Interprocessor communication schemes in parallel finite-discrete element analysis on PC clusters

F. Wang
College of Water Conservancy and Civil Engineering, China Agricultural University, Beijing, People’s Republic of China

Y.T. Feng and D.R.J. Owen
Civil and Computational Engineering Centre, School of Engineering, University of Wales Swansea, Swansea, UK

Keywords Finite element analysis, Numerical analysis, Parallel programming

Abstract Some issues related to effective parallel implementation of the combined finite-discrete element approach on PC clusters are discussed. Attention is focused on the interprocessor communications. Three communication schemes suitable for different problems are presented. The worker-to-manager scheme is simple to implement. The neighbour-to-neighbour scheme is sophisticated with regard to programming, and requires extra memory space, but has good overall performance for larger problems. The mixed worker-manager scheme can balance the difficulty in programming and the overall communication performance. The effects of subdomain buffer zone on communications are also demonstrated by numerical examples.

1. Introduction
The combined finite-discrete element method has matured considerably over the last decade as a computational tool for simulating problems comprising a large number of separate bodies (Owen and Feng, 2001). However, it is difficult to obtain solutions to very large scale problems on sequential machines, including the latest PCs. Even for a smaller practical engineering problem, the wall clock time required to perform 3D simulations still remains substantial.

Recently several attempts have been made to execute simulations on parallel computing environments (Owen and Feng, 2001; Owen et al., 2000, 2001) Some parallel algorithms, including parallel contact search, graph representation of elements, dynamic domain decomposition, and dynamic load balancing have been reported. However, most of the work is based on shared-memory

This work is funded by the Engineering and Physical Sciences Research Council (EPSRC) of UK under grant No. GR/R21219. This support is gratefully acknowledged. The authors also wish to express thanks to James Armstrong, who enabled us to run the performance studies on the PC cluster.
machines. Generally, super shared-memory computers are not available in many industrial organizations. With the remarkable advances in the power of PC hardware, performing simulations on a network of PC is in many cases the most direct and least expensive solution.

A network of PCs can be viewed as a distributed-memory environment. However, the interprocessor communication overhead plays a much active role in the parallel implementation (Carre et al., 1999) The algorithm adopted in interprocessor communications determines the complexity of programming and the overall parallel performance, which is the goal pursued by researchers (Krysl and Belytschko, 1998) and therefore, the algorithm for interprocessor communication needs to be given more consideration when dealing with the parallel programming.

The aim of this paper is to present some guidelines for conversion of an existing sequential program to run on a PC cluster with different user requirements. Particularly, the algorithms that can be adopted for interprocessor communication are investigated.

The outline of this paper is as follows. First, the performance of a typical PC cluster is investigated in Section 2. Next, the issues related to parallel implementation of a combined finite-discrete element approach are discussed in Section 3. Then, in Section 4, we focus our attention on interprocessor communication schemes. Finally, Section 5 provides numerical examples to compare the various schemes.

2. Communication performance of PC clusters
Before discussing the interprocessor communication schemes, it is necessary to present some numerical test results to estimate the communication performance of a PC cluster.

The cluster tested consists of seventeen 1 GHz processors running LINUX. Each processor is equipped with 512 MB local memory. One of the machine is the master node and the other 16 are slave nodes. The slave nodes have a cut down version of LINUX which contains the bare minimum for the machine to operate in the cluster. The 16 slaves and the master node are connected by a private 100baseT network operating through a fast ethernet switch. The master node has a second network connection which connects to the public network. Only the master node is accessible from other (non-cluster) machines. The slave nodes are of a rack-mounted configuration housed in a 45U rack-mount cabinet.

The test program is a parallel code that can be used to perform finite-discrete element simulations. The MPICH package (Gropp et al., 1996; Group et al., 1994; MPICH Home Page, 2002) is selected as the message-passing library. The standard GCC C compiler is used to compile the program. The communication timing results are obtained by calling standard MPE functions provided by
MPICH, and is abstracted by Jumpshot software (MPICH Home Page, 2002). The timing results are shown in Table I.

Based on the above results, the time required to send a message through the PC cluster can be approximately expressed as:

\[ T_m = T_0 + T_b B \]  

(1)

where \( T_m \) is the time required to communicate a message of \( B \) bytes, \( T_0 \approx 1 \text{ ms} \) is the network latency time, and \( T_b \approx 8.84 \times 10^{-5} \text{ ms/byte} \) is the time required to transfer one byte of data.

It is clear that the communication costs are of the same level for transferring up to 144 kB message. Additionally, the larger the data packets are, the lower the effect of network latency is, and the better the efficiency achieved. Therefore, to “eliminate” the effect of network latency, it may not be necessary to pursue the goal of minimising data exchange.

3. Parallel computational model in finite-discrete element analysis

The combined finite-discrete element system comprises some continuous regions and a large number of separate bodies (Munjiza et al., 1995). The continuous regions are usually discreted by finite elements. The original separate bodies, such as particulates or fractured zones generated from continuous regions are represented by discrete elements. For simplifying the simulation, discrete elements are usually treated as rigid and represented by simple geometric entities, such as disks or spheres, while the finite elements are deformable (Owen et al., 2000).

3.1 Governing equation

Using the standard discretisation procedure for the finite element method, the discretised governing equation for the finite element sub-system can be written as (Munjiza et al., 1995; Watson and Noor, 1997).

\[ M \dot{u} + C \ddot{u} + F^{\text{int}} - F^{\text{ext}} - F^c = 0 \]  

(2)

<table>
<thead>
<tr>
<th>No.</th>
<th>Message type</th>
<th>Number of messages</th>
<th>Message volume (byte)</th>
<th>Time elapsed (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Integer</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Double real</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Integer</td>
<td>1,000</td>
<td>4K</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Double real</td>
<td>1,000</td>
<td>8K</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Mixed structure</td>
<td>3,004</td>
<td>144K</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Mixed structure</td>
<td>239,000</td>
<td>480K</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>Mixed structure</td>
<td>503,024</td>
<td>1M</td>
<td>105</td>
</tr>
<tr>
<td>8</td>
<td>Mixed structure</td>
<td>503,024</td>
<td>2M</td>
<td>218</td>
</tr>
<tr>
<td>9</td>
<td>Mixed structure</td>
<td>239,000</td>
<td>3.8M</td>
<td>377</td>
</tr>
<tr>
<td>10</td>
<td>Mixed structure</td>
<td>239,012</td>
<td>6.9M</td>
<td>610</td>
</tr>
</tbody>
</table>

Table I. Interprocessor communication results
where $\mathbf{M}$ and $\mathbf{C}$ are the mass and damping matrices, respectively, $\mathbf{F}^{\text{int}}$ is the global vector of internal resisting nodal forces, $\mathbf{F}^{\text{ext}}$ is the vector of consistent nodal forces for the applied external loads, $\mathbf{F}^{c}$ is the vector of consistent nodal contact forces, $\ddot{\mathbf{u}}$ is the global vector of nodal accelerations and $\dot{\mathbf{u}}$ is the global vector of nodal velocities.

For a discrete element sub-system, the discrete elements are governed by the Newton’s second law:

$$\mathbf{M}\ddot{\mathbf{u}} - \mathbf{F}^{\text{ext}} - \mathbf{F}^{c} = 0$$

(3)

The two sub-systems are coupled by contact forces, $\mathbf{F}^{c}$. Because a discrete element can be viewed as having the same topology as a finite element, i.e. one discrete element has only one node, equation (3) can be viewed as a special case of equation (2). Therefore, the complexity of algorithm design for solving equations (2) and (3) is reduced, and only equation (2) needs to be considered.

In order to obtain the solution of the governing equation (2), a central difference scheme (Owen and Hinton, 1980) is adopted. Thus, the displacements at time $t + \Delta t$ are given explicitly in terms of the displacements at time $t$ and $t - \Delta t$. If the mass matrix $\mathbf{M}$ and damping matrix $\mathbf{C}$ are diagonal then the solution of equation (2) becomes trivial. Details for solving equation (2) can be found in Wang et al. (2003).

3.2 Subdomain and its buffer-zone

Parallel strategies typically attempt to distribute computational work by dividing a large problem into a number of smaller subproblems. Domain decomposition is employed for dividing the entire domain into subdomains. Each subdomain is confined to be an axis-aligned box in this work. Figure 1 shows a system divided into nine subdomains.

A buffer zone used to control the frequency of domain decomposition is introduced along the boundary of each subdomain. The buffer zone of

**Figure 1.**
Subdomains and the buffer-zone of subdomain S6
subdomain S6 is shown in Figure 1. One half of the buffer zone is located inside the current subdomain, and another half is located outside the boundary. Thus, there are three boxes associated with each subdomain which are termed as subdomain box, internal box and external box, respectively.

The choice of the buffer-zone size depends on the type of application. It has conflicting effects on the overall efficiency of the domain decomposition and communications at each time step. A larger buffer zone will result in more elements in each subdomain, which will increase the amount of simulation and communication, but will reduce the frequency with which the domain decomposition has to be performed. With a smaller buffer zone, the number of elements and nodes within the zones will be fewer and the interprocessor communications becomes less expensive at each step, but the domain decomposition should be conducted more frequently thus increasing the computational cost of the decomposition. A carefully selected buffer zone can balance the costs in the two phases to achieve a better overall performance. Some numerical tests will be given in Section 5.

3.3 Type codes of elements and nodes
To perform the parallel computing, particularly to treat the contact between the elements on the boundary, all elements in a subdomain are classified into internal, interfacial and external elements. If the bounding box of an element is totally inside the internal box of a subdomain, such as element A in subdomain 1 shown in Figure 2, it is defined as internal. If the bounding box of an element overlaps with the subdomain buffer zone and its centre is inside the subdomain box, the element is defined as interfacial, such as elements B and E in subdomain 1. Other elements are defined as external, such as elements C and F.
in subdomain 1. Note that element D is not associated with subdomain 1 and therefore no classification is given. We introduce type codes “I”, “F”, and “X” to represent the types of internal, interfacial and external elements, respectively.

Similarly, all nodes in a subdomain are classified into internal and external nodes. If a node is located inside the subdomain box, it is defined as an internal node, and is assigned a type code “T”. Otherwise, it is defined as an external node, and is assigned a type code “X”.

Thus, all contact pairs can be identified by six labels: II, IF, IX, FF, FX, and XX. For instance, the contact between two internal elements can be labelled as II.

The basic parallel strategy to be proposed is based on these definitions. Each processor performs the computation of internal forces only for internal and interfacial elements (with type codes I and F), and performs the temporal integration only for internal nodes (with type I). The details regarding how to use the type codes in contact computations are given in Sections 3.5 and 4.1.

3.4 Parallel computational procedure

In the context of the explicit integration scheme, a combined finite-discrete element approach typically performs the contact computations, internal force computations, external force computations, temporal integration, and configuration update.

To perform parallel computation on distributed-memory environments, the manager/worker approach is adopted in the present work. Figure 3 shows the parallel computational procedures implemented on manager processor and worker processors. The manager processor serves as the controller for the computations by conducting the necessary sequential calculations, including partitioning/re-partitioning the entire domain into subdomains. Each worker processor stores all the data for the problem confined within its subdomain, and performs simulation for the problem. The worker processor operates as an usual processor that is running a sequential code. The main change is that some communication functions are added to exchange information with other worker processors or with the manager processor.

As shown in Figure 3, once initial data are distributed by the manager processor, the local element computations, including contact, internal force, external load, and nodal displacement calculations are conducted in each worker processor as a sequential calculation.

The items that are highlighted with boxes are not performed at every time step.

3.5 Contact detection and contact force evaluation

The contact computations generally include global search, local resolution, and contact force evaluation. As mentioned above, all three phases are conducted locally in each subdomain as a sequential process.

In the global search phase, each worker processor is responsible for detecting all potential contact pairs within its own subdomain. Any element,
whether it is internal, interfacial or external, is treated as a regular element, and there is no need to distinguish contacts between elements with different type codes. Therefore, any algorithm used in the sequential code can be employed to perform the global search in this parallel strategy. In the present work, the Augmented Spatial Digital Tree (ASDT) algorithm (Feng and Owen, 2002) is adopted to search all the potential contact pairs in a subdomain. This algorithm can accommodate various geometrical entities including facets and particles. This is achieved by representing each entity with an axis-aligned bounding box extended by a buffer zone. However, the contacts between the boundary elements will be specially handled in the communications for contact forces.

After the global search, each potential contact pair is locally resolved on the basis of their kinematic relationship and individual geometric shapes by employing a local contact resolution procedure. The local resolution procedure is a purely geometric operation, and thus no further description is given here.

---

**Figure 3.**
Flow chart of the parallel computational procedure

---

Interprocessor communication schemes
Once an actual contact pair is determined, a contact interaction algorithm is employed to evaluate the contact forces. In this work, a penalty function based interaction law (Han et al., 2000a, b) is employed. This approach can deal with normal and tangential contact forces effectively, especially in 3D problems. It should be mentioned that this approach is history dependent. Thus, the data redistribution for contact history must be performed in every domain decomposition.

The contact force acting on an element should be distributed among all nodes of the element. For finite elements, there is a standard procedure that distributes the force according to the element shape-functions. For discrete elements, the distributing procedure is more straightforward.

3.6 Dynamic domain decomposition

Domain decomposition is one of the basic approaches employed for parallel computing. It normally includes data gathering, partitioning, redistributing, and database updating. A number of domain decomposition methods (Hendrickson and Devine, 2000; Hsien et al., 1997; Karypis and Kumar, 1998; Karypis et al., 1998) including partitioning algorithms and load balancing algorithms, have been proposed in recent years. However, these methods are mainly focused on general finite element analysis. For combined finite-discrete element simulations, there are often a large number of moving particles that contact with neighbouring particles or finite elements. As neighbours vary over time, these contacts change frequently and this dynamic contact feature dominates the whole system. Therefore, how to handle contacts in the domain decomposition method will have significant effects on the parallel performance.

In this work, a dual-level domain decomposition strategy (Wang et al., 2003) is adopted. In this strategy, the decomposition can be at a low level or at a high level. At a low level, all subdomain positions are updated with the evolution of the sub-problems. At a high level, only the sub-problems are updated, while the subdomain positions remain unchanged. Thus, the partitioning procedure is not required in the high-level decomposition. The time instances at which the low-level or high-level decomposition should be performed depend on two parameters: the accumulative maximum displacement and the level of workload imbalance. The accumulative maximum displacement since the last decomposition can be evaluated based on the maximum velocity in each time step and the step size. The level of workload imbalance in the current time step can be evaluated based on the actual runtime of each processor during the time step.

If the accumulative maximum displacement is more than or equal to one half of the size of the subdomain buffer zone, i.e. when an internal element of the current subdomain has crossed the subdomain boundary to enter a neighbouring subdomain, the high-level domain decomposition has to be performed. All the data associated with this element need to be
copied/migrated so as to keep the completeness and validity of each sub-problem. Clearly, the size of the subdomain buffer zone will significantly affect the frequency of such decomposition.

If the level of workload imbalance exceeds a prescribed value, i.e. a significant workload unbalancing is detected, the low-level decomposition must be performed. At this level, a global domain re-partition is involved.

To partition/re-partition the entire domain, a recursive coordinate bisection (RCB) (Hendrickson and Devine, 2000) based algorithm is used in the present work. The algorithm takes the geometric locations of all the nodes, determine in which coordinate direction the set of nodes is most elongated, and then divides the nodes by splitting based upon that long direction. The two halves are then further divided by applying the same algorithm recursively. This algorithm is detailed by Wang et al. (2003).

To balance the workload in the partitioning process, the contact list is used to evaluate the computational cost of each element. For the simulation that is dominated by one type of interaction and the objects have nearly even distribution across the whole domain, the global contact list can be used. However, it is more reasonable to use the local (actual) contact list generated at the last time step for general problems. Once the computational costs of all elements are determined, the workload of each node can be derived from its corresponding elements.

4. Interprocessor communications
As shown in Figure 3, there are three groups of communications during a regular time step. The first group is the exchange of the nodal forces of the boundary nodes shared by adjacent subdomains. The nodal forces consist of contact forces, internal forces and external loads. The second group of communications exchanges the displacements and velocities of the boundary nodes after the temporal integration. The third group of communications is the exchange of necessary information between the manager processor and worker processors. The “necessary” information consists of three major parts. Two of these are the maximum velocity and the actual runtime of each processor in the current time step. They are sent from the worker processors to the manager processor. The maximum velocity is used to evaluate the accumulative maximum displacement, and the actual runtime is to evaluate the workload. Another part of the “necessary” information is a flag that indicates whether a domain decomposition needs to be performed. This flag is set, and broadcast by the manager processor according to the maximum velocities and actual runtimes of all worker processors.

When a domain decomposition is required to be performed during a step, the following additional three or four groups of communications are necessary. The fourth group of communications involves gathering the data on the manager processor that is responsible for domain decomposition. The fifth
The group of communications is the redistribution of the subdomain data after the decomposition. The sixth group of communications is the redistribution of the element data. If friction is considered, the seventh group of communications is required to exchange the contact history because the evaluation of the tangential component of contact forces is dependent on the contact history.

Among the seven groups, the communications for exchanging the contact forces and contact history are not as straightforward as the others. Thus, special consideration is given to these communications in this section. Subsequently, we discuss different interprocessor communication schemes used to complete the seven groups of communication. Numerical comparison and further analysis will be presented in Section 5.

4.1 Communication for contact force

When examining the domain decomposition scheme considered, one should notice that one contact pair may have two or more copies in two or more adjacent subdomains. For instance, if the interfacial element E (shown in Figure 2) contacts with external element B, the contact pair generated in the subdomain 1 can also be found in subdomain 2, although its label is changed from FF to XX. Except for II, IF, and IX contact pairs, all other contact pairs probably have at least two copies in different subdomains. In the phase of contact force evaluation, the contact forces are computed for all contact pairs in each subdomain except for XX contact pairs. Thus, the duplicated computations may be performed for FX contact pairs in different subdomains. Although the FF contact pair may exist in another subdomain, it must be a XX contact pair in that subdomain. Therefore, the duplicated computations do not occur for FF contact pairs.

The data redundancy and duplicated computations for some contact pairs consume more memory space and CPU time, but a good overall performance can be achieved. In particular, the communication of contact forces is considerably simplified and the complexity of programming is reduced. This can be demonstrated from the following analysis for contact force distributions.

For a contact pair labeled as II, IF, IX, or FF, the contact forces acting on the two contact elements will be distributed among their nodes by the standard distribution procedure described in Section 3.5. If the contact label is FX, a further check should be carried out to determine whether the interfacial element could be found in the subdomain to which the external element belongs. If the answer is no, the contact force should be distributed in the same way as above. Otherwise, the contact force should be distributed only to the interfacial element.

After the distribution, the nodal forces of the nodes that have the X type codes will be transferred to the other subdomains by the communication. It should be mentioned that each processor performs the computations of internal
forces and external loads only for internal and interfacial elements, and performs the temporal integration only for internal nodes.

As illustrated above, no special communication step is introduced for exchanging contact forces, which is integrated with the communication for internal forces of finite elements and external forces applied on elements. On the other hand, except for the checking operation for FX contact pairs, no other computation is required in the contact evaluation in parallel environments. Despite the data redundancy and duplicated computations for some contact pairs, the simplification of parallel programming and reduction of communication are significant.

4.2 Exchange of contact history

Almost all the practical algorithms used to evaluate contact forces with friction are dependent on contact history, including the one employed in the present work. For each contact pair, the history includes the contact facet/segment number, contact direction vector, normal contact force, and tangential contact force. Because a new object may be created or existing ones may disappear in a subdomain after domain decomposition, the contact history has to be copied/migrated into new subdomains to keep the completeness and validity of contact information.

The contact history of an element is different from the other data of the element, because it not only depends on its own element, but is also associated with other elements. From our experience, the exchange of contact history is one of the most sophisticated phases in parallelising an existing sequential code for combined finite-discrete element simulations on distributed-memory environments. It should be carefully designed to obtain a good overall performance for different problems. Some of the main aspects for exchanging contact history are considered below.

The first step for exchanging the contact history is to store the contact history locally or globally before the decomposition is conducted. In the contact calculations, the contact facets instead of elements are used as the basic objects and they are locally defined in each subdomain. An element may move into another subdomain after the decomposition, thus the facet number in the contact history must be replaced by the combination of an element number and a local serial facet.

Secondly, the contact history list to be exchanged must be generated according to the decomposition results. The generating procedure can be carried out locally in each worker processor, or globally in the manager processor, depending on the interprocessor communication scheme used. If it is local, the worker processors must keep track of which processor is responsible for which interface element. Thus the algorithm may be complicated, and the number of communications will increase. If the generating procedure is global,
the communication algorithm will be simpler, but the memory requirements in the manager processor will be high.

The last step is to reorganise the contact history in a “new” subdomain according to the new contact facet number. This step follows the interprocessor communication.

4.3 Interprocessor communication schemes

As discussed in Section 2, parallelisation for a PC cluster, or computer network, requires that the number of communications is minimised and that a much larger number of messages are sent to ameliorate the effects of network latencies. A high performance can be obtained only if the number of communications and the amount of messages communicated are kept at an optimal combination. On the other hand, the complexity of programming should be considered when designing the communication algorithm. In what follows, we present three schemes to perform the interprocessor communication on a PC cluster.

4.3.1 Neighbour-to-neighbour scheme. In the neighbour-to-neighbour scheme, the worker processors exchange information with other worker processors as necessary. For instance, the nodal forces of a node with type code X are communicated directly between the current processor and another processor that is responsible for the temporal integration of this node.

In this scheme, the worker processors must keep track of which processor is responsible for which interface node/element. For different groups of communications, there are different sets of interface nodes/elements. For example, a node-set is required for the nodal forces communication between the current subdomain and any other subdomain. The nodes in the set must have a type code X in the current subdomain and a type I in the other subdomain.

During a regular time step without involving a domain decomposition, the first three groups of communications can be implemented rather straightforwardly. The challenge to the neighbour-to-neighbour scheme is the communications after a domain decomposition is undertaken, especially for the sixth and seventh groups of communication presented above. For the sixth group, i.e. the communications for redistributing element data, several element sets that contain the boundary elements must be built for all other worker processors. Building these sets may take much more time in programming and in running, because some necessary information regarding the boundary elements may require extra communications between the current processor and manager processor, following exchanging the sets between any two worker processors. Then, each worker processor needs to reorganise the elements after it has received all the information from all other worker processors.

For the seventh group, i.e. the communications for the contact history, an additional difficulty is that another element and a local facet number are involved in the contact history of an element. This difficulty is present before
and after the communication. It requires more memory requirements on the worker processor and makes the related computation more sophisticated.

Based on the above analysis, the characteristics of the neighbour-to-neighbour scheme can be described as follows:

1. The communication consists of a large number of small-grain messages (typically only several floating-point numbers) and the total number of interprocessor communications is large. However, the communication between the neighbours can proceed concurrently in some cases if independent communication paths between different processors are available. Therefore, this communication scheme may be faster in some cases.

2. In order to keep track of which processor is responsible for which interface node, the worker processors must undertake the book-keeping. This will increase the local memory requirement on the worker processors and make the programming more complex.

Such an algorithm has been reported by many researchers (Brown et al., 2000) for finite-element analysis. It is the most commonly employed scheme in parallel computing, especially on MPP machines.

4.3.2 Worker-to-manager scheme. In the worker-to-manager scheme, the worker processors exchange information only with the manager processor. For instance, the nodal forces of a node with a type code X are sent to the manager processor from the current processor, and then the forces assembled are distributed to the corresponding worker processor by the manager.

In this scheme, the manager is a driving program that is responsible for assembling and distributing interface data. The workers manipulate the partitions as if they are ordinary sequential programs, hiding a few points at which a communication with the manager is necessary in a well-protected “parallel” interface.

This scheme simplifies the algorithms dealing with communications. For example, to exchange the contact history, the workers simply require to send their contact pairs to the manager. The workers do not need to keep track of which processor is responsible for which interface node/element. Thus, the memory requirements for the worker processor and the programming complexity decrease accordingly. Because the manager already holds the information regarding which processor is responsible for which node/element, it is convenient and fast for the manager to assemble all the contact lists and distribute them to the workers.

The total number of interprocessor communications in this scheme is also very low. For instance, if eight worker processors are used to perform the simulations, the number of communications for the contact history is at least 56 in the neighbour-to-neighbour scheme. Note that other communications, such as the tracking of the boundary elements and nodes, are excluded here.
However, the number of interprocessor communications is only 16 in the worker-to-manager scheme, and no other communication is necessary. At the same time, large message size may be obtained, which is suitable for ameliorating the effects of the network latencies.

Based on the above analysis, the characteristics of the worker-to-manager scheme can be summarised as follows.

1. Compared with the neighbour-to-neighbour scheme, large-grain message sizes are obtained in the worker-to-manager scheme, and the total number of interprocessor communications is low. Thus, the effects of the network latency decrease. However, the interprocessor communications can only be conducted serially, while this could be concurrent in the neighbour-to-neighbour scheme.

2. The workers do not need to keep track of which processor is responsible for which interface element/node, thus the memory requirements for worker processors are not as high as in the neighbour-to-neighbour scheme. Also, the parallel programming complexity decreases considerably. The workers manipulate the partitions as if they are ordinary sequential programs. Few algorithms are required to handle the parallel communications. This feature may be very important in some cases.

3. Since all the data of the boundary element and nodes are assembled on the manager processor, the memory requirements of the manager processor may be high. Due to the fact that the manager processor may not perform the simulation, this higher memory requirement may be acceptable in many cases. If possible, a machine with large memory can be employed as the manager.

Such an algorithm has been reported for use in a finite element analysis on PC networks (Krysl and Belytschko, 1998). Some numerical studies are presented in the next section to further compare it with the neighbour-to-neighbour scheme, and test its applicability for combined finite-discrete element analysis.

4.3.3 Mixed worker-manager scheme. The mixed worker-manager scheme is a combination of both the neighbour-to-neighbour scheme and the worker-to-manager scheme. In this scheme, the neighbour-to-neighbour scheme is adopted for the first two groups of communications and the worker-to-manager scheme is adopted to perform the other communications. Thus, the data communications during a time step without decomposition are performed between the worker processors, while all the data exchanges after domain decomposition take place between the manager processor and the worker processors.

The performance of this scheme is between the other two schemes. It does not have the serious drawbacks associated with the other two schemes. For example, it can avoid the sophisticated programming for the exchange of the
contact history and element data, and at the same time it can obtain a better overall performance in many cases.

5. Numerical studies and performance evaluation
The present code was implemented on the PC cluster described in Section 2. Two examples are used to evaluate the interprocessor communication schemes.

5.1 Impact between a large sphere and a system of smaller spheres
As shown in Figure 4(a), a total of 3,375 small spheres (radii 10 cm) occupy a cubical domain in a fixed box with one side free, and a large sphere (radius 35 cm) above the cubical domain falls freely and impacts the small spheres under the influence of gravity.

The time step is fixed at $1 \times 10^{-5}$ s. The friction coefficient is taken as 0.5. The contact damping coefficients are taken as 0.8 in both normal and tangential directions. The total degrees of freedom for the simulation are 20,274. Figure 4(b) shows the distribution of the spheres at time = 1.5 s. The total runtimes for different interprocessor communication schemes with various number of processors are presented in Figure 5.

It takes 150,000 time steps to simulate a duration of 1.5 s, during which 190 domain decompositions are conducted when the size of the subdomain buffer zone is set to be 4 cm. Figure 5 shows that the best performance is obtained when communication scheme 2 (worker-to-manager scheme) is adopted for all

![Figure 4. Impact of a large sphere with some smaller spheres](image)
If the size of the subdomain buffer zone increases from 1 to 8 cm, the number of domain decompositions will decrease, but the boundary elements and nodes will increase. Therefore, the amount of communication during a regular time step will increase and the total amount of communications for the steps with decomposition will decrease because the frequency of decompositions is lower. However, the size of all communication messages during a regular time step is less than 144 kB. Due to the large network latency mentioned in Section 2, the increase of the amount of communication during a regular time step will not result in a significant increase of runtime. Thus, the overall performance for scheme 2 with a larger buffer zone is the best option in this case. Figure 6 shows the effect of the size of subdomain buffer zone when four processors are used. Note that the performance difference between 8 and 4 cm of buffer zone sizes is not as large as between 4 and 1 cm. In our experience, the buffer zone size can be set to be one half of the average radius of all discrete elements, regardless of the particle velocities in the simulation.

Figure 5. Total runtime with different interprocessor communication schemes

Figure 6. Total runtime with different buffer zone sizes (number of processors = 4)
5.2 Dragline bucket filling

This example simulates a dragline bucket filling process, where the blasted soil/rock is modelled by discrete elements as a collection of layered particles (spheres). The bucket is modelled as conventional finite elements and the filling is simulated by dragging the bucket with a prescribed motion. This example contains 101,322 discrete elements and 13 finite elements resulting in a total number of 607,980 DOF. The configuration of the problem at an intermediate stage of the simulation is shown in Figure 7.

Up to eight processors are used to simulate this problem. Data in the first subdomain at time step = 1,000 are shown in Table II. Figure 8 shows the total runtime for the simulation of 1,000 steps. In this problem, the best performance is obtained when communication scheme 1 (neighbour-to-neighbour scheme) is adopted for all the cases, whereas the communication scheme 2 (worker-to-manager scheme) yields the best result in the first example. The performance difference between scheme 1 and scheme 2 increases with the number of processors used for this larger problem. For the simulation with eight processors, communication scheme 1 runs about 35 per cent faster than

<table>
<thead>
<tr>
<th>Number of worker processors</th>
<th>1 (sequential)</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOF</td>
<td>607,980</td>
<td>309,864</td>
<td>160,794</td>
<td>88,392</td>
</tr>
<tr>
<td>Number of discrete elements</td>
<td>101,322</td>
<td>51,636</td>
<td>26,793</td>
<td>14,725</td>
</tr>
<tr>
<td>Number of finite elements</td>
<td>13</td>
<td>12</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>Number of total nodes</td>
<td>101,338</td>
<td>516,52</td>
<td>26,809</td>
<td>14,739</td>
</tr>
<tr>
<td>Number of internal nodes</td>
<td>–</td>
<td>49,880</td>
<td>25,037</td>
<td>12,656</td>
</tr>
<tr>
<td>Number of global (potential) contact pairs</td>
<td>1,81,8309</td>
<td>914,139</td>
<td>462,752</td>
<td>249,725</td>
</tr>
<tr>
<td>Number of local (actual) contact pairs</td>
<td>52,146</td>
<td>20,838</td>
<td>7,806</td>
<td>3,580</td>
</tr>
</tbody>
</table>

Table II. Data in one subdomain (first subdomain)
scheme 2. The reason for this is that the size of all communication messages are relatively large, and the effect of network latency is low. Thus, the communication time is approximately proportional to the amount of communication. It should also be noted that scheme 3 has almost the same performance as scheme 1.

6. Concluding remarks
Some issues related to interprocessor communication in combined finite/discrete element simulations on a PC cluster are discussed. Three communication schemes that have different programming complexities and different performances for different problems are presented. Based on the discussions and the numerical comparative studies presented, the conclusions of this work are summarized as follows.

(1) Unlike other distributed-memory environments, a PC cluster (or computer network) has a significant network latency. All algorithms related to interprocessor communications should be carefully considered.

(2) The worker-to-manager scheme is easy to implement. It is usually suitable for smaller problems. If the resulting parallel efficiency is not a main issue, this scheme is a good choice even for larger problems. In this scheme, more memory may be required on the manager processor to hold some global information.

(3) The neighbour-to-neighbour scheme exhibits good suitability for larger problems. It is about 35 per cent faster than the worker-to-manager scheme in our numerical tests. However, it has extra memory requirements on each worker processor to keep track of which processor is responsible for which interface node, and has a higher programming complexity.
(4) The mixed worker-manager scheme has a similar performance to the neighbour-to-neighbour scheme, but its programming complexity is lower than for the latter. It can balance the difficulty in programming and the overall communication performance.

(5) The size of subdomain buffer zone has significant effects on communication. It determines the decomposition frequency and the amount of boundary information to be transferred. A large value is suitable for smaller problems, while the effect on larger problems is not significant. It usually can be set to be one half of the average radius of all particles in the system.

References


**Further reading**
