THIRD AUSTRALIAN

SUPERCOMPUTER CONFERENCE

PROCEEDINGS

University of Melbourne

3 to 6 December 1990

Organised by:

Strategic Research Foundation

The University of Melbourne

PARALLEL PROCESSING FOR THE DISTINCT ELEMENT METHOD OF STRESS ANALYSIS

G.K. Egan and M.A. Coulthard

1. Introduction

Computational stress analysis is now widely used in geomechanics for back analysis of observed rock mass behaviour around surface and underground excavations and as a tool for excavation design in mining and civil engineering. The distinct element (DE) method, which represents a rock mass as a discontinuum, has been shown to be more realistic than finite element (FE) or boundary element (BE) (continuum) methods for modelling systems such as subsiding strata over underground coal mine excavations. However, whereas even 3D FE and BE analyses can now be performed readily on engineering workstations or the more powerful personal computers, the DE method generally requires orders of magnitude more computer processing time for analyses of comparable complexity. This has so far prevented the DE method from being applied widely in excavation design in industry.

Some significant work has been done, particularly at several of the U.S. National Laboratories, on developing DE codes which can be run very much faster by utilising the vector-processing capabilities of supercomputers. These machines, and therefore these DE codes, are generally not available to engineers in industry, so other options for speeding up DE programs need to be explored.

Most DE codes are based upon an explicit time integration of Newton's laws of motion for each DE, usually involving many thousands of time steps or solution cycles in a full analysis. The explicit numerical method implies that, within each cycle, calculations for each DE could be carried out in parallel. The potential therefore exists for creating much faster DE codes, which would run on moderately-priced and therefore more accessible machines, through the use of parallel processing.

Several different software techniques and hardware options have been applied to develop parallel processing versions of two relatively simple 2D DE stress analysis codes.

The changes made to the programs will be outlined, times for the original and modified versions reported, and some preliminary conclusions drawn regarding the usefulness of various parallel processing options for DE stress analysis codes.

2. The Distinct Element Method

The DE method of stress analysis was introduced in [1] to deal with problems in rock mechanics which could not be treated adequately by the conventional continuum methods. The earliest DE programs (e.g. program RBM in [2]) assumed that

---

G.K. Egan is Professor of Computer Systems Engineering and Director of the Laboratory for Concurrent Computing Systems at the Swinburne Institute of Technology, John Street, Hawthorn 3122, ACSNet: gke@stan.xx.swin.oz.au.

Dr. M.A. Coulthard is Principal Research Scientist at the CSIRO Division of Geomechanics, P.O. Box 54, Mt. Waverley 3149, ACSNet: mac@dogmelb.dog.oz.au.

the blocks were rigid, so that all deformations within the system took place at the block interfaces. A second program described in [2], SDEM, allowed modelling of three simple modes of deformation of each block - two compressive and one shear mode. The DE programs which are most widely used at present are UDEC [3] and 3DEC [4]; the blocks in each of these may be modelled as fully deformable via internal finite difference zoning.

The main factor working against the adoption of these programs for routine engineering design of excavations in highly jointed rock is the very large computer execution times which are required for analyses involving substantial numbers of distinct elements.

The feasibility of parallel processing has been studied in this paper, using two of the simpler distinct element stress analysis programs which are available:

- **DECYL**: 2-D analysis of systems of interacting, rigid circular DEs of equal radius [5];
- **SDEM**: 2-D analysis of systems of interacting, simply deformable polygonal DEs [2][6].

2.1 Theoretical basis

Most DE programs are based on force-displacement relations describing block interactions and Newton's second law of motion for the response of each block to the unbalanced forces and moments acting on it.

The normal forces developed at a point of contact between blocks are calculated from the notional overlap of those blocks and the specified normal stiffness of the inter-block joints. Tensile normal forces are usually not permitted, i.e. there is no restraint placed upon opening of a contact between blocks.

Shear interactions are load-path dependent, so incremental shear forces are calculated from the increments in shear displacement, in terms of the shear stiffness of the joints. The maximum shear force is usually limited by a Mohr-Coulomb or similar strength criterion.

The motion of each block under the action of gravity, external loadings and the forces arising from contact with other blocks is determined from Newton's second law. A damping mechanism is also included in the model to account for dissipation of vibrational energy in the system.

The equations of motion may be integrated with respect to time using a central difference scheme to yield velocities and then integrated again to yield displacements. The velocity dependent damping terms have been omitted here for simplicity, but the same form of equations hold even when damping is included.

$$\dot{u}_i(t+\Delta t/2) = \dot{u}_i(t-\Delta t/2) + (\sum F_i(t)/m + g_i) \cdot \Delta t \qquad (1)$$

$$u_i(t+\Delta t) = u_i(t) + \dot{u}_i(t+\Delta t/2) \cdot \Delta t \qquad (2)$$

where $i = 1,2$ correspond to x and y directions respectively;
u_i are the components of displacement of the block centroid;
F_i are the components of non-gravitational forces acting on the block;
g_i are the components of gravitational acceleration;
m is the mass of the particular block.

The equation of rotation for each block can be integrated similarly. Note that, in the integrated equations, block velocities and displacements are expressed explicitly in term of values at a previous time and so may each be calculated independently.

The calculation cycle proceeds, with the calculated displacements being used to update the geometry of the system, and thence to determine new block interaction forces. These, in turn, are used in the next stage of the explicit integration of Newton's equations.

This explicit time integration scheme is only conditionally stable. Physically, the time step must be small enough that information cannot pass between neighbouring blocks in one step, thus justifying the independence of the integrated equations of motion. In practice, this implies that very many (often tens of thousands) solution cycles must be calculated. For example, some very large UDEC analyses of roof strata collapse and subsidence induced by underground coal mining [7], which included over 2000 fully deformable DEs, required about 200,000 solution cycles and about six days of Sun SPARCstation CPU time to come to final equilibrium.

The complex data structures which are used in DE programs to keep track of changing block contacts in an efficient manner, would need to be substantially rewritten to allow vectorisation on a conventional supercomputer. This has been done by several groups at different U.S. National Laboratories [8]. Another approach to developing DE programs which are computationally more efficient is to use parallel processing to integrate the equations of motion for many blocks simultaneously.

3. Machines and languages

3.1 Machines

The machines used in this study were the Encore Multimax multiprocessor and the IBM RS6000 Model 530 uniprocessor workstation. The latter was chosen as its CPU performance is likely to be representative of CPUs in future medium cost multiprocessors.

3.2 Languages

The languages used in the study were SISAL and FORTRAN.

SISAL [9][10] is an applicative language which has been targetted at a wide variety of systems including uniprocessors, current generation multiprocessors such as the Encore Multimax and research dataflow machines [11][12][13]. The textual form of SISAL, in terms of control structures and array representations, (Appendix) provides a relatively easy transition for those familiar with imperative languages and the optimising SISAL compiler (osc) from Colorado has yielded performance competitive with FORTRAN [14][15]. SISAL requires no directives or annotation at the source level.

The epf and xlf FORTRAN compilers were used for the Encore and RS6000 respectively. The epf compiler provides automatic analysis and annotation of the FORTRAN source with parallel and other directives. The intermediate annotated source is available for further explicit or manual annotation; alternatively the analysis and annotation stages may be bypassed permitting explicit annotation only.

Both the epf and current SISAL compilers exploit loop concurrency. Loops with no dependencies between cycles are 'sliced' into several loops each over some sub-interval of the original loop bounds. The number of slices is determined at runtime with the slices being executed concurrently.

4. Results

4.1 DECYL

DECYL is a simple program which is based on the same explicit integration algorithm used in DE programs for practical stress analysis of highly jointed rock. DECYL assumes that any of the circular DEs comprising the system being modelled may be in contact with any other DE; contact lists are not maintained and physical locality is not exploited. The diagramatic representation of a DECYL system is is shown in Figure 1. The general flow of computation in DECYL is shown in Figure 2.

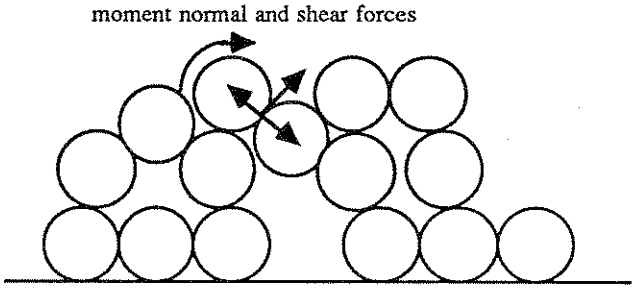

Figure 1. a DECYL system

```
while not stable do
    for j in 1 to no_DEs do
        for i in j+1 to no_DEs do
          if touching DE[i] and DE[j] then
             compute normal force
             if normal force > 0 then
                compute moment
                compute shear force
                if slipping then
                   adjust shear force
             resolve force components
             accumulate forces and moments acting on DE[j] and DE[i]
        integrate accelerations on DE[j]
        compute new position of DE[j]
```

Figure 2. DECYL computational flow

4.1.1 Automatic annotation

The DECYL program written in FORTRAN was translated to SISAL and run against FORTRAN on the Encore and IBM RS6000 systems. The speedups for a 1000 DE system are shown in Figure 3.

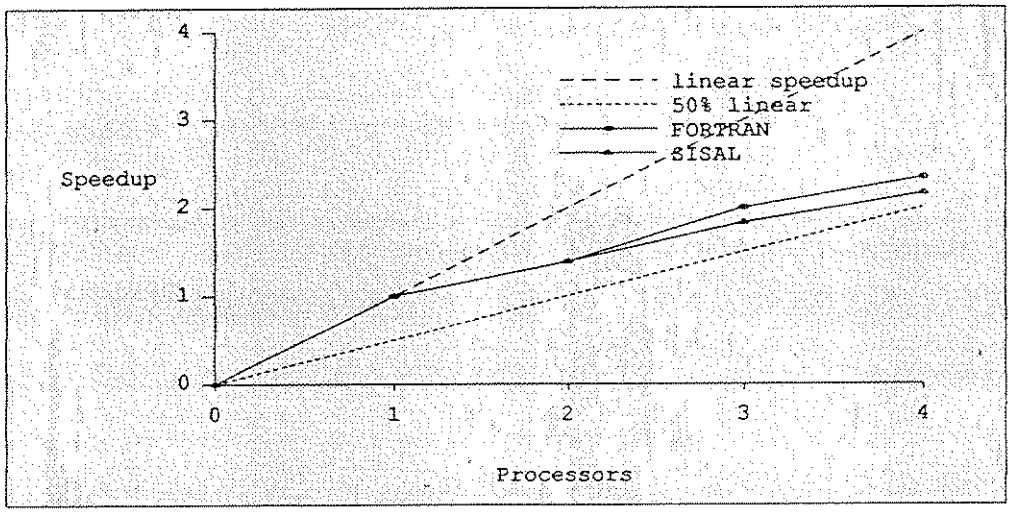

Figure 3. Speedup of DECYL in FORTRAN and SISAL

The speedup for FORTRAN is 2.34 and for SISAL 2.16. Greater than 50% machine utilisation is achieved in both cases.

4.1.2 Explicit Annotation

DECYL searches for contacting DEs on each iteration. This is done for each DE within an outer loop over all DEs. Inspection of the annotated FORTRAN revealed that the innermost loop was selected by epf to run in parallel rather than outer loop. In DECYL the outer loop has a sequential tail where the accelerations and displacements are computed and this in turn adversely effects speedup in that maximum speedup is the ratio of the time to execute the sequential section of the application to the time to execute the whole application on a single processor e.g. 5% sequential code implies a maximum speedup of 20.

To amortise the startup cost of loop slices it is best to maximise the work done for each slice. The strategy then is parallelise outer loops where possible; inner loops need only be parallelised if the bounds of the outer loop are such as to not provide sufficient slices to load the machine. In most scientific and engineering problems the loop ranges exceed the number of available processors by some factor.

The speedup for DECYL with the outer loop annotated explicitily is shown in Figure 4 and the collected time results for a single time step of DECYL for FORTRAN and SISAL on the Encore Multimax and IBM RS6000/530 are given in Table 1.

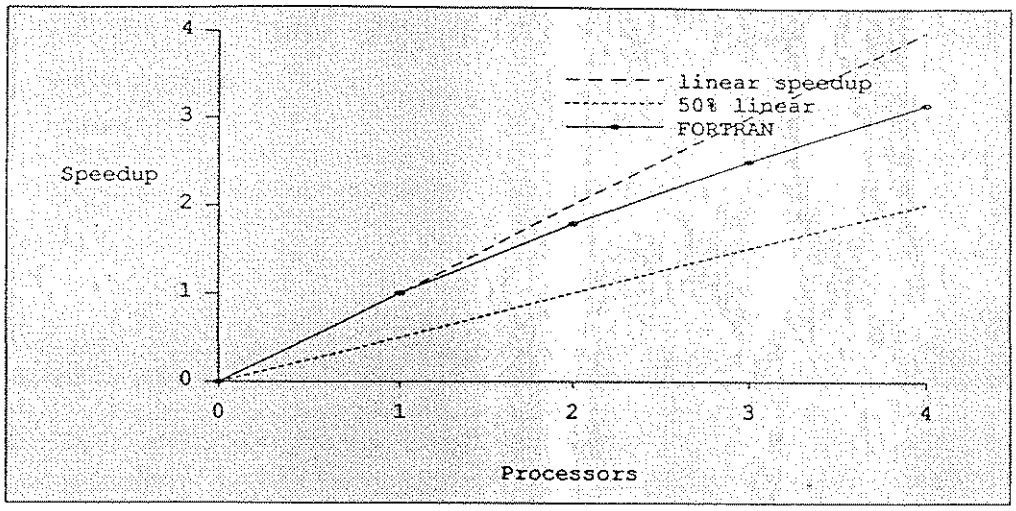

Figure 4. Speedup of DECYL in FORTRAN with explicit annotation

Language	1 processor	4 processors	(speedup)
FORTRAN (Encore epf implicit)	31.5+3.1	12.9+1.9	(2.3)
FORTRAN (Encore epf explicit)	28.7+3.3	8.2+1.9	(3.2)
SISAL (Encore osc)	42.2+3.2	20.0+1.1	(2.2)
FORTRAN (IBM xlf)	6.5+1.9		
SISAL (IBM osc)	9.0+1.2		

Table 1. Times (user+system) for DECYL (1000 DEs 1 time step)

Further improvement in SISAL performance is expected, reducing overheads due mainly to the array construction and access mechanisms of the current implementation of SISAL. SISAL permits structures to change size at runtime involving indirect access to matix elements via a vector of pointers to each matrix row. There are also consequential memory allocation overheads as structures are progressively constructed. The potential gains from static allocation of structures have been acknowledged by the developers of SISAL and will be seen in SISAL version 2.0 [15].

4.2 SDEM

SDEM's main computational cycle (Figure 5) consists of computing the new position of blocks given the current forces acting on them, and then from these positions, determine the new forces induced by blocks on their neighbours; these steps are repeated until the system stabilises. Contact lists, or lists of the other blocks a block is touching, are maintained to exploit locality in contrast to DECYL. If the displacement of any block is greater than some threshold then the contact lists of all blocks are updated; this deals with sudden events/collapses in the system occurring in a particular time step.

```
while not stable do

    if update required then
        update all contacts                    - currently sequential

    for all blocks do
        compute motion
        if major block displacement then
          record update required

    for all blocks do
        if block moved then
          for each corner do
              recompute bounding box

    for all blocks do
        compute stresses

    for all blocks do
        for each contacting block do
          compute forces
          lock block records
              accumulate inter block forces
          unlock block records
```

Figure 5. SDEM computational flow (explicit parallel version)

4.2.1 Automatic annotation

The Encore Multimax epf compiler was used to automatically annotate and compile the original FORTRAN source. The speedup obtained is shown in Figure 6.

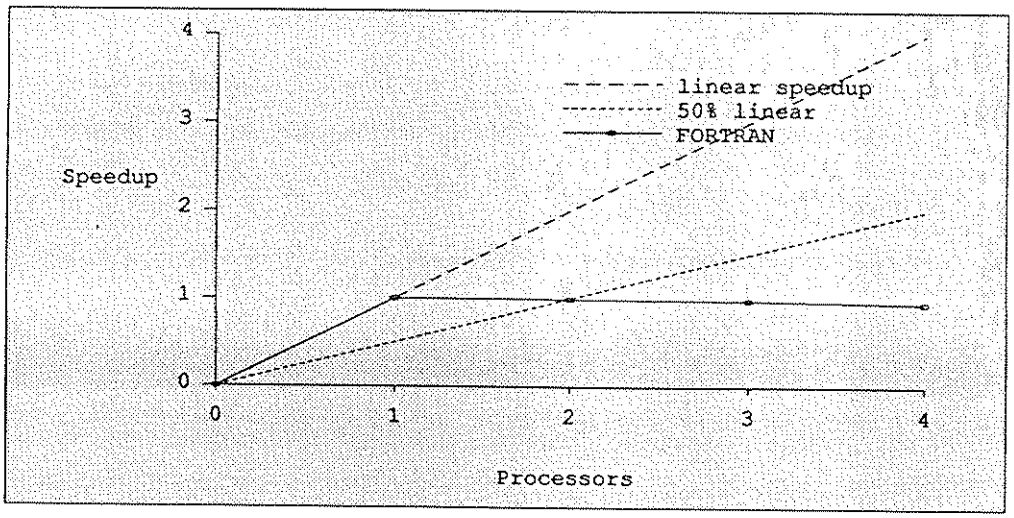

Figure 6. Speedup of SDEM with automatic annotation

It can be seen that not only was no speedup obtained, but there was actually some slow down. In the case of SDEM all major processes are called as subroutines e.g. computation of block motion. Inspection of the analysis listings produced by epf showed that any statement involving a subroutine call was deemed to be not concurrent i.e. epf does not perform inter-procedural analysis. This is reasonable if one assumes that FORTRAN routines may be linked as precompiled modules and that they may involve hidden manipulation of structures through COMMON. It does however mitigate against the use of subroutine calls in loops which are the primary source of concurrency!

4.2.2 Explicit annotation

It was decided to reject the expansion of subroutines inline as a solution, as this would result in a cumbersome difficult to maintain source. Further inspection showed that the loops over all blocks in the main iterative cycle and its subroutines could be made parallel subject to:

- the detection of a major block displacement for conditional updating of all block contacts;

- the 'locking' of blocks when accumulating new inter-block forces.

The resulting speedup for explicit annotation of SDEM is shown in Figure 7.

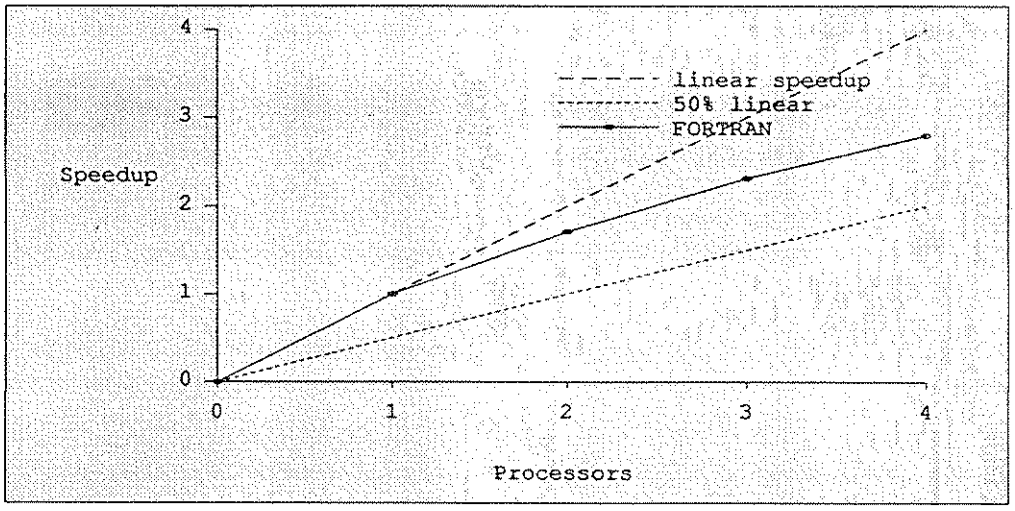

Figure 7. Speedup for explicit annotation (105 DE system for 20000 time steps)

The execution times for a number of machines for system of 105 blocks over 20000 time steps is shown in Table 3.

Machine	1 processor	4 processors
Encore Multimax	2735.6+8.9	982.1+4.2 (2.79)
IBM RS6000/530	145.0+1.4	

Table 2. Times (user+system) for SDEM (105 DEs 20000 time steps)

5. Conclusions

The analysis and initial parallelisation of SDEM was made difficult by the complicated data structures. The major data structure is contained in a single integer/real vector with several consecutive elements of the vector constituting a record describing a block. Some of the elements point in turn to other records of contact blocks and all fields are referred to numerically rather than symbolically. It is not suprising that the annotators and vectorisers have some difficulty extracting performance gains.

There remain a number of sequential regions of code in the main time step loop, and its associated subroutines, which are limiting the speedup obtained to date. Work is continuing on reducing their impact and it is expected there will be further small gains.

The study has shown that it is possible to obtain good speedup on current multiprocessors. These processors have relatively poor scientific performance but it is likely that next generation processors will be greatly improved in this respect.

Current FORTRAN annotators provide some promise of implicit parallel programming without resorting to explicit annotation although there appears to be some scope for improvement in the heuristics used by the epf annotator.

Acknowledgements

The authors thank Warwick Heath for his work on the original SISAL version of DECYL and Siong Tang for her assistance in the tuning of SDEM. The authors also thank the other members of the Laboratory for Concurrent Computing Systems, at the Swinburne Institute of Technology, for their contributions to the work presented in this paper.

References

1. Cundall, P.A., A computer model for simulating progressive large-scale movements in blocky rock systems. Proc. ISRM Symp. on Rock Fracture, Nancy, vol. 1, paper II-8. 1971.

2. Cundall, P.A. et al., Computer modeling of jointed rock masses. Technical Report N-78-4, U.S. Army Engineer Waterways Experiment Station,

Vicksburg, Miss., 1978.

3. Itasca. UDEC - Universal distinct element code, version ICG1.6; User's manual. Itasca Consulting Group, Inc., Minneapolis, 1990.

4. Itasca. 3DEC - 3-D distinct element code, version 1.2; User's manual. Itasca Consulting Group, Inc., Minneapolis, 1990.

5. Lorig, L.J., Private communication with M.A. Coulthard, 1985.

6. Lemos, J.V., A hybrid distinct element - boundary element computational model for the half-plane. M.S. Thesis, Dept. of Civil and Mineral Engng., University of Minnesota, Minneapolis, 1983.

7. Choi, S.K. and M.A. Coulthard, Mechanics of jointed rock masses using the distinct element method. Int. Conf. on Mechanics of Jointed and Faulted Rock, Vienna, 1990, (in press).

8. Taylor, L.M., BLOCKS: A block motion code for geomechanics studies, Sandia National Laboratories, Albuquerque, Report SAND-82-2373, 1983.

9. McGraw et al, SISAL: Streams and Iteration in a Single Assignment Language, Language Reference Manual, Lawrence Livermore National Laboratories, M146.

10. Skedzielewski S. and J. Glauert , IF1 An Intermediate Form for Applicative Languages, Lawrence Livermore National Laboratories, 1985.

11. Bohm A.P.W and J. Sargeant, Efficient Dataflow Code Generation for SISAL, Technical Report UMCS-85-10-2, Department of Computer Science, University of Manchester, 1985.

12. Webb N.J., Implementing an Applicative Language for the P.MIT/CSIRO Dataflow Machine, Department of Computer Science, Royal Melbourne Institute of Technology, *M.App.Sci. Thesis in preparation*, 1990.

13. Egan G.K., N.J. Webb and A.P.W. Bohm, Some Features of the CSIRAC II Dataflow Machine Architecture, in Advanced Topics in Data-Flow Computing, Prentice-Hall 1990, *in print.*

14. Cann D.C. and R.R. Oldehoeft, Compilation Techniques for High Performance Applicative Computation, Technical Report CS-89-108, Colorado State University, May 1989.

15. Cann D.C., High Performance Parallel Applicative Computation, Technical Report CS-89-104, Colorado State University, Feb.1989.

16. Chang P.S. and G.K. Egan, An Implementation of a Barotropic Numerical Weather Prediction Model in the Functional Language SISAL, Second ACM Sigplan Symposium on Parallel Programming (PPoPP), Seattle, March, 1990.

APPENDIX - Fragment of DECYL in SISAL

```
while (iter <= its) repeat  % for number of iterations

 fxsumj, fysumj, msumj, fn, m := for j in 1,ncyl
   alldu : vector := array_addh(old du,old du[j]);
   alldv : vector := array_addh(old dv,-r);
   allddu : vector := array_addh(old ddu,0.0);
   allddv : vector := array_addh(old ddv,0.0);
   allgamma : vector := array_addh(old gamma,(-old ddu[j])/r);

   fxsumji, fysumji, msumji, fnji, mji := for i in j+1,n
      dudif : real := alldu[j] - alldu[i];
      dvdif : real := alldv[j] - alldv[i];
      z : real := sqrt(dudif*dudif + dvdif*dvdif);
      % Interactions with other cylinders?
      fxsumji, fysumji, msumji, fnji, mji : real :=
       if (z > d | j = i) then
         0.0, 0.0, 0.0, 0.0, 0.0
       else
         let
         zdudif :real := dudif/z;
         zdvdif :real := dvdif/z;
         ddudif :real := allddu[i] - allddu[j];
         ddvdif :real := allddv[i] - allddv[j];
         % calculate the normal force between i and j
         dfn : real := akn * (ddvdif*zdvdif + ddudif*zdudif);
         testfn : real := (old fn[j,i]) + dfn;
         retfxsum, retfysum, retmsum, retfn, retm : real :=
          if (testfn < 0.0) then
            0.0, 0.0, 0.0, 0.0, 0.0
          else
            let
            % Calculate moment
            theta : real :=
              (ddvdif*zdudif - ddudif*zdvdif)/d;
            dm : real := -aks *(allgamma[j] + allgamma[i] - theta);
            mji : real := old m[j,i] + dm;
            % Calculate shear force
            testft : real := mji/r;
            % Is slip occurring
            ff : real := mu * testfn;
            abft : real := abs(testft);
            ft : real :=
              if (abft > ff) then ff * testft / abft
              else testft
              end if;
            retmji := r*ft;
            % Calculate force components (fx,fy)
            fx : real := (testfn*zdudif) - (ft*zdvdif);
            fy : real := (testfn*zdvdif) + (ft*zdudif);
            in
              fx, fy, retmji, testfn, retmji
            end let
          end if;
         in
          retfxsum, retfysum, retmsum, retfn, retm
```

```
          end let
        end if;
      returns
        value of sum fxsumji
        value of sum fysumji
        value of sum msumji
        array of fnji
        array of mji
      end for;
    returns
      array of fxsumji
      array of fysumji
      array of msumji
      array of fnji
      array of mji
    end for;
  % Now integrate accelerations to find displacements
  u,v,w,ddu,ddv,gamma,du,dv:=
  for j in 1, ncyl
  uj,vj,wj,dduj,ddvj,gammaj,duj,dvj:=
   if j=1 then
  ((fxsumj[j]/mass)*tdel) /(1.0+con1),  ((fysumj[j]/mass+g)*tdel) /(1.0+con1),
    ((msumj[j]/moi)*tdel) /(1.0+con1),
    0.0, 0.0, ((msumj[j]/moi)*tdel) /(1.0+con1)*tdel, 0.0,0.0
  else
    let
      tuj:real:=(old u[j]*(1.0-con1) + ((fxsumj[j]/mass)*tdel)) /(1.0+con1);
      tvj:real:=(old v[j]*(1.0-con1) + ((fysumj[j]/mass+g)*tdel)) /(1.0+con1);
      twj:real:=(old w[j]*(1.0-con1) + ((msumj[j]/moi)*tdel)) /(1.0+con1);
      tdduj:=tuj*tdel;
      tddvj:=tvj*tdel;
      tgammaj:=twj*tdel
    in
      tuj, tvj, twj,tdduj,tddvj, tgammaj,old du[j] + tdduj,old dv[j] + tddvj
    end let
  end if;
returns
array of uj
array of vj
array of wj
array of dduj
array of ddvj
array of gammaj
array of duj
array of dvj
end for;
    iter := old iter + 1;
  cylinders := for j in 1,ncyl
  returns
    array of record dudv_rec[du:du[j]; dv:dv[j]]
  end for;
returns
    array of cylinders when (mod(iter,50) = 0)
end for % number of iterations
```