Finite/Discrete Element Analysis of Multi-fracture and Multi-contact Phenomena

Invited Talk

D.R.J. Owen, Y.T. Feng, Jianguo Yu, and Djordje Perić

Department of Civil Engineering, University of Wales Swansea
Swansea, SA2 8PP, UK
{D.R.J.Owen, Y.Feng, J.Yu, D.Peric}@swansea.ac.uk

Abstract. A dynamic domain decomposition strategy is proposed for the effective parallel implementation of combined finite/discrete element approaches for problems involving multi-fracture and multi-contact phenomena. Attention is focused on the parallelised interaction detection between discrete objects. Two graph representation models, are proposed and a load imbalance detection and re-balancing scheme is also suggested. Finally, numerical examples are provided to illustrate the parallel performance achieved with the current implementation.

1 Introduction

The last several decades have witnessed the great success of the finite element method, along with a tremendous increase of computer power, as a numerical simulation approach for applications across almost every engineering discipline. However, for situations involving *discrete* or *discontinuous* phenomena, a combined finite/discrete element method naturally offers a more powerful solution capability. Typical examples that can considerably benefit from this combined solution strategy include process simulation (e.g. shot peening, granular flow, and particle dynamics) and fracture damage modelling (e.g. rock blasting, mining applications, and projectile impact).

Besides their discrete/discontinuous nature, these applications are often characterised by the following additional features: they are highly *dynamic*; with rapidly *changing domain configurations*; *sufficient resolution* is required; and *multi-physics phenomena* are involved. In the numerical solution context, *contact detection and interaction computations* often take more than half of the entire simulation time and the *small time step* imposed in the explicit integration procedure also gives rise to the requirement of a very large number (e.g. millions) of time increments to be performed. For problems exhibiting multi-fracturing phenomena, the necessity of frequent introduction of new physical cracks and/or adaptive re-meshing at both local and global levels adds another dimension of complexity. All these factors make the simulation of a realistic application to be extremely *computational intensive*.

J.M.L.M. Palma et al. (Eds.): VECPAR 2000, LNCS 1981, pp. 483–505, 2001.

Consequently, parallelisation becomes an obvious option for significantly increasing existing computational capabilities, along with the recently remarkable advances in hardware performance. In recent years considerable effort has been devoted to the effective parallel implementation of conventional finite element procedures, mainly based on a *static domain decomposition* concept. The features itemised above associated with the applications of interest make such a parallelisation much more difficult and challenging. Only very recently, have some successful attempts emerged at tackling problems of a similar nature[1,2,3].

It is evident that a *dynamic domain decomposition* (DDD) strategy plays an essential role in the success of any effective parallel implementation for the current problem. The ultimate goal of this work is therefore to discuss the major algorithmic aspects of dynamic domain decomposition that make significant contributions to enhancing the parallel performance. Our implementation is also intended to be general for both shared and distributed memory parallel platforms.

The outline of the current work is as follows: In the next section, a general solution procedure of the combined finite/discrete element approach for problems involving material discontinuity and failure is reviewed and a more detailed description is devoted to the multi-fracture modelling and the interaction detection; Section 3 presents dynamic domain decomposition parallel strategies for the combined finite/discrete element approach and parallel implementation for both the finite element computation and the global search is also investigated. Issues relevant to the dynamic domain decomposition algorithm for the contact interaction computation, mainly the two graph representation models for discrete objects, is proposed in Section 4. Section 5 is devoted to the description of a load imbalance detection and re-balancing scheme. Finally, numerical examples are provided to illustrate the parallel performance achieved with the current implementation.

2 Solution Procedures for Combined Finite/Discrete Element Approaches

Engineering applications involving material separation and progressive failure can be found, amongst others, in masonry or concrete structural failure, demolition, rock blasting in open and underground mining and fracture of ceramic or glass-like materials under high velocity impact. The numerical simulation of such applications, especially in large scale, has proved to be very challenging. The problems are usually represented by a small number of discrete bodies prior to the deformation process. In the combined finite/discrete element context, the deformation of each individual body is modelled by the finite element discretisation and the inter-body interaction is simulated by the contact conditions. During the simulation process, the bodies are damaged, by, for example, tensile failure, and modelling of the resultant fragmentation may result in possibly two or three orders of magnitude more bodies by the end of the simulation. In addition, the substantial deformation of the bodies may necessitate frequent adaptive

remeshing of the finite element discretisation. Therefore the configuration and number of elements of the finite element mesh and the number of bodies are continuously changing throughout the simulation.

Similarly, process engineering applications often contain a large number of discrete bodies. In many cases, these bodies can be treated as *rigid* and represented by *discrete elements* as simple geometric entities such as disks, spheres or ellipses. Discrete elements are based on the concept that individual material elements are considered to be separate and are (possibly) connected only along their boundaries by appropriate physically based interaction laws. The response of discrete elements depends on the interaction forces which can be short-ranged, such as mechanical contact, and/or medium-ranged, such as attraction forces in liquid bridges.

Contact cases considered in the present work include node to edge/facet and edge/facet to edge/facet. Short-range mechanical contact also include disk/sphere to disk/sphere and disk/sphere to edge/facet. Medium-range interactions can be represented by appropriate attractive relations between disk/sphere and disk/sphere entities.

The governing dynamic equations of the system are solved by explicit time integration schemes, notably the central difference algorithm.

2.1 Procedure Description

In the context of the explicit integration scheme, a combined finite and discrete element approach typically performs the following computations at each time step:

1. Finite element and fracture handling:
   - Computation of internal forces of the mesh;
   - Evaluation of material failure criterion;
   - Creation of new cracks if any;
   - Global adaptive re-meshing if necessary;
2. Contact/interaction detection;
   - Spatial search: detection of potential contact/interaction pairs among discrete objects;
   - Interaction resolution: determination of actual interaction pairs through local resolution of the kinematic relationship between (potential) interaction pairs;
   - Interaction forces: computation of interaction forces between actual interaction pairs by using appropriate interaction laws.
3. Global solution: computation of velocities and displacements for all nodes;
4. Configuration update: update of coordinates of all finite element nodes and positions of all discrete objects;

The procedures of finite element internal force computation, equation solution and configuration update in the above approach are all standard operations, and therefore further description is not necessary. However, the fracture modelling and the interaction detection warrant further discussion.

2.2 Multi-fracturing Modeling

Two key issues need to be addressed for successful modelling of material failure: (i) the development of constitutive models which reflect the failure mechanism; (ii) the ability of numerical approaches to handle the discontinuities such as shear bands and cracks generated during the material failure and fracture process.

Failure Models A variety of constitutive models for material failure have appeared over the years, with softening plasticity and damage theory being the two most commonly adopted in the nonlinear finite element analysis of failure. For brittle materials, a simple Rankine failure model can be employed. After initial yield, a rotating crack formulation may be employed in which the anisotropic damage is modelled by degrading the elastic modulus in the direction of the current principal stress invariant.

For all fracture or localisation models regularisation techniques must be introduced to render the mesh discretisation objective. Optional formulations include non-local damage models, Cosserat continuum approaches, gradient constitutive models, viscous regularisation and fracture energy releasing/strain softening approaches. All models effectively result in the introduction of a length scale and have specific advantages depending on the model of fracture and loading rate.

More detailed description of various fracture models can be found in [4].

Topological Update Scheme The critical issue of fracture modelling is how to convert the continuous finite element mesh to one with discontinuous cracks and to deal with the subsequent interactions between the crack surfaces. The most general approaches permit fracture in an arbitrary direction within an element and rely on local adaptive re-meshing to generate a well-shaped element distribution.

A particular fracture algorithm is developed in this work to model the failure of brittle materials subject to impact and ballistic loading. The fracture algorithm inserts physical fractures or cracks into a finite element mesh such that the initial continuum is gradually degraded into discrete bodies. However, the primary motivation for utilising the algorithm is to correctly model post-failure interaction of fractures and the motion of the smaller particles created during the failure process.

Within this algorithm, a nodal fracture scheme is employed to transfer the virtual smeared crack into a physical crack in a finite element mesh. The scheme is a three stage procedure: (i) Creation of a failure map for the whole domain; (ii) Assessment of the failure map to identify where fractures should be inserted; (iii) Updating of the mesh, topology and associated data.

In 2-D cases, the failure direction is defined to coincide with the maximum failure strain direction and the crack will propagate orthogonal to the failure direction. Associated with the failure direction, a failure plane is defined with the failure direction as its normal and the failed nodal point lying on the plane. A crack is then inserted through the failure plane. If a crack is inserted exactly

through the failure plane, some ill-shaped elements may be generated. Local remeshing is then needed to eliminate them. Alternatively, the crack is allowed to propagate through the most closely aligned element boundary. In this way, no new elements are created and the updating procedure is simplified. However, this procedure necessitates a very fine mesh discretisation around the potential fracture area. Within the current algorithm a minimum element area criterion is used to ensure that excessively small sliver elements are not created. If the element area obtained by splitting the elements is below this threshold value then the fracture is forced to be along element boundaries. For 3-D situations, the corresponding fracture algorithm is basically the same but the implementation procedure is more complicated.

In addition, an element erosion procedure is also applied to deal with the situation that the material represented by the element to be eroded no longer contributes to the physical response for the problem, such as the case where the material is melted down or evaporated at high temperature or is transformed into very small particles.

2.3 Interaction Detection

The interaction contact detection comprises three phases: (global) spatial search, (local) interaction resolution and interaction force computation.

The spatial search algorithm employed is a combination of the standard space-cell sub-division scheme with a particular tree storage structure - the *Augmented Digital Tree* (ADT) [5] - and can accommodate various geometrical entities including points, facets, disks/spheres and ellipses. Each entity is represented by a bounding box extended with a buffer zone. The size of the buffer zone is a user-specified parameter. In the case of medium-range interaction, it must not be smaller than the influence distance of each object considered. The algorithm eventually locates for each object (termed the *contactor*) a list of neighbouring objects (termed *potential targets*) that may potentially interact with the contactor.

In the second phase, each potential contactor-target pair is locally resolved on the basis of their kinematic relationship, and any pair for which interaction does not occur, including medium range interaction if applied, is excluded. In the final phase, the interaction forces between each actual interaction pair are determined according to a constitutive relationship or interaction law.

Effects of Buffer Zone Sizes The global spatial search may not necessarily be performed at every time step, as will be described below, while the interaction resolution and interaction force computations should be conducted at each time step. Furthermore, some kinematic variables computed in the interaction resolution phase will be used in the force computation. For these reasons, the interaction resolution and force computations are actually performed together in the current implementation.

The computational cost involved in the interaction force computation phase is fixed at each time step, if no new surfaces are created during the fracturing

process. It may, however, vary significantly for the other two phases if different sizes of buffer zone are specified.

Basically, the size of buffer zone has conflicting effects on the overall costs of the global search and the local interaction resolution. First of all, after performing a global search at one time step, a new search is required to be performed only if the following condition is satisfied

$$l_{max} > l_{buff} \tag{1}$$

where l_{buff} is the size of the buffer zone; and l_{max} is the maximum accumulated displacement of a single object for all discrete bodies in any axial direction after the previous search step:

$$l_{max} = \max_{i,j} \Big(\sum_{k=1} v_i^j \Delta t_k \Big) \qquad i \in [1, n_{body}], j \in [1, n_{dim}] \tag{2}$$

in which v_i^j is the velocity of object i in the j direction; Δt_k the length of the time step at the k-th increment after the global search; n_{body} the total number of objects and n_{dim} the number of space dimensions.

Given a larger buffer zone, the spatial search will create a longer list of potential targets for each contactor, which will increase the amount of work in the phase of interaction resolution in order to filter out those pairs not in potential interaction. On the other hand, the global search will be performed with less frequency which leads to a reduced overall cost in the spatial search phase. With a smaller buffer zone, the potential target list is shorter and the local interaction resolution becomes less expensive at each step, but the global search should be conducted more frequently thus increasing the computational cost in searching.

A carefully selected buffer zone can balance the cost in the two phases to achieve a better overall cost in interaction detection. Nevertheless, an optimal buffer zone size is normally difficult to select for each particular application.

Incremental Global Search Generally speaking, the spatial search is an expensive task that often consumes a substantial portion of the total simulation time if each new search is performed independently from the previous search. The cost could however be reduced to some extent if the subsequent searches after the initial one are conducted in an *incremental* manner. In this incremental approach, the tree structure (ADT) required in the current search can be obtained as a modification of the previous structure and some search operations can also be avoided.

This approach is based on the observations that even though the configuration may undergo significant changes during the whole course of the simulation, the actual change between two consecutive search steps is bounded by the buffer zone and therefore is local. In addition, the space bisection tree is characterised by the fact that each node in the tree represents a subspace of the whole simulation domain. As long as each node (i.e. object) itself stays in the subspace

it is associated with, the whole tree will not need to be modified at all. Consequently, in the new search, it is possible to avoid building an entirely new tree by modifying only the subtrees that are affected by those objects which have moved out of their represented subspace.

The detail of the above (incremental) global search algorithm can be found in [5]. A more detailed description for interaction detection can be found in [4], while the interaction laws applied to particle dynamics, and shot peening in particular have been discussed in [6,7].

3 Parallel Implementation Strategies

Domain decomposition based parallelisation has been established as one of the most efficient high level (coarse grain) approaches for scientific and engineering computations, and also offers a generic solution for both shared and distributed parallel computers.

A highly efficient parallel implementation requires two, often competing, objectives to be achieved: a well balanced workload among the processors, and a low level of interprocessor communication overhead.

For conventional finite element computations with infrequent adaptive remeshing and without contact phenomena, a *static* domain decomposition is generally an ideal solution that initially distributes the sub-domain data to each processor and redistributes them after each adaptive remeshing. A well load-balanced situation can be achieved if each processor is assigned an equal number of elements. Interprocessor communications can be reduced to a low level if the interface nodes that are shared by more than one sub-domain are small.

For the current situation involving both finite and discrete elements, to apply a single static domain decomposition for both finite and discrete element domains will apparently not achieve a good parallel performance due to the highly dynamic evolution of the configuration concerned. Therefore a *dynamic* domain decomposition strategy should be adopted.

3.1 Dynamic Domain Decomposition

The primary goal of the dynamic domain decomposition is to dynamically assign a number of discrete elements or objects to each processor to ensure good load balance as the configuration evolves.

Besides the same two objectives as for a static domain decomposition, the dynamic domain decomposition should also achieve an additional two objectives: *minimum data movement* and *efficiency*.

Firstly, completely independent domain decompositions between the consecutive steps normally give rise to very large amount of data movement involved among the processors, leading to a substantial communication overhead. Therefore, the dynamic domain decomposition should provide efficient re-partitioning algorithms that can keep the domain partitioning constant as much as possible during the simulation. Secondly, since the partitioning may need to be performed

many thousands of times during simulations, the dynamic domain decomposition must be very efficient.

Most dynamic domain decomposers can be classified into two general categories: *geometric* and *topological*. Geometric partitioners divide the computational domain by exploiting the location of the objects in the simulation, while topological decomposers deal with the connectivities of interactions instead of geometric positions. Both methods can be applied to both finite element and discrete element simulations. Generally, topological methods can produce better partitions than geometric methods, but are more expensive. A particular topological based decomposer called (Par)METIS [8] is chosen as the domain partitioner in our implementation as it appears to meet the above criterion for an effective domain decomposition algorithm.

When applying the dynamic domain decomposition strategy to the current finite/discrete element simulation, there are two different approaches. The first approach dynamically decomposes both finite and discrete element domains. Different characteristics of the two parts makes it very difficult to achieve a good load balance.

An alternative solution, which is employed in this work, is to decompose the computations associated with the two domains separately. Within the interaction detection of the discrete elements, slightly modified schemes are employed for the global search, and the combined interaction resolution and force computations. This strategy provides maximum flexibility in the implementation and thus allows the most suitable methodologies and data structures to be applied in different stages of the simulation. This may however cause extra communications in information exchange among processes due to the data structure inconsistencies between different solution stages at each time step.

Dynamic decomposition of finite element computations and a particular parallel implementation version of the global search are discussed below, while the parallel issues for the combined interaction resolution and force computation, including dynamic domain partitioning and dynamic load re-balancing, will be addressed respectively in the next two sections.

3.2 Dynamic Decomposition of Finite Element Computations

Compared to the discrete element domain, the configuration change of the finite element mesh is relatively less frequent and insignificant. The major concern for a dynamic domain decomposition algorithm is its ability to adaptively partition the domain so as to minimize the cost due to the data migration among different processors after a new partitioning.

To achieve a well-balanced workload situation often needs a fine tuning. In the case that different elements have different orders (e.g. linear or quadratic), and/or different material models, and/or different stress states (e.g. elastic or plastic), thus having different computational effort, each element should be weighted proportional to its actual cost to ensure a good load balance. In heterogeneous computer clusters, an uneven workload decomposition should be ac-

Box 1: Parallel Implementation of Initial Global Search

1. Globally distribute objects among processors using any available means of domain decomposition;
2. Each processor defines the bounding box of each object with an extended buffer zone in its subdomain;
3. Each processor computes the global bounding box of its domain and broadcasts this information to other processors;
4. Each processor applies the sequential global search algorithm to its domain and constructs its own ADT and potential target lists;
5. Each processor identifies in its domain the objects that overlap with the global bounding boxes of the other domains and then sends the objects to the processors owning the overlapping domains;
6. Each processor receives from other processors the objects that overlap with its own global bounding box; conducts the search for each object in its ADT to build the potential target list, if any; and sends the result back to the processor to which the object originally belongs;
7. Each processor collects the lists created in other processors and merges them to the original lists to obtain the final lists.

complished to take into account different computing powers that each member machine of the group can offer.

3.3 Dynamic Parallelisation of Global Search

Global search is found to be the most difficult operation to be parallelised efficiently due to its global, irregular and dynamic characteristics. In our parallel implementation, some domain decomposition strategies are also employed to both initial and subsequent incremental search steps. Box 1 outlines the algorithmic steps involved in the initial global search.

As the initial search is conducted only once, the performance of the algorithm is not a major concern. Therefore, it can employ any available means to distribute the objects among the processors. In some cases, the algorithm can even be performed sequentially.

For the subsequent incremental searches, an effective parallel implementation of the algorithm becomes critical. Box 2 presents the corresponding parallel algorithm [9] and only the differences from the previous initial search approach are listed.

In the algorithm, it is essential to assume that a good dynamic object distribution is performed in order to achieve a good load balance, and one such distribution scheme will be proposed in the next section.

Box 2: Parallel Implementation of Incremental Global Search

1. Each processor migrates those objects that are assigned to different domains in the current partition to their new processors;
2. Each processor defines/modifies the bounding box of each object with an extended buffer zone in its subdomain;
3. Each processor updates the global bounding box of its domain and broadcasts this information to other processors;
4. Each processor modifies its own ADT and constructs its potential target lists;

5-7. Same as the Implementation in Box 1.

4 Dynamic Domain Decomposition for Interaction Computations

Many scientific and engineering applications can be abstractly expressed as weighted graphs. The vertices in the graph represent computational tasks and the edges represent data exchange. Depending on the amount of computation performed by each task, the vertices are assigned a proportional weight. Similarly, the edges are assigned weights that reflect the data needed to be exchanged.

A domain partitioning algorithm aims to assign each processor a subset of vertices whose total weight is as the same as possible so as to balance the work among the processors. At the same time, the algorithm minimises the edge-cut (subject to load-balance requirements) to minimise the communication overhead.

As the simulation evolves, the computational work associated with an object can vary, so the objects may need to be redistributed among the processors to balance the workload. The objective of dynamic re-partitioning is therefore to compute a balanced partitioning more effectively that minimises the edge-cut, and to minimise the amount of data movement required in the new partitioning.

4.1 *A Priori* Graph Representation of Discrete Objects

The first important step for the development of an efficient dynamic partitioning for the interaction computation lies in an appropriate graph representation of discrete elements. Unlike a finite element mesh, discrete objects do not have underlying connectivity to explicitly associate with, and thus some form of connections among the objects should be established. An obvious choice is to use the potential target list of each contactor as its connectivity, upon which a graph representation of the interaction computation can therefore be established.

Each vertex is initially assigned a weight that is equal to the number of the targets in its potential list and no weight is placed on the edges. This weighting scheme will work reasonably well if the simulation is dominated by one type

of interaction and the objects have nearly even distribution across the whole domain.

As the above graph model is established based on *a priori* estimation of the interaction relationship and the cost of interaction force computation, it is therefore termed the *a priori* model.

This special choice of connectivity has the following implications: First, as the potential target list will not be changed in two consecutive global searches, the graph partitioning may need to be performed only once after each global search rather than at each time step. Second, depending on the size of buffer zones specified, the list is only an approximation to the actual interaction relationship between the contactor and target. Furthermore, the actual relationship between objects is also undergoing dynamic changes at each time step. These considerations indicate that the resulting graph may not be a perfect representation of the problem under consideration. Consequently some degree of load imbalance may be expected.

4.2 *A Posteriori* Draph Representation

In order to improve the accuracy of the above graph model, the following inherent features in the current solution procedure for the interaction resolution and force computation should be addressed:

- The target lists created by the global search provide only a potential interaction relationship between contactors and targets, and this relation can only be established after the local resolution, and therefore can not be accurately established *a priori*;
- The computation costs associated with each object, including the kinematic resolution and interaction force computation, are also unknown *a priori*, and are also very difficult to be measured precisely;
- Both the interaction relationship and the computational cost for each object are undergoing constant changes at each time step.

By specifying a very small buffer zone, the potential interaction lists can be much closer to the actual interaction relationship, but this will significantly increase the costs of simulation as indicated earlier, and therefore is not a suitable option.

In view of these facts, an alternative graph representation model is suggested. The basic idea is to use the actual information obtained in the previous time step as the base for modelling the situation at the current step. More specifically, a (nearly) accurate graph model is built for the previous time step. The graph is based on the actual contactor-target list as the connectivity and the (nearly) actual computational cost for each contactor as the vertex weighting. Since the computation domain will not undergo a significant change in two consecutive time steps as a result of a small time increment, it is reasonable to assume that this graph model is also a good representation of the problem at the current time step. Due to the *a posteriori* nature, this model is thus termed the *a posteriori graph representation.*

Another advantage of this *a posteriori* model is that the global search and the domain partitioning can now be performed at different time instances and intervals. This means that the global search is performed when the potential interaction list will be no longer valid; while a dynamic graph partitioning can be conducted when load balancing is required to be maintained. The latter aspect will be further expanded upon in the next section.

4.3 Integration with Global Search

In principle, both global search and graph partitioning can have totally independent data structures, mainly with different object distribution patterns. In distributed memory platforms, if the same object is assigned to different domains/processors in the search and partitioning phases, some data movement between two processors becomes inevitable. The extra communication results from the data structure inconsistency between the two phases. If the two data structures can be integrated to a certain degree, the communication overhead can be reduced.

In the parallel implementation, the data associated with a particular object, such as coordinates, velocities, material properties, forces and history-dependent variables, is more likely to reside on the processor determined by the graph partitioning. Therefore, it is a good option for the global search to use the same object distribution pattern as generated in the graph partitioning phase, i.e. a single dynamic domain decomposition is applied to all computation phases in the interaction detection step.

Another advantage of this strategy is that a good load balance may also be achieved in the global search. This can be justified by the fact that a similar total number of potential interaction lists among the processors also implies a similar number of search operations. In addition, the incremental nature of the dynamic repartitioning employed ensures that only a small number of objects is migrated from one domain to another, and hence only a limited amount of information is required to be exchanged among the processors when re-constructing the ADT subtrees and potential target lists in the incremental global search.

5 Dynamic Load Re-balancing

The domain partitioning algorithm is supposed to generate a well load-balanced partitioning. For various reasons, this goal may, however, be far from easy to achieve, especially when dynamic domain decomposition is involved. It is essential, therefore, to have a mechanism in the implementation that can detect the problem of load imbalance when it occurs and take proper actions to restore, or partially restore, the balance if necessary. As a matter of fact, dynamic load re-balancing is an integrated part of the dynamic domain decomposition which determines the time instances at which a dynamic re-partition should be performed. In this section, a dynamic load balancing scheme will be proposed in an attempt to enhance the performance of the parallelised interaction resolution and force computations.

5.1 Sources of Load Imbalance

Load imbalance may be caused by the following factors:

- A perfect partition may not be able to achieve a perfect CPU time balance due to sophisticated hardware and software issues;
- The partition produced by the domain decomposer is not perfectly balanced in terms of workload;
- In the finite element computation phases, imbalanced workload may arise when mechanical properties of some elements change, for instance, from elastic to plastic; or when local re-meshing happens that leads to some extra or fewer number of elements for certain domains;
- In the interaction resolution and force computation phases, workload unbalancing may happen for two reasons: The connectivity for each object used in the graph is only an approximation to the real interaction relationship as addressed earlier; and/or the assigned weighting for each object/edge may not represent the real cost associated with the object/edge;

The first two sources of load imbalance are beyond the scope of the present work and thus will not be given further consideration.

An important part of the load rebalancing scheme is to obtain a fairly accurate measurement of the actual cost for each object. This is however not easy to fulfill. The difficulty is due to the fact that the relative difference in computation costs between different types of interaction, such as sphere to sphere and sphere to facet, and sphere to sphere and node to facet, are difficult to define accurately.

A possible solution to this difficulty is by means of numerical experiment. We can design a series of experiments on one particular system to establish a relative cost of each element operation, including each type of interaction resolution and force computation. This relative cost, however, should be updated accordingly if the program is to be run on a new system.

With the relative cost model, we can compute the number of different computation operations an element/object participates in at each time step and then calculate its actual cost upon which the assignment of a proportional weighting to the element/object in the graph is based. This provides a basis on which further operations aimed at maintaining load balancing can take place.

5.2 Imbalance Detection and Re-balancing

Most load re-balancing schemes consist of three steps: imbalance detection, rebalancing criterion and domain re-decomposition.

The first step of load re-balancing schemes is to constantly monitor the level of load imbalance among the processors during the simulation. The workload of one processor at each time step can be measured, for instance, by summing up the relative costs of all objects in the processor. These costs are already computed during the interaction computation by following the procedure outlined earlier. Alternatively, the workload can also be accurately obtained by measuring its

actual runtime, but this is not trivial. Using either approach, the level of load imbalance can be defined as

$$B = \frac{W_{max} - \overline{W}}{\overline{W}} \tag{3}$$

where

$$W_{max} = \max_i W_i, \quad \overline{W} = \Big(\sum_{i=1}^{n_p} W_i \Big) / n_p$$

and W_i is the workload of the i-th processor; and n_p is the number of processors.

The next step is to decide when to perform a new graph partitioning to re-balance the load. Such a decision requires consideration of complicated tradeoffs between the cost of the graph partitioning, the quality of the new partitioning and the redistribution cost. A simple approach is to start the re-partitioning when the level of imbalance, B, exceeds a prescribed value, τ, i.e.

$$B > \tau \tag{4}$$

Similar to the size of buffer zone, τ may also need to be carefully tuned in order to achieve a better overall performance.

The final step is to perform a domain re-partitioning as described in the previous sections when condition (4) is satisfied.

6 Numerical Experiments

In this section, three examples are provided to illustrate the performance of the algorithms and implementation suggested. As the parallel performance of a domain decomposition for conventional finite elements is well established, the experiments will focus on the interaction resolution and force computations in both 2D and 3D cases.

In addition, in view of the fact that the efficiency of a parallelised program is often affected to some extent by complex hardware and software issues, the contribution to the overall performance from only the algorithmic perspective is therefore identified. More specifically, the following issues will be investigated: 1) the cost of the dynamic domain partitioning and repartitioning in terms of CPU time, and the quality of the partitioning; 2) the behaviour of two proposed graph representation models in terms of load balancing.

The parallelised finite/discrete element analysis program is tested on an SGI Origin 2000 with 8 processors. Due to the shared memory feature of the machine, interprocessor communication overhead plays a much less active role in the parallel performance. Each example is respectively tested with 1, 2, 4 and 6 processors.

The type of interaction considered between discrete elements is standard mechanical contact and the contact cases include node to edge, disk to disk and disk to edge contact in 2D, and sphere to sphere and sphere to 3-noded facet contact in 3D.

6.1 Example 1: 2D Dragline Bucket Filling

This example simulates a three-stage dragline bucket filling process, where the rock is modelled by discrete elements as disks, while the bucket is modelled as rigid, and the filling is simulated by dragging the bucket with a prescribed motion.

Figs. 1 and 2 respectively illustrate the initial configuration of the problem and two subsequent stages of the simulation. The discrete element model contains over 5000 disks with a radius of 20 units. The size of buffer zone is chosen to be 1 unit, and the time step is set to be 2.6×10^{-6} sec that leads to a total number of 1.6 million time increments required to complete the simulation. With the current buffer zone size, the global (re-)search is performed at about every $30 \sim 50$ steps and takes about 12.9% of the total CPU time in the sequential case.

For the dynamic domain partitioner employed, the CPU time consumed is 0.7% of the total time in the sequential case and 1.2% with 6 processors. These indicate that the re-partitioning algorithm in ParMETIS is efficient. It is also found that the partitioner produces partitioning with up to 3.5% load imbalance in terms of the weighted sub-total during the simulation.

Fig. 3 demonstrates the necessity of employing a dynamic re-partitioning scheme, in which, the sub-domain distributions obtained by the re-partitioning at two stages are compared with those produced by a completely new partitioning at each occasion. It confirms that a series of consistent partitions that minimise the redistribution cost will not be achieved unless a proper re-partitioning algorithm is adopted in the dynamic domain decomposition. Note that, for better viewing, a much larger disk radius is used in Fig. 3.

The CPU time of each processor for the contact resolution and force computations using the *a priori* graph model at the first 500k time increments is shown in Fig. 4a with various number of processors, where it is clearly illustrated that severe load imbalance occurs. This can be explained, for instance in the 2-processor case, by the fact that although the domain is well partitioned according to the potential contact lists, the actual computation cost on the second sub-domain is much less because the disks in this region are more scattered i.e. more false contact pairs are included in the corresponding lists.

A much better load balancing situation, depicted in Fig. 4b, has been achieved by the *a posteriori* graph model together with the proposed load re-balancing scheme. Table 1 also presents the overall speedup obtained by these two models with different number of processors.

Table 1. Speedup obtained by two graph models for Example 1

Model	2 processors	4 processors	6 processors
a priori model	1.63	3.20	4.41
a posteriori model	1.86	3.55	5.01

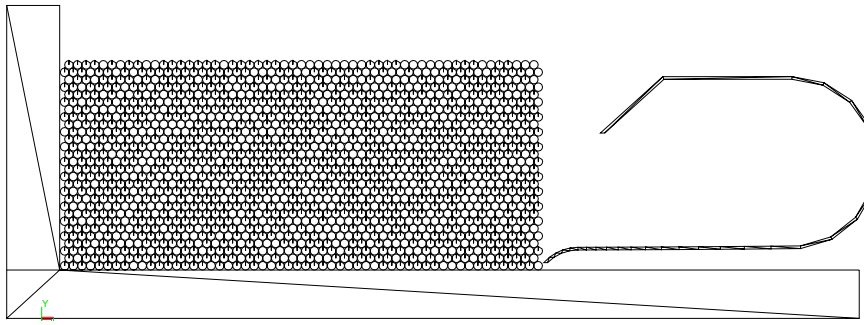
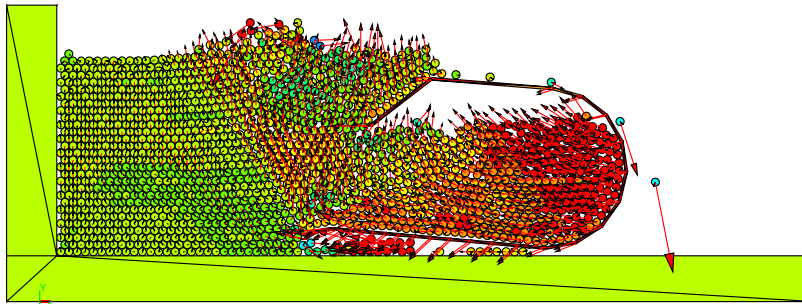
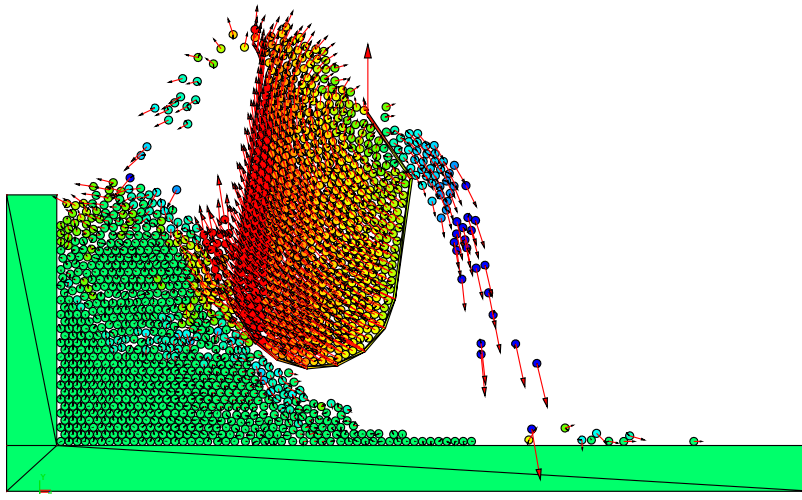


Fig. 1. Example 1 - Dragline bucket filling: Initial configuration



(a)



(b)

Fig. 2. Example 1 - Dragline bucket filling: Configurations at two stages showing particle velocities: (a) t=2s; (b) t=3.3s

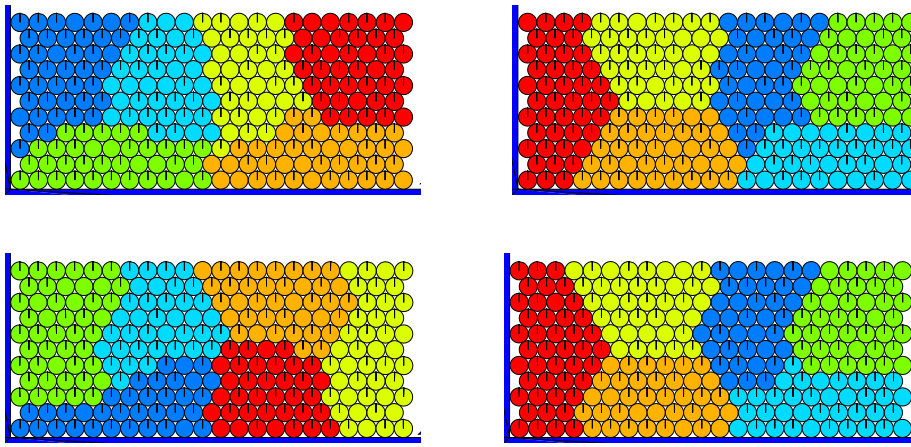
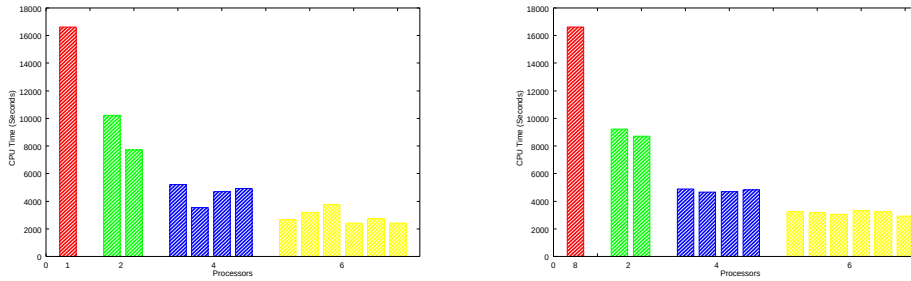


Fig. 3. Example 1 - Dragline bucket filling: Domain partitions at two different time instants: complete partitioning (left column) and re-partitioning (right column)



(a) (b)

Fig. 4. Example 1 - Dragline bucket filling: CPU time of each processor: (a) the *a priori* model; (b) the *a posteriori* model

Table 2. Speedup obtained by two graph models for Example 2

Model	2 processors	4 processors	6 processors
a priori model	1.86	3.55	4.72
a posteriori model	1.88	3.65	5.23

6.2 Example 2: 3D Hopper Filling

The second example performs simulations of a 3D ore hopper filling process. The ore particles are represented by discrete elements as spheres and the hopper and the wall are assumed to be rigid. The particles are initially regularly packed at the top of the hopper and then are allowed to fall under the action of gravity. The configuration of the problem at an intermediate stage of the simulation is illustrated in Fig. 5.

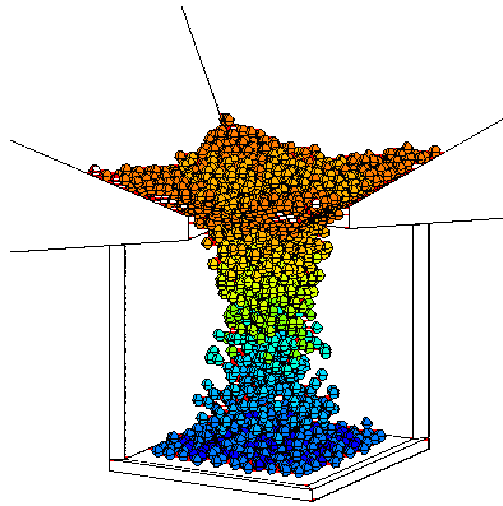


Fig. 5. Example 2 - 3D Hopper filling: Configuration at t=0.75s

The radius of the spheres is 0.25 units and the buffer zone is set to be 0.0075 units. The global search is conducted less frequently at the beginning and end of the simulation due to a relatively small velocity of motion.

A similar quality of the partitioning as in the previous example is observed in this example. Table 2 shows the speedup achieved by both graph models with various number of processors. It appears that the *a priori* graph model exhibits a similar performance as the *a posteriori* model for the cases of 2 and 4 processors, but shows a performance degradation in the 6-processor case. The reason for this is because of the symmetry in both x- and y-directions in the

problem, the domain decomposer produces a well balanced partitioning, in terms of actual computation cost, in the 2 and 4 processor cases, but fails to achieve the same quality of partitioning in the case of 6 processors. Also note that since the computational cost associated with each contact pair in 3D is more expensive than that in 2D, a slightly better overall parallel performance is achieved in this example.

6.3 Example 3: Axisymmetric Layered Ceramic Target Impact

This example consists of a tungsten long rod impacting a composite target comprising an RHA backing block, three ceramic tiles of approximate thickness 25mm, with 6mm RHA cover plate. The ceramic tiles are unconfined. The computational model is axisymmetric with the tile radius set as 74mm. Four noded quadrilateral elements are used to represent the metal components and three noded triangular elements are used for the ceramic. Each component is initially set up as an individual discrete body with contact conditions between the bodies modelled using Coulomb friction. The centreline is modelled using a contact surface as a shield to prevent any object crossing the symmetry axis. The initial finite element discretisation is shown in Fig. 6.

The tungsten is modelled using an Armstrong-Zerilli AZUBCC model whilst the RHA is modelled using an AZMBCC model. Topological changes via erosion due to plastic strain is employed for both materials. The ceramic is treated as a brittle fracturing material and is modelled using the rotating crack model.

Two impact velocities, 1325m/s and 1800m/s, are considered.

The development of damage in the ceramic with increasing penetration at different stages is shown in Figs. 7a -7d. The configuration at $t = 200\mu$s is also depicted in Fig. 8.

Similar to the previous examples, the *a posteriori* graph model for discrete objects achieves better performance, which is demonstrated in Table 3.

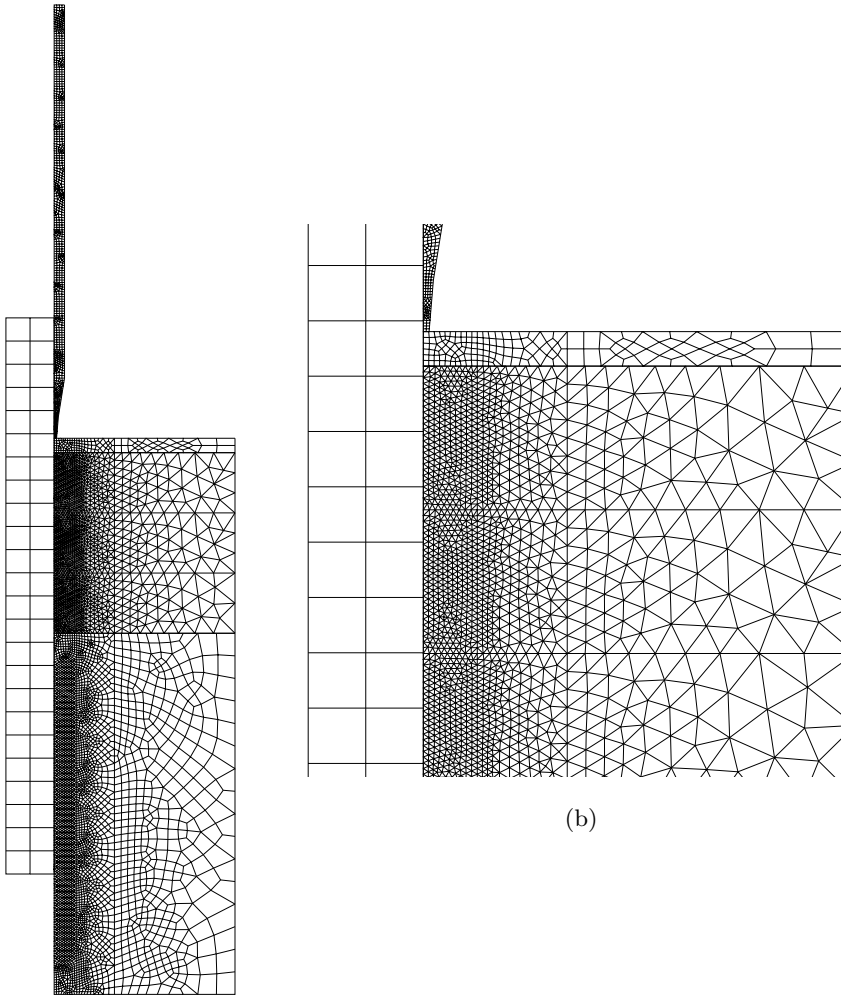
Table 3. Speedup obtained by two graph models for Example 3

Model	2 processors	4 processors	6 processors
a priori model	1.82	3.52	4.60
a posteriori model	1.86	3.63	5.33

7 Concluding Remarks

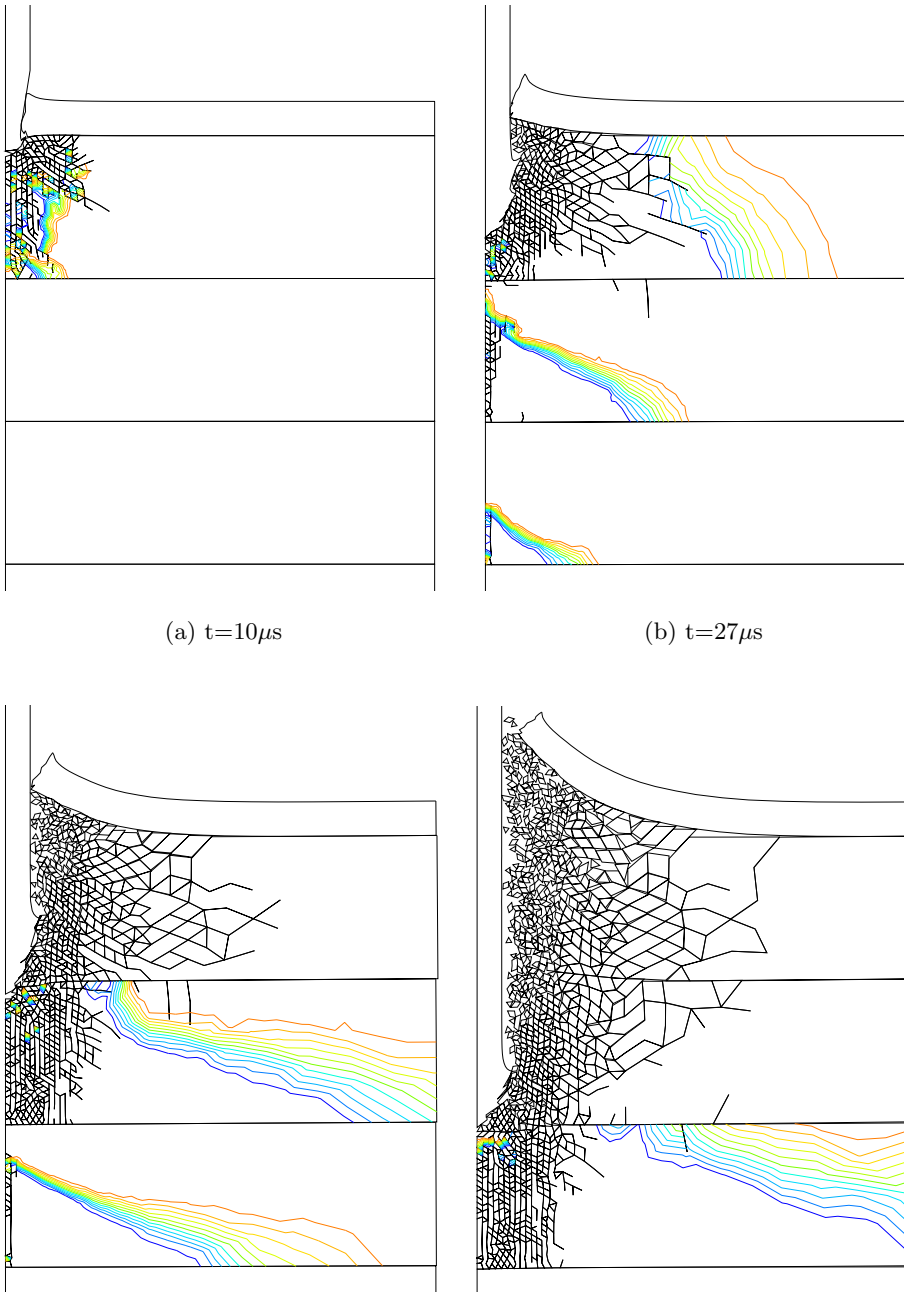
The algorithmic aspects of a parallel implementation strategy for a combined finite and discrete element approach are presented in this work. The main features of the implementation include: 1) a dynamic domain decomposition is applied independently to the finite element computation, the contact detection and discrete element computation; 2) different methodologies can be employed

in the global search and the interaction computations; 3) a dynamic graph repartitioning is used for the successive decomposition of the moving configuration; 4) two graph models are proposed for the representation of the relationship between the discrete objects; 5) load imbalance can be monitored and re-balanced by the proposed scheme.



(a) (b)

Fig. 6. Example 3 - Ceramic target impact: (a) initial mesh; (b) zoomed mesh.



(a) t=10μs (b) t=27μs

(c) t=40μs (d) t=72.5μs

Fig. 7. Example 3 - Ceramic target impact: Progressive damage indicating regions with radial fractures

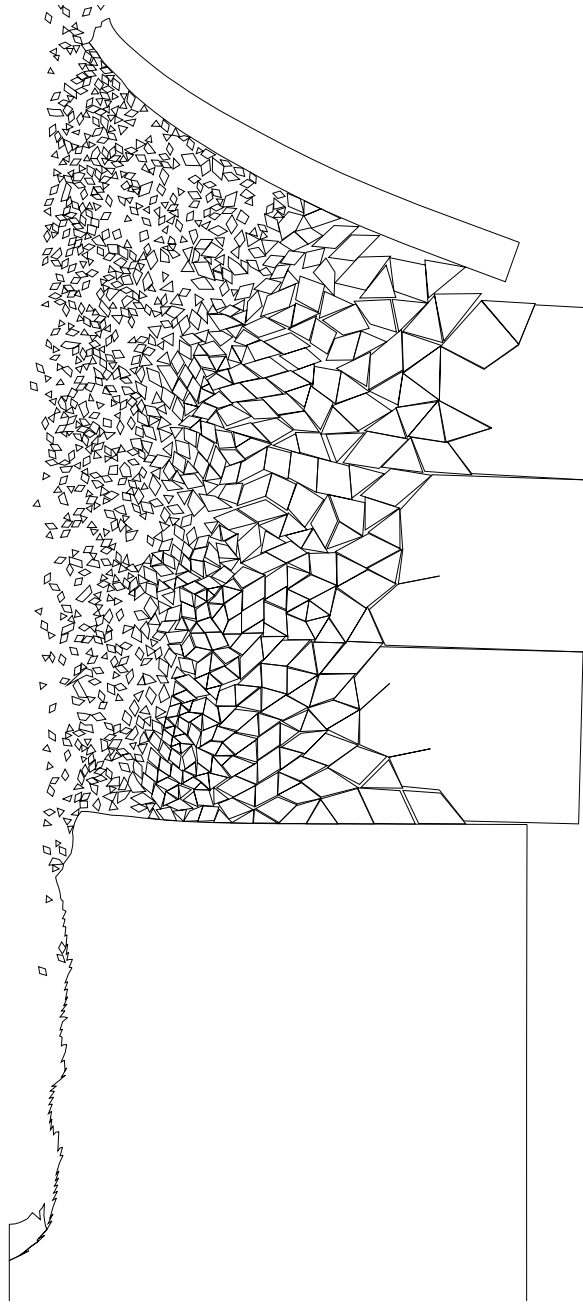


Fig. 8. Example 3 - Ceramic target impact: The configuration at t= 200μs

By means of numerical experiment, the performance of the proposed algorithms is assessed. It is demonstrated that the dynamic domain decomposition using the second graph model with dynamic re-partitioning, load imbalance detection and re-balancing scheme can achieve a high performance in applications involving both finite and discrete elements. It is worth mentioning that the strategy suggested can also be applied to other areas such as smooth particle hydrodynamics (SPH), meshless methods, and molecular dynamics.

8 Acknowledgement

This work is partially funded by the EPSRC of UK under grant No. GR/M01449. This support is gratefully acknowledged.

References

1. Owen, D. R. J., Feng, Y. T., Han, K., and Perić, D.: Dynamic domain decomposition and load balancing in parallel simulation of finite/discrete elements. In *European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2000)*, Barcelona, Spain, 11-14 September 2000.
2. Brown, K., Attaway, S., Plimpton, S., and Hendrickson, B.: Parallel strategies for crash and impact simulations. *Comp. Meth. Appl. Mech. Engng.*, 184:375–390, 2000.
3. Hendrickson, B., and Devine, K.: Dynamic load balancing in computational mechanics. *Comp. Meth. Appl. Mech. Engng.*, 184:485–500, 2000.
4. Yu, J.: *A Contact Interaction Framework for Numerical Simulation of Multi-Body problems and Aspects of Damage and Fracture for Brittle Materials.* Ph.D. Thesis, University of Wales Swansea, 1999.
5. Feng, Y. T., and Owen, D. R. J.: An augmented digital tree search algorithm in contact detection. 2000 (to appear).
6. Han, K., Perić, D., Crook, A.J.L., and Owen, D.R.J.: A combined finite/discrete element simulation of shot peening process. part I: Studies on 2D interaction laws. *Engineering Computations*, 2000 (in press).
7. Han, K., Perić, D., Owen, D.R.J., and Yu, J.: A combined finite/discrete element simulation of shot peening process. part II: 3D interaction laws. *Engineering Computations*, 2000 (in press).
8. Karypis, G., and Kumar, V.: METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system. *Technical Report, Department of Computer Science, University of Minnesota*, 1998.
9. Macedo, J.: *An Integrated Approach to the Parallel Processing of Explicit Finite/Discrete Element Problems.* Ph.D. Thesis, University of Wales Swansea, 1997.