      GOTO 50
C---NOW IN A GAP---
   20 IF(.NOT.SEP(LINE(L2))) GOTO 40
   30 L2=L2+1
      IF(L2.LE.80) GOTO 20
      GOTO 50
C---START OF A STRING---
C---FIRST CHECK FOR TERMINATOR
   40 IF((LINE(L2).EQ.'*') .OR. (LINE(L2).EQ.';')) GOTO 50
      NPAR=NPAR+1
      LPNT(NPAR)=L1
      GOTO 15
   50 LPNT(NPAR+1)=L1
      RETURN
      END
      LOGICAL FUNCTION SEP(C)
C
C RETURNS .TRUE. IF C IS A SEPARATOR
C
      save
      CHARACTER*1 C, CSEP(6)
      DATA CSEP /' ',',','/','=',')','('/
C
      ic = ichar(c)
      if ((ic.ge.97) .and. (ic.le.122)) c = char(ic-32)
      DO 10 I=1,6
           IF(C.EQ.CSEP(I)) THEN
            SEP = .TRUE.
            RETURN
           END IF
   10 CONTINUE
      SEP=.FALSE.
      RETURN
      END
      SUBROUTINE MATCH (NTAB,NPAR,JUMP)
C
C TO MATCH INPUT STRING TO KEYWORD IN TABLE
C
C INPUT:  NTAB  TABLE OF KEYWORDS IN CHARACTER*1 FORMAT, SEPARATED
C               BY BLANKS AND TERMINATED WITH ITERM. ALL KEYWORDS
C               MUST CONSIST OF AT LEAST 2 CHARACTERS, UNLESS
C               THEY REALLY ARE SINGLE-LETTER COMMANDS.
C         NPAR  PARAMETER NO. IN INPUT LINE (STORED IN ARRAY LINE()
C               WHICH IS ALSO IN CHARACTER*1 FORMAT)
C
C OUTPUT:       JUMP  DISPATCH NUMBER CORRESPONDING
C                     TO POSITION OF KEYWORD IN NTAB.
C               BAD   .TRUE. FOR MISSING PARAMETER OR STRING NOT FOUND
C               MISS  .TRUE. FOR MISSING PARAMETER
C
      save
      CHARACTER*1 IBLK, NTAB(1), NTI
      include 'matcom.inc'
      DATA IBLK /' '/
C
      NPBAD = NPAR - 1
      L1=LPNT(NPAR)
      L2=LPNT(NPAR+1)-1
      IF (L2.LT.0) THEN
           JUMP=1
           MISS = .TRUE.
           BAD  = .TRUE.
           RETURN
      ENDIF
      I=0
      NT=0
   40 NT=NT+1
      IF(NTAB(I+2).EQ.IBLK.AND.L2.GT.L1) GOTO 80
      DO 50 L=L1,L2
      I=I+1
      NTI=NTAB(I)
      IF(NTI.NE.LINE(L)) GOTO 70
   50 CONTINUE
   60 JUMP=NT
           BAD=.FALSE.
           MISS = .FALSE.
           RETURN
   65 IF (L.EQ.L1) THEN
           JUMP = 2
           MISS = .FALSE.
           BAD  = .TRUE.
           RETURN
       ENDIF
       GOTO 60
   70 IF(NTI.EQ.IBLK.OR.NTI.EQ.ITERM) GOTO 65
   80 I=I+1
```

```
      NTI=NTAB(I)
      IF(NTI.EQ.IBLK) GOTO 40
      IF(NTI.NE.ITERM) GOTO 80
            JUMP = 2
            MISS = .FALSE.
            BAD=.TRUE.
            RETURN
      END
      SUBROUTINE VAR(NPAR)
C
C COMMON ROUTINE FOR IVAR & RVAR
C
      save
      CHARACTER*1 IBUF(20), IBL
      CHARACTER*20 BUF
      include 'matcom.inc'
      COMMON /CVAR/ BUF
      DATA IBL /' '/
C
      NPBAD = NPAR - 1
      NP=NPAR
      DO 10 I=1,20
   10 IBUF(I)=IBL
      LL=LPNT(NP)
      NUM=LPNT(NP+1)-LL
      NUM=MIN0(NUM,20)
      IF (NUM.LE.0) THEN
            MISS=.TRUE.
            WRITE (BUF,100) IBUF
            RETURN
      ENDIF
      DO 20 L=1,NUM
            N1=20-NUM+L
            IBUF(N1)=LINE(LL)
   20 LL=LL+1
      WRITE (BUF,100) IBUF
      MISS = .FALSE.
      RETURN
  100 FORMAT (20A1)
      END
      FUNCTION IVAR(NPAR)
C
C TO RETURN INTEGER VALUE OF PARAMETER NPAR
C   MISS  IS SET .TRUE. IF MISSING
C   BAD   IS SET .TRUE. IF FORMAT BAD
C
      save
      CHARACTER*20 BUF
      include 'matcom.inc'
      COMMON /CVAR/ BUF
C
      IF (BAD) GOTO 100
      CALL VAR(NPAR)
      READ (BUF,200,ERR=100) IV
      IVAR=IV
      RETURN
  100 BAD = .TRUE.
      IVAR=0
      RETURN
  200 FORMAT(I20)
      END
      FUNCTION RVAR(NPAR)
C
C TO RETURN REAL VALUE OF PARAMETER NPAR
C   MISS  IS SET .TRUE. IF MISSING
C   BAD   IS SET .TRUE. IF FORMAT BAD
C
      save
      CHARACTER*20 BUF
      include 'matcom.inc'
      COMMON /CVAR/ BUF
C
      IF (BAD) GOTO 100
      CALL VAR(NPAR)
      READ (BUF,200,ERR=100) RV
      RVAR=RV
      RETURN
  100 BAD = .TRUE.
      RVAR=0.0
      RETURN
  200 FORMAT(F20.0)
      END

      subroutine test1
c test is used to generate selected Particles
c
      save
      INCLUDE 'tpm.inc'
      GENFLAG=.TRUE.
      nreq = 403
      nball=403
      do 5 i=1,nreq
       read(4,1605)con(i,1),con(i,2),con(i,3)
       read(8,*)walp(i,1),layp(i,2)
       read(9,*)con1(i,1),con1(i,2)
5      continue
*       con( 1,1)=82
*       con( 1,2)=105
*       con( 1,3)=100
*       con( 2,1)=100
*       con( 2,2)=100
*       con( 2,3)=100
*       con( 3,1)=118
*       con( 3,2)=95
*       con( 3,3)=100
*       con(typ,:,:)=1
*       con(mat,:,:)=1
*       rrr(1:3)=15
```

```
      rrr(1:50)=15
      rrr(51:150)=10
*
* Loading the x coordinates
*
      do 10 i=1,cor
       do 10 j=1,cor
       x1x(i,j)=con(i,1)
       x2x(i,j)=con(j,1)
10    continue
*
* Loading the y coordinates
*
      do 20 i=1,cor
       do 20 j=1,cor
       x1y(i,j)=con(i,2)
       x2y(i,j)=con(j,2)
20    continue
*
* Loading the z coordinates
*
      do 30 i=1,cor
       do 30 j=1,cor
       x1z(i,j)=con(i,3)
       x2z(i,j)=con(j,3)
30    continue
*
* Loading the radius
*
      do 40 i=1,cor
       do 40 j=1,cor
       rd1(i,j)=rrr(i)
       rd2(i,j)=rrr(j)
40    continue
*
* Loading the fx coordinates
*
      do 50 i=1,cor
       do 50 j=1,cor
       f11x(i,j)=con1(i,1)
       f22x(i,j)=con1(j,1)
50    continue
*
* Loading the fy coordinates
*
      do 60 i=1,cor
       do 60 j=1,cor
        f11y(i,j)=con1(i,2)
        f22y(i,j)=con1(j,2)
60    continue
*
* Loading the mas1 coordinates
*
      walp=spread(walp(:,1),dim=2,ncopics=cor)
      twalp=spread(walp(:,1),dim=1,ncopies=cor)
      layp=spread(layp(:,1),dim=2,ncopies=cor)
      tlayp=spread(layp(:,1),dim=2,ncopies=cor)
      forall(i=1:cor,j=1:cor,i.eq.j)walp(i,j)=.false.
      forall(i=1:cor,j=1:cor,i.eq.j)layp(i,j)=.false.
      forall(i=1:cor,j=1:cor,i.eq.j)twalp(i,j)=.false.
      forall(i=1:cor,j=1:cor,i.eq.j)tlayp(i,j)=.false.
*       forall(i=1:nreq)ityps(i)=sph(typ,i,1)
*       forall(i=1:nreq)sph(rdd,i,:)=r(ityps(i))
      write (lunw,1610) nball
      genflag=.false.
      return
1610  format(1x,i3,' particles have been generated in 0 tries')
1605  FORMAT(1X,6X,3F9.3)

1607  FORMAT(1X,6X,66X,2F6.0)
      end

      SUBROUTINE PLAT
C     TO CREATE THE GRAPH OF OUTPUT
C
      save
      include 'matcom.inc'
      include 'trbcom.inc'
      real ROWC,DISPL,HGHT
      DIMENSION FM(3),FP(3)
C---------------- LOADS ON PLATTENS -----------------
      if(ntot.le.600)goto300
      FM=0.0
      FP=0.0
      where(v1x.lt.0.0)
     .   fm(1)=sum(f1x(1,:),dim=2,mask=walp)/2
      where(v1y.lt.0.0)
     .   fm(2)=sum(f1y(1,:),dim=2,mask=walp)/2
      where(v1z.lt.0.0)
     .   fm(3)=sum(f1z(1,:),dim=2,mask=walp)/2

      where(v1x.gt.0.0)
     .   fp(1)=sum(f1x(1,:),dim=2,mask=walp)/2
      where(v1y.gt.0.0)
     .   fp(2)=sum(f1y(1,:),dim=2,mask=walp)/2
      where(v1z.gt.0.0)
     .   fp(3)=sum(f1z(1,:),dim=2,mask=walp)/2

      ROWC=FM(3)/(4.48E4*5E4)
      displ=-2*(25-s1z(1,:))
      DHGHT=ABS(s1z(1,:)-s1z(366,:))
      STRAINS=DISPL/DHGHT
      WRITE(6,*)STRAINS,ROWC,NTOT
  300 RETURN
 1612 FORMAT(1X,3(1P,2E11.3,2X))
      END
```

REFERENCES

Almasi, George S. and Gottlieb, A.(1994).*Highly Parallel Computing*,The Benjamin/Cummings Publishing Company, Inc.,second edition,New York

Amdahl,G.M (1967) "Validity of Single-Processor Approach to Acheiving Large-Scale Computing Capability", *Proc. AFIPS Conf.*, pp. 483-485, Reston, VA.

Bathurst, Richard John (1985),"A Study of Stress and Anisotropy in Idealized Granular Assemblies".PhD dissertation, Civil Engineering, Queen's University at Kingston, Ontario, Canada

C.S. Chang, A. Misra and S.S. Sundaram(1990),"Micromechanical modelling of cemented sands under low amplitude oscillations", *J.Geotechnique* 40, No.2,251-263

Chang C.S and Acheampong K.B.(1993) ,"Accuracy and Stability for Static Analysis Using Dynamic Formulation in Discrete Element Methods," *Proceedings of the 2nd International Conference on Discrete Element Methods(DEM)*,pp.379-389,MIT,Boston,MA

Chen , Y.C.(1986) "Experimental Determination of Fabric for Granular Material", Ph.D. thesis, Civil Engineering , Cornell University,Ithaca, New York

Cundall P.A. and Strack O.D.L(1978), "The Distinct Element Method As A Tool For Research In Granular Media", Report to the National Science Foundation Concerning NSF Grant ENG76-20711, Part I

---(1979a), "The Distinct Element Method As A Tool For Research In Granular Media", Report to the National Science Foundation Concerning NSF Grant ENG76-20711, Part II

---(1979b). "A Discrete Numerical Model for Granular Assemblies", *J.Geotechnique*, vol. 29 , pp. 47-65

---(1979c). "The Development of Constitutive Laws for Soils Using the Distinct Element Method", *Proc. 3rd Int. Conf. on Numerical Methods in Geomechanics.*, Aachen, vol.1, Balkema, Rotterdam, pp.289-317

Cundall, P.A.(1971), "A Computer Model for Simulating Progressive, large scale Movements in Block Rock Systems", *Proc. Int. Symp. on Rock Fracture*, Nancy, France, II-8

Cundall P.A.,Drescher A.& Strack O.D.L.(1982),"Numerical experiments on granular assemblies; Measurements and observations",*IUTAM conference on Deformation and Failure of Granular Materials*/Delft/Aug.31-Sept3, pp.355-370

Cundall, P.A., and Hart, R.D.(1990), "Numerical Modeling of Discontinua", Itasca Consulting Group, Inc., Minneapolis, Minnesota

Dantu, P. (1957) "Contribution a l'etude mechanique et geometrique des milieux pulverulents",. *Proc. 4th Int. Conf. Soil Mech. Foundation Eng.*, London 1,144ff

De Josselin de Jong, G. & Verruijt, A.(1969),"Etude photo-elastique d'un empilement de disques.", *Cahiers du Groupe Francais de Rheologie II*, No. 1, 73-86

Deresiewcz, H. (1958)."Stress-strain relations for a simple model of a granular medium." J. Appl. Mech., Trans. ASME, 25(3), 402-406.

Drescher A. and De Josselin de Jong, G.(1972), "Photoelastic Verification Of a Mechanical Model For the Flow Of a Granular Material", *J.Mech.Phys. Solids*, Vol.20, pp 337-351

Duffy, J. (1959)."A differential stress-strain relation for the hexagonal close packed array.", *J.Appl. Mech., Trans.* ASME, pp. 88-94.

Ghaboussi, J., Basole, M. & Ranjithan, S.(1993)"Three Dimensional Discrete Element Analysis on Massively Parallel Computers",*Proceedings of the Second International Conference on Discrete Element Methods*, Massachusetts Institute of Technology, Boston, MA, March 18-19

Geist, A. , Beguelin, A., et. al (1994). *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*,The MIT Press Cambridge, Mass., London ,England

Gili, J.A, and Alonso, E.E.(1988),"Discontinuous numerical model for partially saturated soils at low saturation", *Proceed. 6th Int. Conf. on Numerical Methods in Geomechanics*. Swoboda (ed.). Balkema Rotterdam. pp. 365-372

Gili, J.A(1988), "Modelo Microestructural para medios granulares no saturados",Tesis Doctoral. Univ. Politec. de Catalunja. , Spain, 621 pag, Julio

Hadj Ounis and Goodarz Ahmadi(1989), "Motions of Small Rigid Spheres in Simulated Random Velocity Field",*Journal of Engineering Mechanics*, Vol.115, No. 10, October

Heermann D.W. and Burkitt A.N.(1990). *Parallel Algorithms in Computational Science*. Number 24 in Springer Series on Informational Sciences.Springer-Verlag,N.Y.

Hustrihild, Andrew I.(1995), "Parallel Implementation of the Discrete Element Method",Colorado School of Mines, March 8th, Internet Online (http:\\ ppl.mines.colorado.edu:80/dempaper/dempaper.html)

Hwang, Kai (1993),*Advanced Computer Architecture:Parallelism,Scalability, Programmability*,McGraw-Hill Inc., N.Y.

Jenkins, James T.(1988), "Volume Change in Small Strain Axisymmetric Deformations of a Granular Material", *Micromechanics of Granular Materials*, Elsevier Science Publishers B.V. , Amsterdam

Krawietz, A. (1982) "Some features of the gross behavior of granular media derived from micromechanics"*,IUTAM Conference on Deformation and Failure of Granular Materials*/Delft/Aug.31-Sept.3

Kuraoka, Senro(1994),"Anisotropic Stiffness and Circulation Flow of Sand: Application for the Expendable Pattern Casting", Ph.D. Thesis, Civil and Environmental Engineering,University of Wisconsin-Madison, Wisconsin

Meegoda N.J.and Washington D.W.(1994),"Massively parallel computers for microscopic modeling of soils", *Proceedings of the Eighth International Conference on Computer Methods and Advances in Geomechanics*,Morgantown, West Virginia, May 22-24, pp.617-622

Mindlin, R.D.(1949),"Compliance of Elastic Bodies in Contact,"*J.Appl. Mech. ASME*, vol.71, pp.A259-268

Ng. T.T.(1989),"Numerical Simulation of Granular Soil under Monotonic and Cyclic Loading:a Particulate Mechanics Approach", Ph.D. thesis, Civil Engineering, Renesselaer Polytechnic Institute

Oda, M. and Konoshi, J.(1974),"Microscopic deformation mechanism of granular material in simple shear",*Soils and Foundations, Japanese Society of Soil Mechanics and Foundation Engineering*, 14, No. 4, 25-38

Ratnaweera, P.(1992) "The Influence of Chemical Contaminants on Shear Strength and Stress-Strain Behavior of Clay Soils", PhD dissertation, Civil Engineering, New Jersey Institute of Technology,Newark, N.J.

Rodriguez-Ortiz, J.M(1974)., "Estudio del Comportamiento de medios granulares heterogeneos mediante modelos discontinous analogicos y matematicos".PhD thesis, Universidad Politecnica de Madrid.,Spain

Rowe, P.W.(1962), "The stress dilatancy relation for static equilibrium of an assembly of particles in contact", *Proceeding of the Royal Society*, 269, No.1339, 500-527

Shi, G.(1988),"Discontinuous deformation analysis- a new numerical model for the statics and dynamics of block systems",Ph.D thesis, Univ. of California, Berkeley,Ca.

Terzaghi, K.(1920),"Old earth-pressure theories and new test results", *Engineering News-Record*,85,no.14.(1960 Reprinted in From theory to practice in soil mechanics.: J.Wiley and Sons., N.Y)

Thorton, C., and Barnes, D.J.(1986),"Computer Simulated Deformation of Compact Granular Assemblies,"*J. Acta Mechanica*, vol. 64, pp. 45-61

Ting, J.M.et. al(1989),"Discrete Numerical Model for Soil Mechanics," *J.Geotech. Eng.*, vol 115, no. 3, pp.379-398

Walton, Otis R(1992*), Numerical Simulation Of Inelastic, Frictional Particle-Particle Interactions*,Particulate Two-Phase Flow,M.C. Roco(Eds),Butterworth-Heinemann,Chapter 25

Walton, Otis. R and Braun, Robert L.(1986),"Viscosity, Granular-Temperature, and Stress Calculations For Shearing Assemblies of Inelastic, Frictional Disks",*Journal of Rheology*, John Wiley & Sons, Inc.,N.Y. 30(5), 949-980

Washington D.W and Meegoda N.J(1996),"Micromechanical Simulation of Geotechnical Problems using Massively Parallel Supercomputers", *Proceedings of the Eleventh ASCE Engineering Mechanics Conference*, Fort Lauderdale, Florida, May 19-22,to be presented

Washington D.W.& Meegoda N.J.(1996),"DEM Simulation of Geotechnical Problems using Massively Parallel Supercomputers", *submitted to Journal of ASCE Geotechnical Engineering*

William, J. & Mustoe, G. (1993) "Preface",*Proceedings of the Second International Conference on Discrete Element Methods*, Massachusetts Institute of Technology, Boston, MA, March 18-19