Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computers and Structures 82 (2004) 1707–1718

Computers & Structures

www.elsevier.com/locate/compstruc

Acceleration of the Discrete Element Method (DEM) on a reconfigurable co-processor

Benjamin Carrión Schäfer [a], Steven F. Quigley [a], Andrew H.C. Chan [b,*]

[a] *Department of Electronic, Electrical and Computer Engineering, University of Birmingham, 52 Pritchatts Road, Edgbaston Birmingham B15 2TT, UK*

[b] *Department of Civil Engineering, School of Engineering, University of Birmingham, Edgbaston Birmingham B15 2TT, UK*

Received 19 December 2002; accepted 4 March 2004
Available online 17 June 2004

Abstract

The Discrete Element Method (DEM) is a numerical method devised to model the behaviour of particle assemblies. However in order to simulate entire engineering structures, which may involve millions of particles, the computing power has to increase as the DEM is computationally extremely expensive.

A dedicated hardware architecture, implemented on a reconfigurable computing platform based on a Field Programmable Gate Array (FPGA) is presented in this paper. The main computational tasks are fully overlapped using domain decomposition techniques, and the lower level parallelism is also exploited by using concurrent arithmetic operations. A speedup of a factor of 30 could be observed compared to an optimised software simulator running on a 1 GHz Pentium III PC with 1.3 Gbytes of RAM for 2-D particles assemblies ranging from 25,000 to 200,000 particles. The scalability of this design was tested on a multi-FPGA system, allowing the complete overlap of communication and computation for two FPGA boards working in parallel, achieving a speedup factor of almost 60.

Keywords: Discrete Element Method (DEM); Reconfigurable computing; Field Programmable Gate Arrays (FPGAs); Hardware acceleration

1. Introduction

The Discrete Element Method (DEM) is a numerical method to model the behaviour of particle assemblies. They can be bonded together to represent rock or remained unbonded to represent soil. Bonded together they can represent entire structures, such as dams or bridges. As the process is explicit, the time step must be limited to a very small value, thus making the DEM extremely computationally expensive. Its wide-spread use is therefore hampered, though the amount of parallelism involved in it is also extraordinarily high.

Many attempts have been made, with varying degrees of success, to run the DEM on multiprocessor systems. Linear speedup with the number of processors would be expected, but synchronisation and communication overheads as well as load balancing problems cause these systems to underachieve their expectations of linear speedup. Transputer systems [1], CrayT3D [2] implementations, Swiss-T0-Dual machines [3], Linux clusters [4] are just some of the multiprocessor systems in which the DEM was implemented. Other ways to allow the simulation of realistic particle problems are therefore needed.

As the complexity of Field Programmable Gate Arrays (FPGAs) is continuously increasing, and entire system can now be implemented on them with minimal off-chip resources, they provide an ideal platform for hardware acceleration. They can be configured to form co-processors to perform custom hardware acceleration. For the right type of application they can rival expensive

[*] Corresponding author.

multi-processor systems that are normally used for such purposes. FPGAs thus open a new window to low cost hardware acceleration.

2. The Discrete Element Method

Cundall and his co-worker Strack [5,6] developed the Discrete Element Method (DEM) in the seventies to model the behaviour of granular materials. The main operations of DEM to be performed in each time step are:

(1) Check which particles are in contact with one another
(2) Increment the inter-particle forces
(3) Update the co-ordinate and velocity of each particle

Furthermore, the list of which particles are in contact must be re-computed for each step. As the process is explicit, the time step must be limited to a very small value. It is in the order of milliseconds for the stiffness and density of a typical material. Although using scaled stiffness or density can change its time-step value, this would not be feasible and many time steps would still be needed if the dynamic behaviour of the system were required to be modelled accurately. This restriction makes the DEM extremely computationally expensive. Nevertheless this method has been widely used in many applications such as silo flows [1], rock fracture and the collapse of buildings [2].

The computational effort of the three main steps involved in the DEM is given in the following section for a 2-D case.

2.1. Contact check

In order to detect if two particles are in contact the following equation has to be solved for circular discs in 2-D:

$$\Delta n = R_1 + R_2 - \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} > 0 \qquad (1)$$

Here x_i and y_i are the co-ordinates of each particle and R_1 and R_2 are the respective radii, D is the distance between the centroids and Δn the amount of overlapping of the two particles in contact. The contact check is computationally expensive, as a square root operation and two multiplications are involved in Eq. (1).

The computational operations required to detect which particles are in contact with each other is the most time consuming operation of all three. Identification of which balls are in contact requires that each possible pairing of balls be examined. For N particles this would require $O(N^2)$ operations. Thus, for large problem sizes, contact identification dominates the complexity of the problem. Dividing the domain up into cells using the domain decomposition method can alleviate this [6]. Each particle is tagged as belonging to a particular cell, and it will only be checked for contacts with particles within the same cell and adjacent cells. If there are c numbers of particles per cell, the execution time is then proportional to $\frac{N}{c}O(c^2)$.

Occasionally a particle may transition from one cell to another, or may straddle the boundary between two cells. A new sub-step has to be included in the data flow of the DEM, as reboxing of the particles is now necessary whenever a particle moves to an adjacent box.

The number of arithmetic operations needed to compute Eq. (1) is given in Table 1.

2.2. Inter-particle forces increment

Once the contact list for a particle has been established, the total force acting on it can be determined. For every contact identified between two particles, the resulting force can be calculated once the force–displacement law is known. For this study, a linear elastic force–displacement law is adopted. This is not exactly correct in reality, as the contact area will increase with the amount of contact thus rendering the force–displacement law non-linear. Although many advanced interaction laws such as Hertzian have been proposed [7], they only add to the complexity of the calculation and do not alter substantially the arguments put forward in this paper.

On the other hand, the model used throughout this paper considers a granular medium with all of its particles having identical radius R. This greatly simplifies the Hardware implementation of the algorithm, because it means that for a 2-D implementation, a particle can have a maximum of six other particles in contact with itself.

Having particles of the same radius R would mean that the maximum number of operations for contact checking needed would be $O(6N)$, for the worst case in which all the particles have the maximum number of particles in contact with each one of them.

Table 1
Arithmetic operations for the contact check

	Additions and subtractions	Multiplications	Square root
Number of arithmetic operations	5	2	1

Table 2
Arithmetic operations for the force update function

	Additions and subtractions	Multiplications	Divisions	Square roots
Number of arithmetic operations	20	18	2	1

Table 3
Arithmetic operations for the position update function

	Additions and subtractions	Multiplications	Divisions
Number of arithmetic operations	8	12	3

Table 2 gives an overview of the total number of arithmetic operations needed to compute the force between two balls in contact.

2.3. Velocity and co-ordinate update

Once the resultant forces of each ball are calculated by summing the forces of all contacts in vectorial form for every ball, these forces can be used to find the new accelerations using Newton's second law. This requires only $O(N)$ operations and this is the fastest of all three calculations within one time step to be performed.

The number of arithmetic operations involved is given in Table 3.

3. Field Programmable Gate Arrays

The complexity of Field Programmable Gate Arrays (FPGAs) is continuously increasing, and, state of the art FPGAs now have up to 10 million system gates, with up to 10 Mbit of embedded RAM. One promising application area for these devices is to form FPGA-based reconfigurable co-processors within standard computers, which can be used for algorithm acceleration [8,9]. For the right type of application, such a reconfigurable computer can rival the expensive parallel computers that are normally used to accelerate computationally expensive algorithms.

The DEM has properties that may be suitable for acceleration using FPGAs:

- It exhibits an enormous degree of parallelism
- It uses very simple arithmetic operations

4. Hardware implementation of the DEM on a reconfigurable computing platform

4.1. Reconfigurable computing platform

The reconfigurable computing platform used in this current work was a PC reconfigurable computing PCI plug-in card. The card was a Celoxica RC1000-PP PCI card containing a single Xilinx Virtex 2000E-6 FPGA with 4 banks of 2 Mbytes of RAM. The RC1000-PP hardware platform is a standard PCI bus card. It has 8 Mb of SRAM directly connected to the FPGA in four 32-bit wide memory banks. The memory is also visible to the host CPU across the PCI bus as if it were normal memory. Each of the 4 banks may be granted to either the host CPU or the FPGA at any one time. Data can therefore be shared between the FPGA and host CPU by placing it in the SRAM on the board. It is then accessible to the FPGA directly and to the host CPU either by DMA transfers across the PCI bus or simply as a virtual address. The board is equipped with two industry standard PMC connectors for directly connecting other processors and I/O devices to the FPGA; a PCI–PCI bridge chip also connects these interfaces to the host PCI bus, thereby protecting the available bandwidth from the PMC to the FPGA from host PCI bus traffic.

A more detailed view of the RC1000-PP architecture is shown in Fig. 1 block diagram.

4.2. System description (software and hardware HW partition)

The first task once the simulation is started (for the software and the hardware simulator) is that the software reads an initialisation file where the system data is stored. It then generates the requested particles and once it finishes it waits for the simulation to run. This initialisation section is performed by the program for both the software and the hardware implementation. Fig. 2 shows a system layout.

Once the particles are generated the user has the option to choose if the user wants to run the simulation in software or hardware. The user can decide to run the simulation in software i.e. on the PC's microprocessor or run it on hardware i.e. on the reconfigurable computing PCI board.

4.3. Hardware implementation

In order to exploit the parallelism of the DEM the four major tasks (contact checking, forces update,

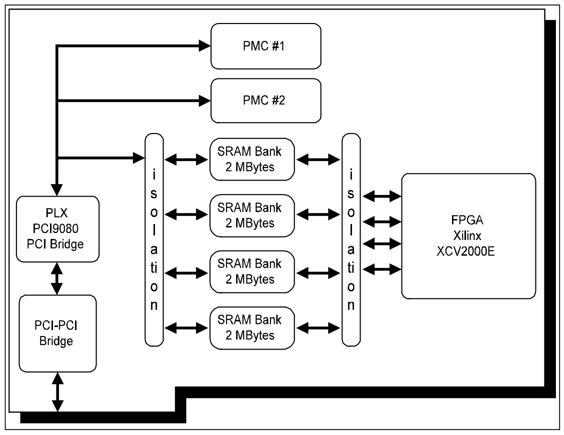

Fig. 1. RC1000 block diagram.

position update and reboxing) need to be performed in parallel.

Fig. 3 shows the internal structure of the design. It consists of six main units:

- A *control unit*, which synchronises all the units and generates all the control and address signals.
- A *contact check unit*, which identifies the particles in contact.
- A *force update unit*, which updates the interparticle forces.
- A *movement update unit*, which calculates the particles' new velocities and coordinates.
- *An interface unit* to read/write data to and from the external memory.
- *A write back unit* to write the results of the arithmetic units back to the internal FPGA memory.

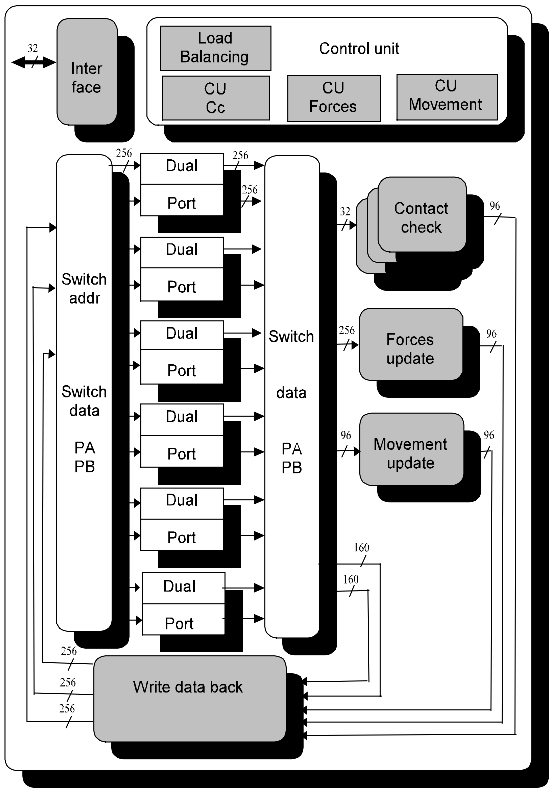

Fig. 3. FPGA internal block diagram.

The embedded FPGA memory (block RAM) is used to hold the data required to describe each particle. This includes position, velocity, rate of angular rotation, identity of neighbours, and the force that it is experiencing.

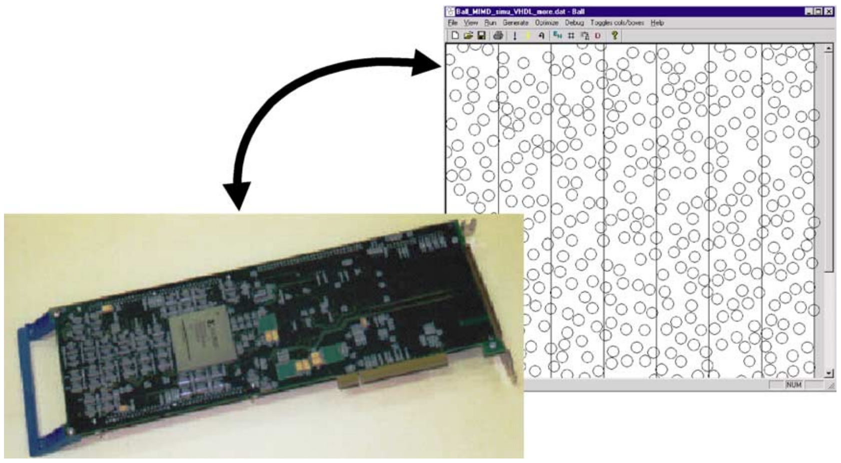

Fig. 2. System diagram.

Each unit of this implementation will be described into detail in the following subsection.

4.3.1. Control unit

The control units generate the necessary control signals to synchronise data between the blocks, and to steer the data output from the RAMs through the switch array to the inputs of the appropriate computation unit. The control units also generate the addresses to read and write data back to the internal and external memory.

4.3.2. Contact check unit

For each particle, a "contact list" is formed, which contains references to each of the particles with which it makes contact. In order to detect if two particles are in contact Eq. (1) has to be solved. If the condition of Eq. (1) is true, the addresses of the two particles are added to each others' adjacency list. For this investigation, all particles are assumed to having the same radius R. Under this circumstance, simple geometry shows that for a 2-D simulation, the maximum number of contacts that each ball can have is 6. This means that contact information can be represented by a very simple data structure, in which each particle has six memory slots allocated to hold the identities of the particles potentially in contact.

If there are N particles within a region of the DEM, then the number of contact checks that must be performed is N^2. The square roots and multiplications used in Eq. (1) are very expensive to perform in FPGA hardware, with the implication that a full contact check would be prohibitively expensive.

Instead of checking for true contacts, it was decided to check which particles are within each others' bounding boxes and this acts as a filter before the actual contact check. Under some circumstances (see Fig. 4), this means that a pair of particles will be classified as neighbours even though they are not truly in contact. This causes no real problem, since it is detected and correctly handled by the force increment unit.

With the use of a bounding box check as a filter makes the contact check unit very cheap in terms of hardware utilisation, as it requires only two additions, two subtractions and four comparisons.

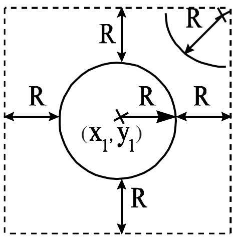

Fig. 4. Neighbour check model.

4.3.3. Inter-particle forces increment

Once the contact list for a particle has been established the total force acting on it can be determined. This will require a full solution of Eq. (1) for each contact identified, but this will be needed to be performed only a maximum of $6N$ times.

For every contact identified between two particles, the resulting force is calculated. For this study, a simple force–displacement law is adopted therefore the resulting force between two balls is directly proportional to the indentation between the balls.

The resultant force on a particle is the vector sum of the forces caused by each contact with its neighbours. The force update unit, which does require the computation, requires a large amount of hardware. It also operates at comparatively low clock speed of 7.5 MHz, because of the multipliers, in contrast to the contact check unit which works at the fully systems clock speed of 30 MHz (four times the clock speed of the forces and position update unit). Fig. 5 shows the internal structure of this unit, where each column represents one pipeline stage.

It can be seen that there are three main paths in this structure. One that calculates the forces in the x direction, the other that calculates the forces in the y direction and another shorter one, which computes the terms in Eq. (1), to check if the particles are in contact.

In order to compute the sine and cosine of the angle of the triangle formed by the union of the particles' centroids a Look Up Table (LUT) is used instead of having to compute them using a square root, which is very expensive in terms of HW resources (see Eqs. (2)–(4)).

$$\sin = \frac{y_2 - y_1}{d} \tag{2}$$

$$\cos = -\frac{x_2 - x_1}{d} \tag{3}$$

with

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4}$$

LUTs can be implemented easily in FPGAs and do take far less logic resources than a square rooter. Pre-defined values of the cosine and sine are stored in this table.

The sine and cosine are needed to decompose the forces from x and y direction to normal and tangential components of the contact.

4.3.4. Velocity and position update

Once the resultant force on each ball has been calculated, these forces are used to find new accelerations using Newton's second law. In this study, it is assumed that the masses of all the balls are identical. These

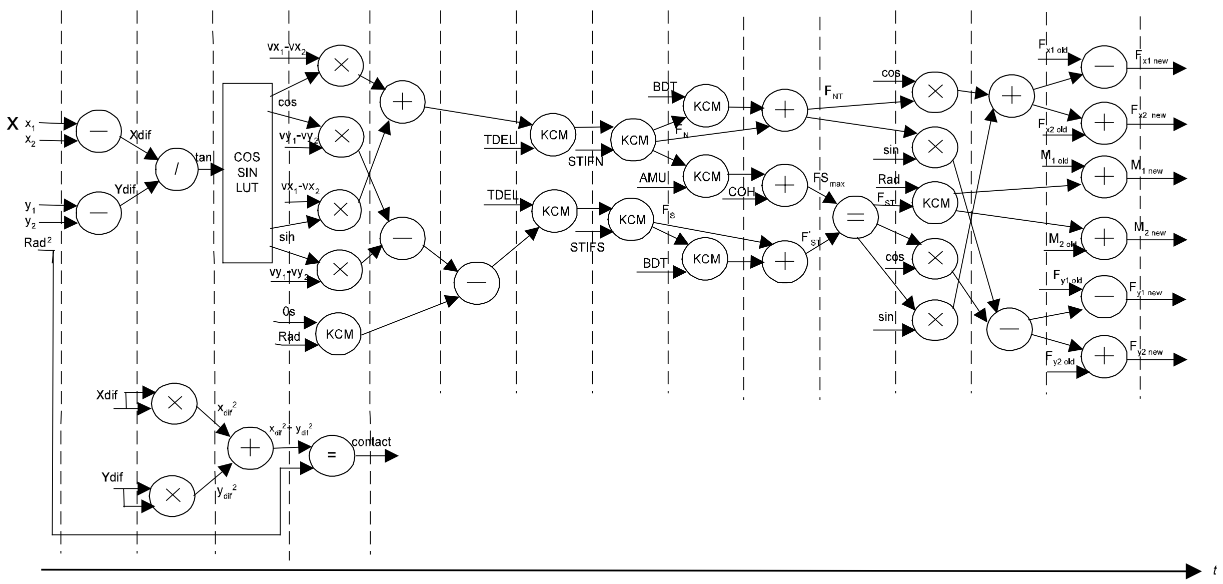

Fig. 5. Forces update unit internal structure.

accelerations are integrated to obtain the velocities in the x and y direction and the angular velocity.

The new coordinates can be found by adding the original coordinates to the incremental displacement obtained by integrating the calculated velocities. This unit has an intermediate level of hardware complexity, and operates at the same speed as the force update unit (7.5 MHz), which is four times slower than the system clock in order to have its pipeline fully loaded, achieving one new result per clock cycle.

It consists of three pipelines in parallel (see Fig. 6) in parallel. In the first one x and v_x are computed, y and v_y are calculated in the second and θ and θ^* in the third one.

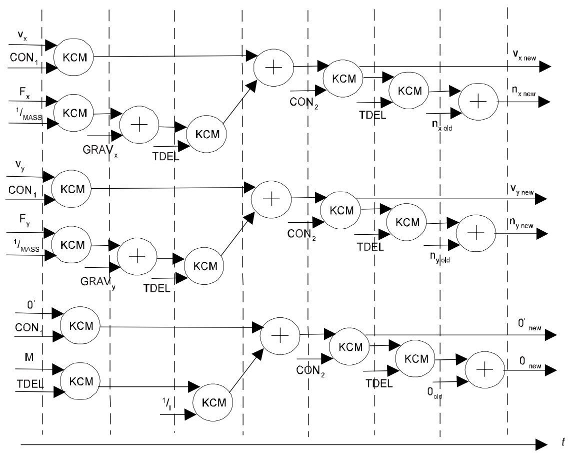

Fig. 6. Position update unit internal structure.

4.3.5. Write back unit

The task of this unit is to merge particle data produced by the computational units, and to write it back to the appropriate locations in the internal memory.

4.3.6. Interface unit

The interface unit handles communication between the FPGA's internal memory and the external memory on the board. The data of one subdomain is written back to the external memory once the position update unit has finished and new data for the next sudomain is read into the FPGA memory.

4.3.7. Hardware requirements

The hardware requirements for each of the main functional units are shown in Table 4. Constant coefficient multipliers (KCMs) require much less hardware resource than multipliers that allow both inputs to vary. Having balls of the same radius gives large savings in hardware resource, since all the multiplications involving the radius can be achieved by KCMs rather than full multipliers.

The contact checking unit is very simple, requiring little hardware resource, and capable of operation at high clock speeds. The force update unit, which does require the computation of the terms in Fig. 5, requires a large amount of hardware. The movement update unit has an intermediate level of hardware complexity.

4.3.8. Working mechanism

In order to allow the computational units to operate in parallel, the domain is decomposed into X vertical

Table 4
Hardware requirements

Contact checking	Force update	Movement update
2 adders	20 adders	8 adders
2 subtractions	10 multiplications	15 KCM
	8 KCM	
	1 divider	
	1 Look Up Table (LUT)	

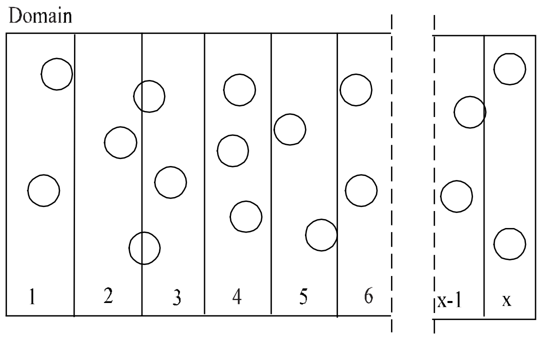

Fig. 7. Domain decomposition.

columnar sub-domains (cells), as shown in Fig. 7. Each particle belongs to a particular cell, and for most particles contact checking and force updating need only be performed against the other particles within the same cell. For the small number of particles that are close to the boundary between two cells, more complicated arrangements are necessary.

The architecture divides the internal block RAM of the FPGA into six dual port RAMs. At any given time, six of the columnar cells shown in Fig. 7 are stored within the dual port RAMs shown in Fig. 3, and undergo processing. The RAM contains two 256 bit entries for each particle within that cell consisting of 16 bit entries for x, y, q, v_x, v_y, q', F_x, F_y, M, a type flag, and the reference of up to six neighbouring particles and another to hold the normal and shear forces for every contact (with a maximum of six balls in contact).

Due to complexities associated with handling particles close to the cell boundaries, the contact check unit may have to update the columns to the left and the right of the column that is currently undergoing contact check, as the contact check unit also deals with particles that have transitioned from one column to another. It deletes the particles, which have moved from the column where the contact check is taking place and moves them either to the right or left column, depending on where the particle has moved. Also, the force update unit may have to interact with the column to the right of the column currently undergoing force update, as a particle in this column might be in contact with particles in the neighbouring column. That is why the FPGA must hold six columns at any given time, rather than three.

4.3.9. Adaptive cell boundaries

A complication appears as simulation progresses. Particles will move between columns, and some columns may become heavily populated, whilst others are sparsely populated. It is then necessary to move the cell boundaries, thus expanding some cells and contracting others. This is needed in order to provide good load balancing, and also to prevent overflow of the block RAMs.

Movement of cell boundaries is fairly simple. The control unit monitors how many particles are held in each block RAM. When the number falls below a minimum threshold or rises above a maximum, the boundary is moved by a distance R so as to expand or contract the cell. When the boundary moves, a number of cells will find that their data is stored in the wrong column of RAM, but this will be automatically detected and corrected by the mechanisms described earlier for handling particles close to boundaries.

Using the procedures described above, the transition of particles from one cell to another is handled without causing any loss of performance. Also, the cell size is adaptively optimised so that good load balancing is always achieved.

The XCV2000E can only hold a maximum of 128 particles per column, because part of the FPGA's embedded RAM has to be allocated to other functions to make the design fit. If a sub-domain were to acquire more than 128 particles, the excess particles would be discarded, leading to the loss of particles from the system. This is avoided by dynamically balancing the load in each sub-domain so that no more than a certain maximum number (always smaller than 128) will be in each sub-domain. The software program will also issue a warning signal if the danger of having more than 128 particles in a sub-domain exists, for example when the domain height compared to the balls radii is so large that more than 128 balls would fit in one column.

With contact checking, force updating and co-ordinate updating being performed in parallel, load balancing problems will inevitably appear, since the overall system speed will be limited by the speed of the slowest of the three units. As shown in Section 2, where the DEM was described in detail, the co-ordinate check is the most time consuming, but requires very simple hardware and can operate at high clock speed.

In order to improve the load balance, more than one contact check units are instantiated, and operate in parallel. The number of contact checks to be used is a parameter of the design, which can be easily changed. The contact check control unit can generate all the required control signals to steer the data correctly between the different check units. Lastly, the contact check units can run at four times the clock speed (30 MHz) of the force update unit and co-ordinate update unit (7.5 MHz), because it is very simple, requiring little hardware resource.

It can also be seen that the coordinate update unit will finish much earlier than the forces update unit. The spare time available at the end of the coordinate update is used to write the data from the block of RAM corresponding to co-ordinate update that has now finished being processed for this time step, into external RAM. A new set of data is also read from external RAM, which corresponds to the next column of the domain that is to be processed. The ideal timing schedule is show in Fig. 8.

In this way, writing to and reading from external RAM can be fully overlapped with computation, and the number of particles that can be processed at full speed is limited only by the size of the external RAM. This means that problems containing tens of millions of particles can be processed easily.

In order to have the system running at its maximal efficiency there must be as many contact check units as needed to make the time for position update and data update (t(position) + t(interface)) equal to the time for contact checking t(cc). Fig. 9 illustrates the number of contact check units needed, using theoretical calculations, for the above condition to be true.

The straight line for t(position) + t(interface) is the number of clock cycles required to perform position update plus the time needed to write and read new data to and from the external memory, for all particles in one column. The curves t(cc) show the number of clock cycles needed to perform contact checking, for varying numbers of contact check units (ccu). Where the t(cc) curve intersects with the t(position) + t(interface) line, this indicates the ideal load balancing for that number of particles. So, for example, a simulation of almost 175 particles/column has almost ideal theoretical load balancing when five contact check units are instantiated.

The time needed to perform the contact check depends not only on the number of particles per column, but also on the domain topology and the size of the particles. The larger the height of the domain Y is, the larger the contact area is. Also the smaller the radius of the particles is, the more particles need to be checked for contacts with the neighbouring column. Fig. 9 is given for a particular value of Y/d with Y being domain height and d balls' diameter.

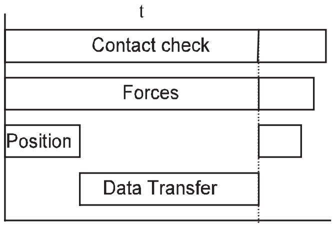

Fig. 8. Task scheduling.

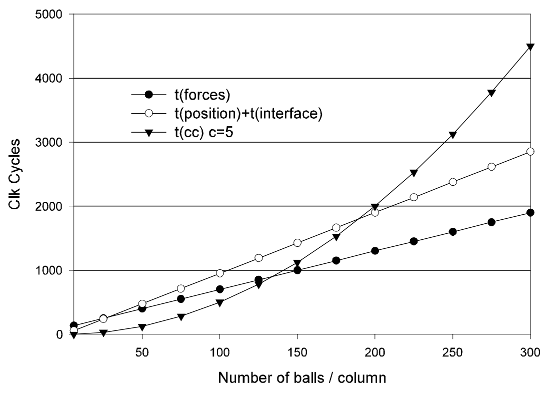

Fig. 9. Timing for each unit.

4.3.10. Validation of the design

In order to validate the hardware implementations a debugger was incorporated into the software environment in order to compare the behaviour of the software and the hardware design. The debugger is capable of running software and hardware simulations simultaneously, and displaying both sets of results on screen for the current simulation cycle. The debugger also computes and displays various measures of the difference between both sets of results, e.g. the difference between particle coordinates, the average velocity difference and the difference in system centroid.

Due to the difference between the 16 bit fixed point data format of the hardware design and the 32 bit floating point format of the software, the motions of individual particles differ slightly. Rounding errors in the hardware data format make the particles move slightly slower than in the software version. However, the overall behaviour of the assembly remains unaltered. Since users are usually interested in the behaviour of the bulk, and not of the individual particles, for most practical purposes the accuracy of the hardware simulation can be regarded as being as good as that of the software simulation.

5. Software and hardware comparison

There are many ways to evaluate the performance of a parallel algorithm, and a very common way is to compare the runtime of the sequential algorithm and the same implemented on a parallel machine, optimised as much as possible for that parallel machine

Table 5
Run time comparison for hardware (HW) and software (SW) implementations

No. of particles	50 000	75 000	100 000	125 000	150 000	175 000	200 000
TSW (s)	1800	2485	3120	3920	5156	6123	7423
THW (s)	51	80	103	130	175	196	245
Speedup	35.3	31.0	29.8	30.2	29.5	31.2	30.3

$$speedup = \frac{Runtime\ of\ the\ fastest\ sequential\ algorithm}{Runtime\ of\ the\ parallel\ algorithm} \tag{5}$$

The runtime of an optimised software simulator will be compared with the hardware implementation in the next section.

5.1. Simulations

An experiment was set up in order to measure the effectiveness of the hardware design. Domains with 50,000, 75,000, 100,000, 125,000, 150,000, 175,000, and 200,000 particles were generated and simulated for 1000 time steps.

The performance of the software version was measured and compared with the results obtained by the hardware version. The software version used an optimal domain decomposition in order to minimise its simulation time. The initial velocities of the balls were initialised pointing towards the centre of the domain.

Table 5 shows a comparison between the speedup achieved by the hardware simulation run at 30/7.5 MHz (30 MHz for the contact check and interfacing with the external memory and 7.5 MHz for the forces and position update units) as compared to the software.

Fig. 10 shows graphically the achieved speedup.

Section 4.3.9 showed that for a maximum of 125 balls/column (limited by the FPGA's internal RAM) the position update and the read and write operation of new data to the external memory was the bottleneck of the design, as it takes longer than the time required by contact check.

The maximum number of balls that can be stored per column in the FPGA being only 125 has no influence in the total execution time as the only deep pipelines are in the force update unit, which is not time critical.

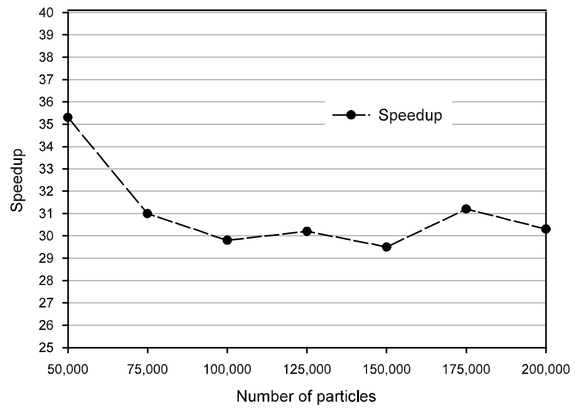

Fig. 10. Speedup measurements.

5.2. Discussion of the results

The hardware implementation has accelerated the simulation by a factor of 30. This shows that the hardware implementations makes full use of the intrinsic parallelism of the DEM, as it runs at a comparatively slow clock speed 30/7.5 MHz, compared to the PC microprocessor.

The hardware implementation will always take the same time to simulate a system with the same number of particles, since it works as fixed-cycles state machines, thus the computation time will grow linearly with the number of particles in the system. The software implementation run time, on the other hand, will depend on other factors. The most important is probably the stiffness. The lower the stiffness the more contacts are generated and the more often the resultant forces need to be computed for each particle. If the stiffness is large enough the system will behave like a billiard table, thus contacts will happen only during a very short time in each time step. Therefore the stiffness selected in the simulations will affect the speedup that hardware achieves over software. For high stiffness values, the software runtime is low, so the speedup achieved is reduced. In order to provide a conservative estimate of the speedup achievable by the hardware, the stiffness value used in the simulations of Fig. 10 was set to the maximum allowed by the 16-bit data arithmetic used. Selecting a lower stiffness value would give even better speedup results.

6. Scalability of the hardware implementation

A hardware implementation of the 2-D DEM on one single FPGA has shown a speedup of a factor of 30. This might be enough for some simulations, but in the case of entire engineering structures, millions of particles are involved. In order to model these structures an even higher speedup is needed. The only way to achieve speedups of some orders of magnitude bigger than what has been achieved is to have multiple FPGAs working in parallel.

A very important aspect of the hardware implementation presented in this paper is how well it will scale on a multiple FPGA system. Will the speedup be linear with the number of FPGAs or will communication, synchronisation overheads and load balancing problems degrade the overall FPGA speedup considerably as in the multiprocessor systems?

The term scalability tries to express the benefit of solving large problems on a multiple processing element system. The algorithm is scalable if the efficiency is more or less constant, where the efficiency is defined as the speedup divided by the number of processing units.

This section will describe a multiple FPGA implementation based on two RC1000 boards connected in parallel. The domain decomposition method used facilitates the spreading of the simulation across multiple FPGA boards with minimal communications overhead, which means that near linear speedup can be achieved. It will be shown that this implementation allows the full overlap of computation and communication between boards.

6.1. System description

Fig. 11 shows a diagram of a distributed memory system containing N RC1000 boards. Each board contains an FPGA, whose block RAM is organised as six dual-port RAMs, each of which is to be used to contain the data for a sub-domain. This data could be swapped in and out of four banks of static RAM present on each RC1000 board. The boards communicate with one another across the PCI bus. As long as the amount of data being transferred across the PCI bus between the boards remains small, linear speedup can be expected as more FPGA boards are added.

Initially the domain is split across the boards so as to equalise the workloads. After one time step is completed, each board needs to exchange its rightmost column including the data structures that catch particles transitioning across sub-domain boundaries, with its right hand neighbour. Similarly each board must exchange its leftmost column with its left hand neighbour. If this transfer can be completely overlapped with computation, then none of the computational pipelines on the FPGAs ever need stall, and speedup should be linear, i.e. use of N boards should provide N times speedup in comparison to a single board.

The data for each particle in two dimensions consists of 34 bytes with 2 bytes each for x, y, v_x, v_y, plus the normal and shear force at each contact, which for the worst case are another 12 items for particles of the same radius. The number of particles in a column is limited to 128 in order to avoid overflow of the block RAM. The load balancing method introduced in Section 4.3.9 will adaptively reduce the size of any domain that has its number of particles approaching this limit. So for N boards, the maximum amount of data to be transferred across the PCI bus for each time step of the DEM method is $34 * 128 * 4 * N$ bytes $= 1741N$ KBytes. The factor of 4 in the previous calculation arises because two columns (left and right-most column) have to be read from each FPGA and two have to be written in the FPGA. Eq. (6) gives the general expression for the amount of data that must be transferred between two boards for each time step.

$$Data_{transferred} = nr_{of\ parameters} \times 2Bytes \times max\ balls\ col \times 4 \times nr\ boards \tag{6}$$

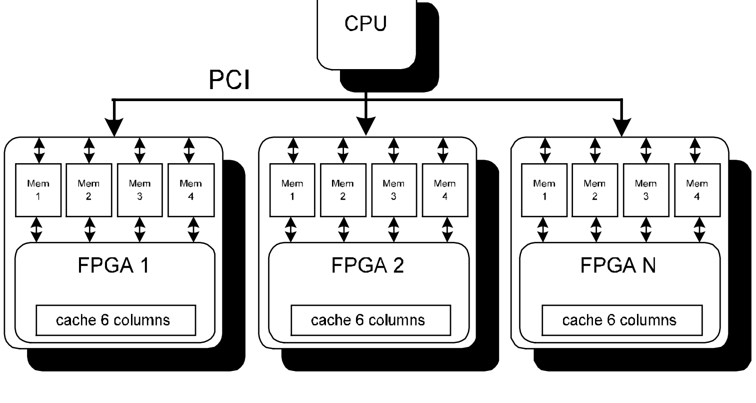

Fig. 11. Multi-FPGA system.

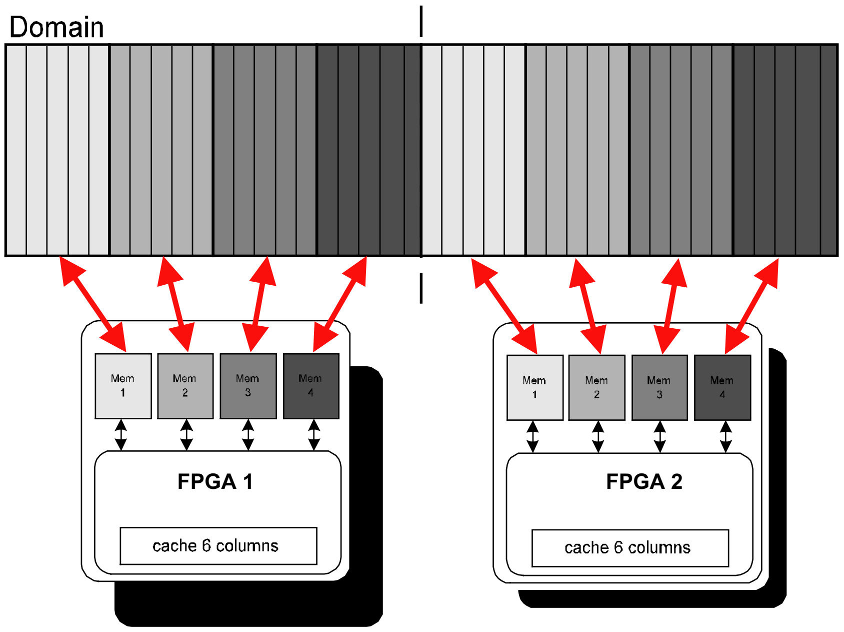

Fig. 12. Multi-FPGA system.

Within each board, the FPGA uses one RAM bank at a time. Transfer of edge data to an adjacent board can be initiated when a bank of memory is released by the FPGA. The transfer must be completed before the FPGA attempts to re-acquire that bank, which occurs after it has finished processing the contents of the other three RAM banks on the board (see Fig. 12). This amounts to a period of time as shown in Eq. (7), where $t_{1column}$ is the time needed to compute 1 sub-domain. The slowest operation in the processing of a sub-domain containing M particles is the position update and data transfer. The time $t_{pos\ data\ transfer}$, taken for this operation for 1 time step is shown in Eq. (8), where f is the frequency at which the FPGA works.

The number of sub-domains contained in each RAM is shown in Eq. (9) for the 2 MByte memory available on the current FPGAs.

$$t_{data\ transfer} = 3 \times t_{1column} \times nr_{cols/RAM} \tag{7}$$

$$t_{pos\ data\ transfer(1cycle)} = 6M + \frac{17 \times M}{2} \times \frac{1}{f\ [\mathrm{Hz}]} \tag{8}$$

$$Nr_{cells/mem\ unit} = \frac{2\ Mbytes}{Max\ nr\ balls_{1col} \times Ball\ parameters \times 2Bytes} \tag{9}$$

The boards are capable of sustaining DMA transfers across the PCI bus at about 12 Mbytes, which means that saturation of the bus will not occur for a number of boards N below about 282. This means that if ideal load balancing is achieved then speedup can be expected to be linear for a number of boards up to that number.

6.2. Simulation results

The average speedup for a single FPGA system was around 30 in comparison to the optimised software version running on a Pentium III processor with 1.3 Gbytes of RAM. It is therefore reasonable to expect a speedup of around 60 with a two-board system, as communication overheads should not influence the computing time, as computation and communication are completely overlapped.

The hardware simulation for a system with two boards gave a result slightly worse than the expected linear speedup of 60, as shown in Table 6. This is due to

Table 6
Run time comparison for hardware and software implementations

No. of particles	50 000	75 000	125 000	125 000	150 s000
Speedup (1 board)	35.3	31.0	29.8	30.2	29.5
Speedup (2 boards)	54.0	55.2	54.7	53.7	54.9

the synchronisation of the FPGAs needed after every cycle. The fastest FPGA needs to wait for the slower ones to complete their computation; therefore it is impossible to have a completely balanced system.

7. Conclusions

A completely new approach to the acceleration of the DEM computation has been presented in this paper. Previous approaches were based on multiprocessor systems. Their speedups were far less than linear, because of communication and synchronisation overheads as well as load balancing problems. This novel approach exploits the intrinsic low and high-level parallelism of the DEM by scheduling the arithmetic operations in parallel and by decomposing the domain so that the four main tasks: contact checking, forces, position updating and re-boxing, can be performed at the same time. A speedup of a factor of 30 for a single FPGA, compared to an optimised software version running on a fast PC, has been demonstrated and of almost 60 for a multi-FPGA system based on two RC1000 boards connected in parallel via the PCI bus, allowing communication and computations to fully overlap.

References

[1] Hustrulid AI. Parallel implementation of the discrete element method. Available from: <http://egweb.mines.edu/dem/>.

[2] Liebling TM, Ferrez JA, Mueller D. Parallel implementations of the distinct element method for granular media simulation on the cray t3d. EPFL Supercomputing review, 8 November 1996.

[3] Clearly PW, Sawley ML. A parallel discrete element method for industrial granular flow simulations. EPLF Supercomputing review, 11 November 1999.

[4] Belytschko TB, Dowding CH, Dymytryshyn O. Parallel processing for a discrete element program. Comput Geotech 1999;25:281–5.

[5] Cundall PA. A computer model for simulation progressive, large scale movements in blocky rock system. In: Proc Symp Int Rock Mechanics, Nancy, France, Paper II-8, 1971, p. 129–36.

[6] Strack ODL, Cundall PA. A discrete numerical model for granular assemblies. Géotechnique 1979;29(1):47–65.

[7] Hertz H. Uber die berhrungfester elastischer korper. J Renie Angew Math 1992;92:156–71.

[8] Becker J, Hartenstein RW. High-performance computing using a reconfigurable accelerator. CPE Journal, Special Issue of Concurrency: Practice and Experience; 1996.

[9] Hartenstein RW, Herz M. On reconfigurable co-processing units. In: Proceedings of the Reconfigurable Architectures Workshop (RAW98), March 1998.