Comp. Part. Mech. (2014) 1:307–319
DOI 10.1007/s40571-014-0026-3

A framework for grand scale parallelization of the combined finite discrete element method in 2d

Z. Lei · E. Rougier · E. E. Knight · A. Munjiza

Received: 24 January 2014 / Revised: 15 May 2014 / Accepted: 21 May 2014 / Published online: 7 June 2014
© OWZ 2014

Abstract Within the context of rock mechanics, the Combined Finite-Discrete Element Method (FDEM) has been applied to many complex industrial problems such as block caving, deep mining techniques (tunneling, pillar strength, etc.), rock blasting, seismic wave propagation, packing problems, dam stability, rock slope stability, rock mass strength characterization problems, etc. The reality is that most of these were accomplished in a 2D and/or single processor realm. In this work a hardware independent FDEM parallelization framework has been developed using the Virtual Parallel Machine for FDEM, (V-FDEM). With V-FDEM, a parallel FDEM software can be adapted to different parallel architecture systems ranging from just a few to thousands of cores.

Keywords Parallelization · FDEM · Virtual engine · Combined finite discrete element · Fracture

1 The combined finite-discrete element method

The Combined Finite-Discrete Element Method (FDEM) merges the finite element based analysis of continua with discrete element based transient dynamics, contact detection, and contact interaction solutions. In FDEM the solid domains (called discrete elements) are discretized into finite elements, where finite rotations and finite displacements are assumed a priori and are formulated using a multiplicative decomposition based finite strain formulation.

Z. Lei · E. Rougier (✉) · E. E. Knight
Geophysics Group, Los Alamos National Laboratory,
Los Alamos, NM 87545, USA
e-mail: erougier@lanl.gov

A. Munjiza
Department of Engineering, Queen Mary, University of London,
London E1 4NS, UK

Through failure, fracture and fragmentation, single domains represented by separate finite element meshes are transformed into a number of interacting domains. The finite element discretization of solid domains is also conveniently used to discretize the contact between discrete elements. Utilizing this approach, discretized contact solutions can then be used for both contact detection and contact interaction, Fig. 1.

MRCK Contact Detection in 2D. To improve FDEM contact detection performance, the MRCK (Munjiza-Rougier-Carney-Knight) contact detection algorithm was developed to replace the widely used NBS and MR (Munjiza-Rougier) contact detection algorithms [1,2] which are utilized in other FDEM codes such as Y2D and Y3D [3,4]. MRCK is based on the decomposition of the physical space (as opposed to the solid domain) into identical square cells.

Within the FDEM framework, for any two given particles or elements, one called contactor and the other one target, the contact search problem is reduced to determine whether the contactor and the target share at least one square cell. Targets are mapped onto these square cells according to their current position. Targets are then sorted according to the cell to which they are mapped onto, Fig. 2. For this operation the MR-Linear sort algorithm is used, which has a total sorting time proportional to the total number of targets. This results in much better performance than the Quick-Sort algorithm which is part of the NBS and MR contact detection [5]. Finally, the sorted list is searched for contacts.

In principle, the MRCK contact detection algorithm can use any type of target but the most often used targets are simply points. MR-Linear sort takes advantage of the fact that no contact target can move more than the size of a single cell in a single time step. The list of contact targets is therefore parsed only once, starting from the head of the double con-

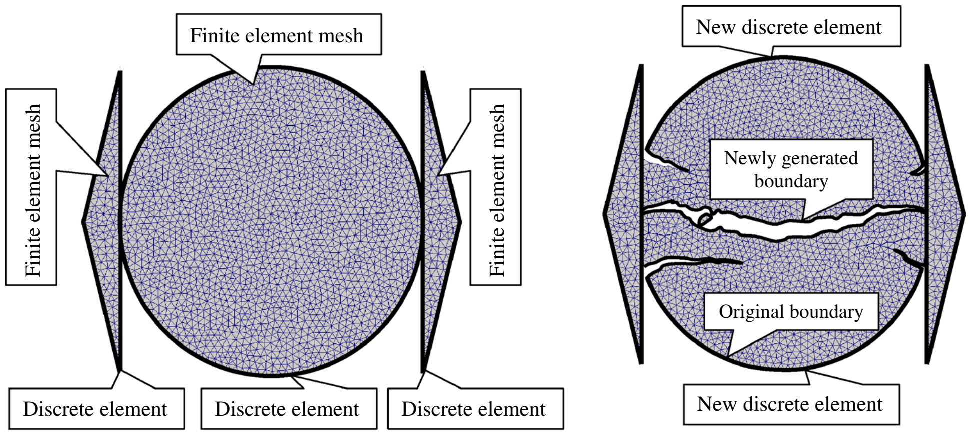

Fig. 1 A typical FDEM simulation showing creation of new discrete elements

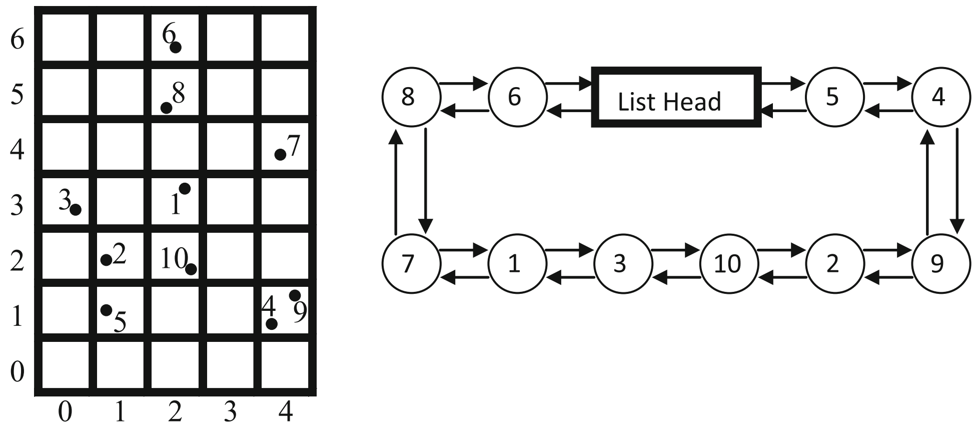

Fig. 2 An example of a system of target points sorted using the MR-Linear sort

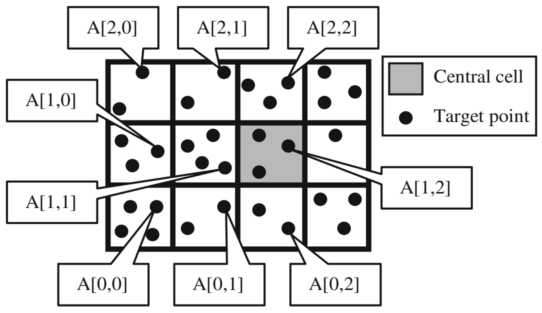

Fig. 3 The matrix **A** used to facilitate MR-Linear sort

nected list and targets are moved within the list as required. This is facilitated by the introduction of a matrix **A**, which is a 3 by 3 matrix of pointers. Each element of matrix **A** (pointer) points to the targets that are located immediately before each of the current target's neighboring cells, as shown in Fig. 3. As the list is parsed, all the pointers are advanced forward resulting in the theoretical CPU time for the MR-Linear sort to be proportional to the total number of targets.

Detection of contact can be done for any contactor shape. The simplest shape in 2D is either a 2D edge or a 2D triangle. For a given contactor, a process called "contactor rendering" is used to detect all the cells that the contactor currently occupies, Fig. 4. The rendering is done in conjunction with the sorted list of targets in such a manner that only the cells that have targets mapped to them are rendered. This speeds up the rendering process significantly, to the point that a contactor with no contacts is not rendered at all, while others may only have a couple of cells rendered.

Triangle to Point Contact Interaction in 2D. The discretized distributed contact force approach has been used in conjunction with FDEM since its inception. This approach enables many different implementations in terms of contact kinematics. Additionally, almost any type of contact physics can be superimposed on the basic contact kinematics. In the earliest versions of FDEM, a "triangle to triangle" contact interaction was used in 2D [3]. This "triangle to triangle" approach exactly considered the geometry of both the contactor and the target triangles and the integration of the contact forces distributed along the edges of the discrete elements (or finite elements) was done analytically. Since this approach integrated contact forces exactly, it was therefore quite time consuming.

Follow-on versions of FDEM employed a "triangle to edge" approach, which was much faster and worked very

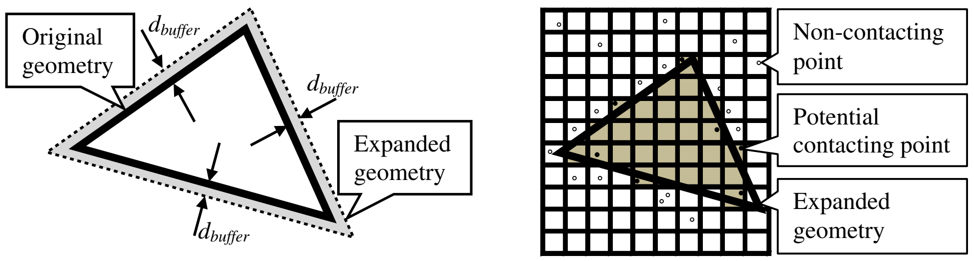

Fig. 4 Rendering of a contactor triangle into cells

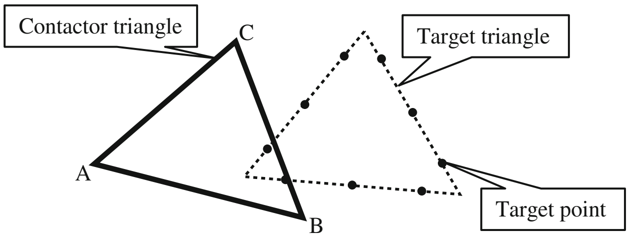

Fig. 5 Contactor triangle and contact force integration points (target points)

well with the MRCK contact detection. This approach also integrated contact forces exactly [4].

In the latest version of FDEM, the contact interaction in 2D has been further simplified by using "triangle to point" contact interaction kinematics. In this case, the contactor triangles also interact with target triangles. However, in order to integrate the distributed contact forces, target triangles are discretized into a series of points distributed on the free boundary lines of the discrete elements, as shown in Fig. 5.

Each target point is considered as a Gauss integration point through which the distributed interaction forces are integrated. The number of target points per triangle is selected according to the desired integration accuracy. It is worth noting that the original distributed potential contact force is still employed, but only in an approximate form.

The MRCK contact search algorithm detects contacting couples which consists of a contactor triangle and target point. The actual contact interaction is processed only if the target point is located inside the contactor triangle. Additional contact physics, such as rock joints, can be superimposed onto the core contact interaction.

Fracture and Fragmentation. In early versions of FDEM, a localization based strain softening approach was used to process fracture and fragmentation [6,7]. The problem with this approach is that it is non-objective in terms of both energy and size of the damage zone (localization band representing a discrete crack).

For more recent versions of FDEM, it is generally accepted that the combined single and smeared discrete crack approach [8] is a better solution in that it is objective in terms of both energy and size of the damage zone (crack). The combined single and smeared crack model is based on actual representations of experimental stress-strain curves. It divides the stress-strain curves into two parts: the strain-hardening part and the strain-softening part, Fig. 6. In the strain-hardening part, no failure occurs in the material and a standard continuum constitutive law is employed together with the incorporation of non-softening material nonlinearity, e.g., plasticity. The strain-softening part of the constitutive law is represented through stress being expressed as a function of displacement (not strain).

With this approach, the "strain localization" part of the constitutive law is introduced as material parameters into the simulation that will describe the width of the damage zone associated with the strain localization band and/or discrete cracks.

Failure of brittle materials, such as rocks, is well described by the above approach as confirmed by the numerous validation cases executed by different research teams [9–15]. However, failure of "ductile" materials, such as metals or plastics, are also well described using this approach. Metals are characterized by flow, Fig. 7. Once the metal's flow ability is exhausted, final failure occurs thereby resulting in a

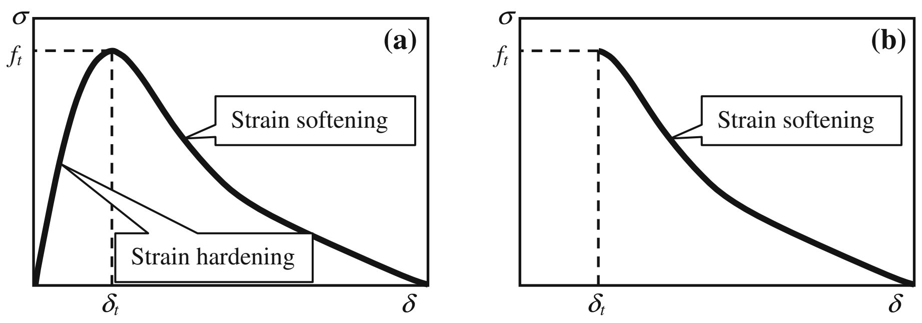

Fig. 6 **a** Stress-strain curve divided into the hardening and the softening part; **b** strain softening constitutive law defined in terms of displacements

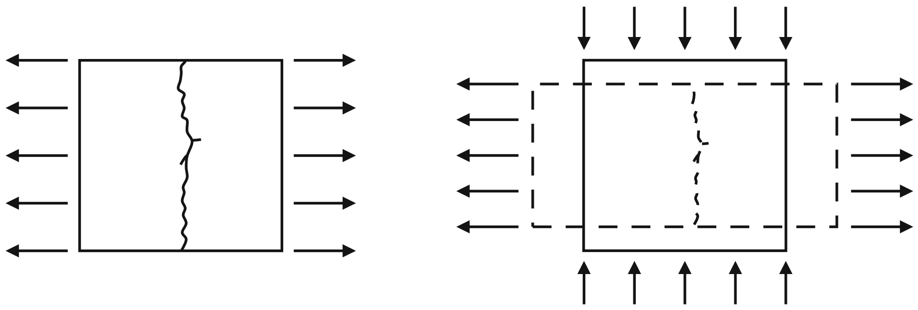

Fig. 7 *Left* Brittle damage (fracture). *Right* ductile damage (flow plus fracture)

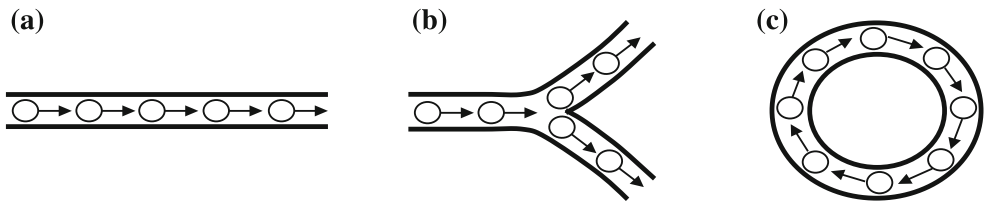

Fig. 8 Three engines of the sequential processor: **a** sequence engine, **b** branching engine and **c** loop engine

discrete crack. The two sets of material properties shown in Fig. 6 describe both processes well except that with ductile materials most of the physics is dominated by the flow, which may then be followed by damage, Fig. 7.

Along the damage zone or discrete crack, the local displacements are given by

$$\boldsymbol{\delta} = \delta_n \mathbf{n} + \delta_t \mathbf{t} \tag{1}$$

where $\mathbf{n}$ and $\mathbf{t}$ are the unit vectors describing the normal and tangential directions of the damage zone (localization band, discrete crack, etc.), while δ_n and δ_t are the respective displacement component magnitudes.

The stress traction vector $\mathbf{p}$ in the strain-softening stage is also divided into two components in the direction of $\mathbf{n}$ and $\mathbf{t}$, and is calculated from the stress tensor $\boldsymbol{\sigma}$,

$$\mathbf{p} = \boldsymbol{\sigma}\mathbf{n} + \boldsymbol{\sigma}\mathbf{t} \tag{2}$$

where the stress tensor $\boldsymbol{\sigma}$ within the damage zone is expressed in terms of displacements and can, in principle, accommodate any damage (softening) model.

Material Deformation Formulation. In this work, a large-displacements large-strains finite element formulation is used for the material deformation description. The details of this formulation are outside the scope of this paper, but they can be found in Munjiza [3].

2 Parallelization challenge

So far, FDEM, in its various implementations, has been applied to a large number of problems across different industrial scale applications. The best example of a successful commercial application is the ELFEN FDEM software package. The best examples of research based software are different implementations of the “Y” open source packages (Y2D [1,4], Y3D [1,4], YFDEM, Y-cgles[16], VGEST [17], Y-Geo [18], Y-nano [4]).

While most of the issues on sequential computers have been resolved, parallelization remains a challenge. A number of parallelization approaches have been proposed by different research groups [19,20].

In this work, yet another approach is being proposed to address the problem of hardware dependency on grand scale parallelization schemes. The tailor-made scheme developed and tested is called the Virtual Parallel Machine for FDEM or V-FDEM.

3 V-FDEM: a virtual parallel machine for fdem

Any sequential computer software can be made by combining a number of these:

1. sequences
2. branchings
3. loops

Thus, one can say that any sequential processor can be represented by a virtual processor that consists of three engines, Fig. 8. The sequence engine can be compared to driving straight down the road. The branching engine can be compared to a cross-roads where one can change the road on which to travel. The loop engine can be compared to driving in a circle, thus coming back repeatedly to the same point.

Using these three engines every single instruction of a sequential code can be reached in a similar manner that a car can reach any point on a road network by using “straight”, “turn” and “loop” commands.

The sequential CPU can be visualized as a single car being driven on an empty road network. In contrast, a parallel computer is like driving 30,000 cars on the same road network: first, there has to be some rules; second, the cars have to talk to each other; third, they have to synchronize to achieve their pre-set task. From this simple comparison, it is evident that the differences between sequential and parallel processors are huge, which is one of the reasons why there are so many different parallel architecture schemes.

Different grand challenge computing tasks in general require different hardware in order to achieve optimum performance. In this sense, FDEM problems are in a class of their own.

When solving parallelization, researchers usually start with a given hardware and develop their parallel solution for the hardware. For example, there were early solutions for the transputer based machine [21], the thinking machine (Massachusetts Institute of Technology) [22], the Silicon Graphics International (SGI) parallel machine [23], clusters, multicore machines, etc. The problem with this approach is obvious: How many different solutions does one develop? When developed, how long do they last? How much does it cost to keep pace with ever changing hardware configurations?

In this work, the authors have taken a different approach:

> Start with the specific problem and find the best way of solving it in parallel without any consideration to the hardware employed.

In this context, the FDEM problem in 2D is well defined: There are millions or billions of finite element meshes representing discontinuous solids that fracture, fragment and interact with each other or with fluid. The domain of computation is therefore an “empty” physical 2D space. As such the most intuitive way to move forward is to divide this physical space into (physical) subspaces. These subspaces are called “computational subspaces”. All the solids, fluids, etc. that are at a given instance—within a given subspace - are said to belong to this subspace.

Consider the subspace as a town with plenty of activity in it. Across the town’s boundaries, there are interactions with inhabitants of neighboring towns. In addition, the inhabitants move all the time from one town to another town. The town is the subspace. The inhabitants are solids, fluids, finite elements, discrete elements, nodes, etc. At every instance (of say time), every one of these inhabitants has a well-defined location in space, which identifies them as belonging to a single particular subspace.

Any parallel computational framework aimed at the above problem has to be able to:

(a) Solve interaction (contact), deformation, fracture, fluid interaction etc., between all entities (solids, fluids, etc.) within a given single computational subspace. This must be done with an existing sequential FDEM software without any modifications to it.
(b) Solve interaction between solids across computational subspace boundaries, including the migration of entities from one computational subspace to another.
(c) Evaluate cumulative quantities, such as the total mass, of all solids in all computational domains. The most naive approach would be for each subspace to “phone” its total mass to a master subspace. The master subspace would then add (accumulate) these masses and would obtain the total mass of the system.
(d) Send entities (objects) from a defined external space to all other subspaces. This is like the master “phoning” each of the subspaces in turn. For one million computational subspaces, this would mean a sequence of one million phone calls. One can easily visualize a long queue of subspaces waiting for the master to phone them and therefore, in the process, staying idle.

The above computational framework is called the “Virtual FDEM Parallel Machine”, or V-FDEM. As stated earlier, a general sequential software code can execute sequence, branching, and looping processes. In a similar manner, the V-FDEM can execute:

(a) Standard sequential FDEM code within a given computational subspace. In follow-on discussions this is referred to as the Virtual Space Engine (VSE).
(b) Communication (phoning) between any two objects from neighboring computational subspaces: hereby referred to as the Virtual Boundary Engine (VBOE).
(c) Simultaneous, gathering of data from any number of computational subspaces: hereby referred to as the Virtual Gathering Engine (VGE).
(d) Simultaneous, broadcasting of data to any number of computational subspaces: hereby referred to as the Virtual Broadcasting Engine (VBRE).

By combining VSE, VBOE, VGE and VBRE, it is possible to solve any FDEM problem using any number of computational subspaces.

4 V-FDEM Engines

Virtual Space Engine VSE. The Virtual Space Engine is the simplest engine of V-FDEM. By definition, it is an existing

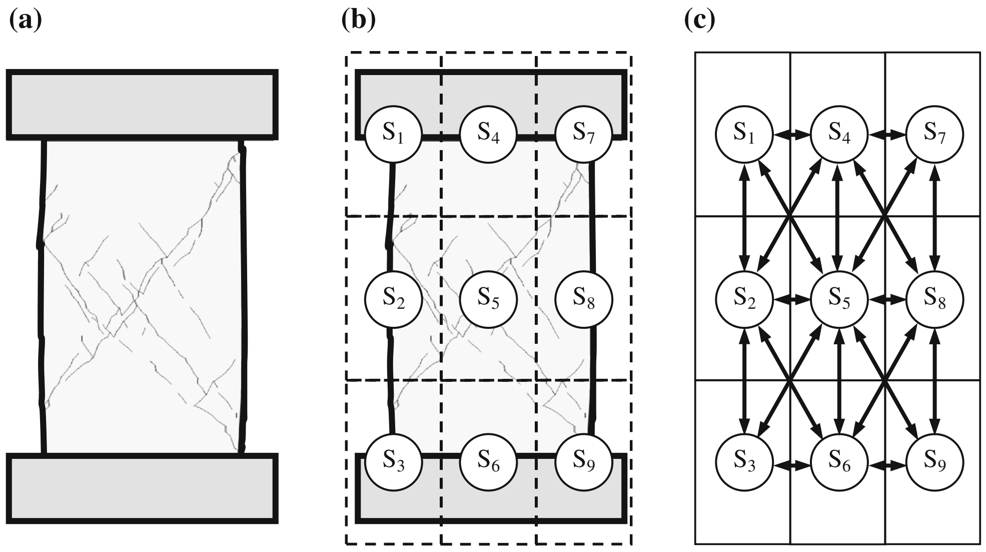

Fig. 9 An example of "talking" couples of computational subspaces. **a** The simulation model, **b** the physic space is divided into computational subspaces, and **c** the "talking" couples

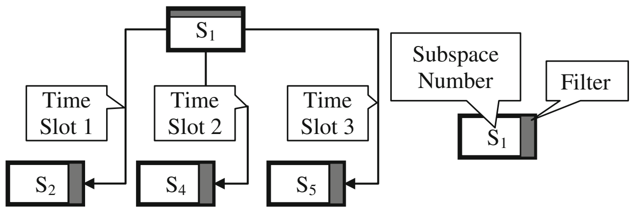

Fig. 10 Virtual Boundary Engine principle: time slots for talking couples of subspace S_1; *shaded areas* indicate data filtering

sequential FDEM code. Details of how a sequential FDEM code is designed and implemented are outside the scope of this paper, but can be found in Munjiza [3] or Munjiza et al. [4].

Virtual Boundary Engine VBOE. Because of "communication overheads," in order to make a parallel FDEM simulation viable, computational subspaces have to be of relatively large size. As a consequence, it is convenient to assume that only neighboring domains can talk to each other, Fig. 9c.

The VBOE schedules communications for each subspace by allocating communication time slots for each talking couple of computational subspaces, as shown in Fig. 10 [4].

Virtual Gathering Engine VGE. In real life there can exist thousands to hundreds of thousands of computational subspaces. Quantities such as the total mass of the system are calculated by adding relevant quantities (masses) of all subspaces. The total mass of all entities within a given computational subspace is calculated by the VSE. The accumulation of the masses of all the computational subspaces is done by employing the VGE, where a tree of subspaces is built in such a manner that, at each communication time slot, each subspace talks only to another subspace. This is shown in Fig. 11 where all communications for the VGE are split into four communication time slots. In this way, most of the computational subspaces are busy most of the time resulting in relatively fast data gathering process.

Virtual Broadcasting Engine VBRE. Very often one needs to transfer information from an external source to each computational subspace. The information being transferred is generally different for each different subspace, i.e., not the same information is being sent to all subspaces. To accomplish this task, the virtual broadcasting engine (VBRE) is used. The idea behind this engine is relatively simply: send all the information to everybody and let the receiving computational subspaces select only what belongs to them. This is done by plugging a "filter" on the receiving end (Fig. 12). This engine is used for reading and distributing input data and other similar operations.

Implementing V-FDEM engines. A FDEM parallel software package implemented using the above engines is:

(a) Free of any parallelization-specific components within VSE;
(b) Optimized to achieve maximum efficiency;
(c) Hardware independent;
(d) Linearly scalable to any number of computational subdomains;

Porting the software to a specific architecture is as simple as writing the architecture specific VBOE, VGE and VBRE engines (with VSE being hardware independent), which is

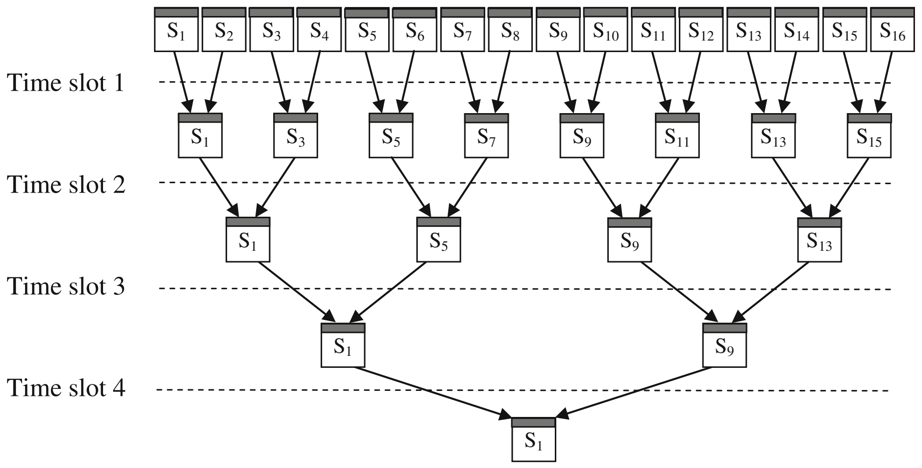
Time slot 1
Time slot 2
Time slot 3
Time slot 4
S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15 S16
S1 S3 S5 S7 S9 S11 S13 S15
S1 S5 S9 S13
S1 S9
S1

Fig. 11 Virtual Gathering Engine: *each pair of arrows* indicates talking couples within a given time slot, while *dashed lines* indicate time slots; *shaded areas* indicate data filtering

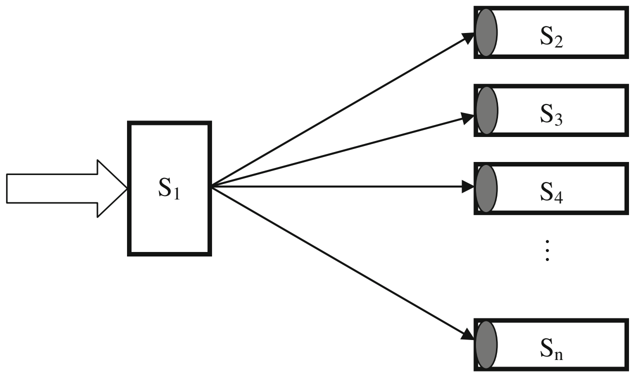

Fig. 12 Virtual Broadcasting Engine principle; *shaded areas* indicate data filtering

therefore a much smaller task than "parallelizing" the whole FDEM code.

As seen in the above illustrations, "filters" are added to all engines. The role of these filters is essential in making the code transparent. The filters are executed by computational subspaces (not by "masters", i.e., "in parallel"), thus speed is achieved and, just as important, these filters may contain data "compression" algorithms which decrease (improve) the communication time even further by reducing the size of the data being passed.

A flowchart of a representative V-FDEM code, with the definition of the four virtual engines is shown in Fig. 13.

4.1 Numerical examples

Measures of parallel computer performance. For the purpose of testing the above V-FDEM, we utilize the standard parallel computing performance parameters: speedup and efficiency. Here speedup defines how much faster the parallel code is when compared to the sequential code, i.e.,

$$S_p = \frac{T_s}{T_p} \tag{3}$$

where T_s is the sequential execution time and T_p is the parallel execution time for p processors. The efficiency of the parallel code is defined as

$$e_f = \frac{S_p}{p} \tag{4}$$

Note that with processor communication, the difference in execution time between processors of the parallel algorithm is very small; we therefore chose the time elapsed in the first processor as the time for the parallel algorithm, T_p.

Domain Decomposition. In the following simulations, the domain decomposition approach is used, where the physic space is divided into $M \times N$ identical Eulerian domains (Fig. 14), each assigned to a single processor. Physical objects, such as finite elements or discrete elements, are assigned to these domains according to their instantaneous position in space. Figure 14b shows the domains for 2, 4, 9 and 16 processors.

A typical continuum mechanics problem. The performance of the parallel FDEM code was first tested on a pure finite element (FEM) problem. A sketch of the computational model used in this case is shown in Fig. 15a. The physical domain consists of a 100 m size 2D square that has a 1 m diameter borehole in its center. The walls of the borehole are subjected to the pressure pulse shown in Fig. 15b. The performance metrics for this problem are shown in Fig. 16.

A typical fracture mechanics problem. The second example utilizes the same setup as the first one, but in this case the material is allowed to fracture. The performance metrics for this problem are shown in Fig. 17. For illustration purposes,

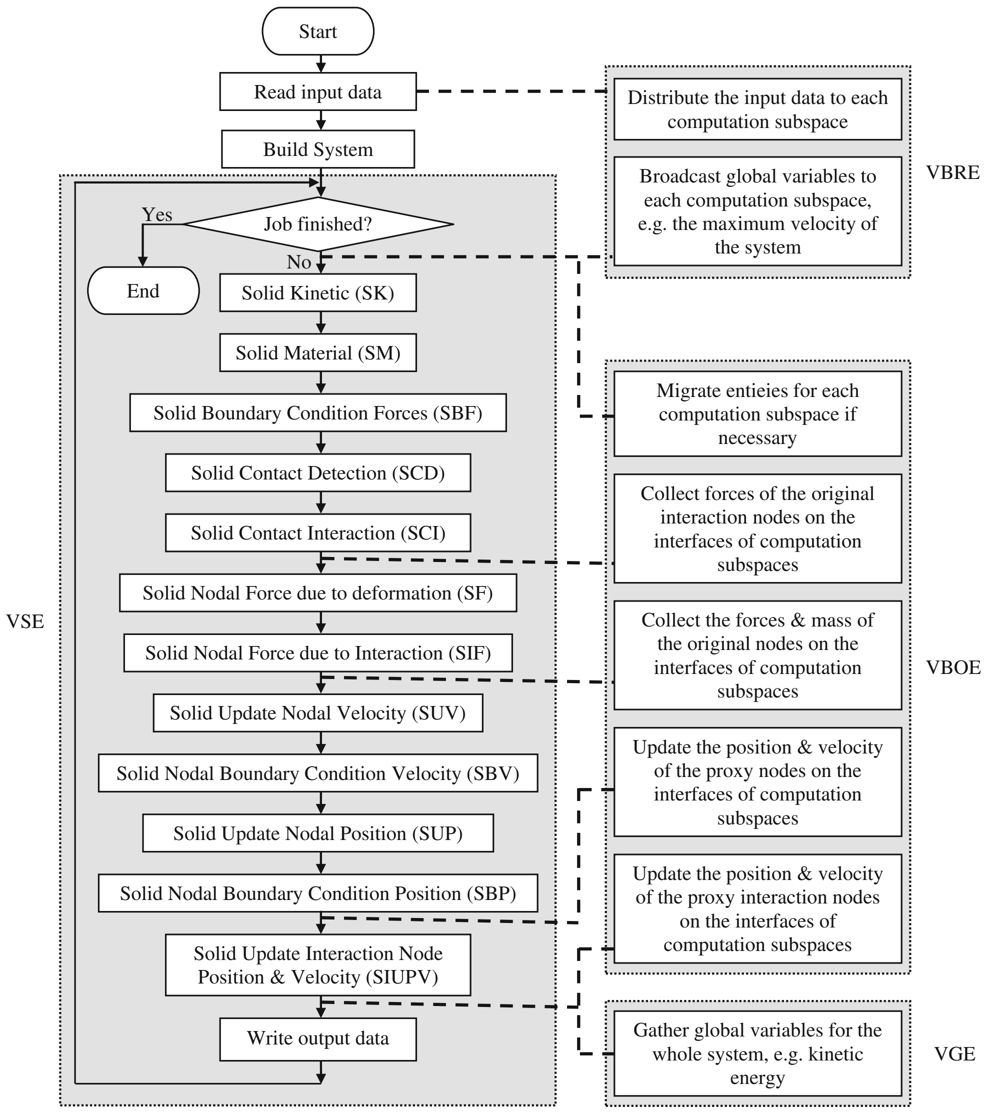
Start
Read input data
Build System
Job finished?
Yes
No
End
Solid Kinetic (SK)
Solid Material (SM)
Solid Boundary Condition Forces (SBF)
Solid Contact Detection (SCD)
Solid Contact Interaction (SCI)
Solid Nodal Force due to deformation (SF)
Solid Nodal Force due to Interaction (SIF)
Solid Update Nodal Velocity (SUV)
Solid Nodal Boundary Condition Velocity (SBV)
Solid Update Nodal Position (SUP)
Solid Nodal Boundary Condition Position (SBP)
Solid Update Interaction Node Position & Velocity (SIUPV)
Write output data
VSE
Distribute the input data to each computation subspace
Broadcast global variables to each computation subspace, e.g. the maximum velocity of the system
VBRE
Migrate entieies for each computation subspace if necessary
Collect forces of the original interaction nodes on the interfaces of computation subspaces
Collect the forces & mass of the original nodes on the interfaces of computation subspaces
VBOE
Update the position & velocity of the proxy nodes on the interfaces of computation subspaces
Update the position & velocity of the proxy interaction nodes on the interfaces of computation subspaces
Gather global variables for the whole system, e.g. kinetic energy
VGE

Fig. 13 Flowchart for a V-FDEM code. *VSE* Virtual Space Engine, *VBOE* Virtual Boundary Engine, *VGE* Virtual Gathering Engine, *VBRE* Virtual Broadcasting Engine

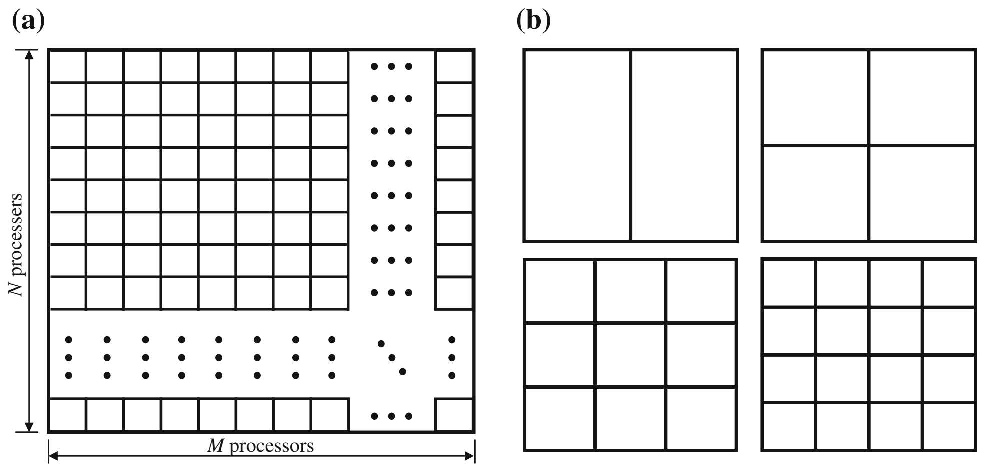

Fig. 14 The space is divided into **a** $M \times N$ regular square domains, **b** 2, 4, 9 and 16 domains

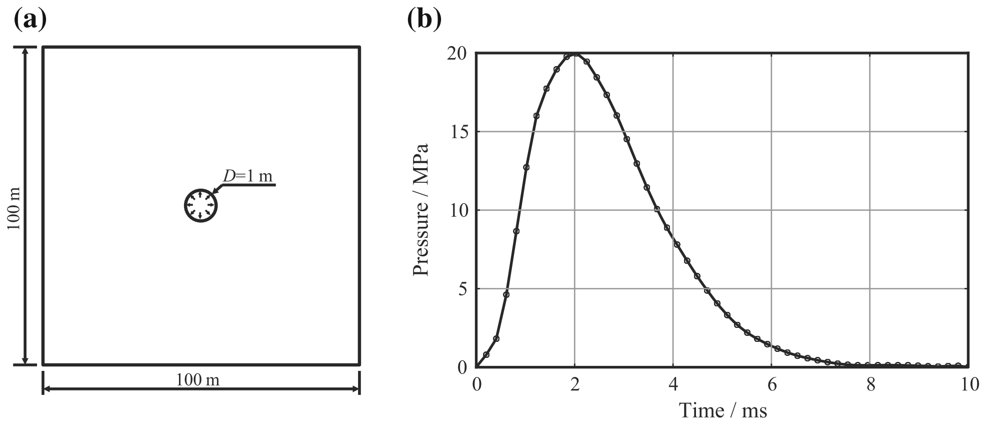

Fig. 15 **a** Borehole model setup; **b** borehole pressure time history

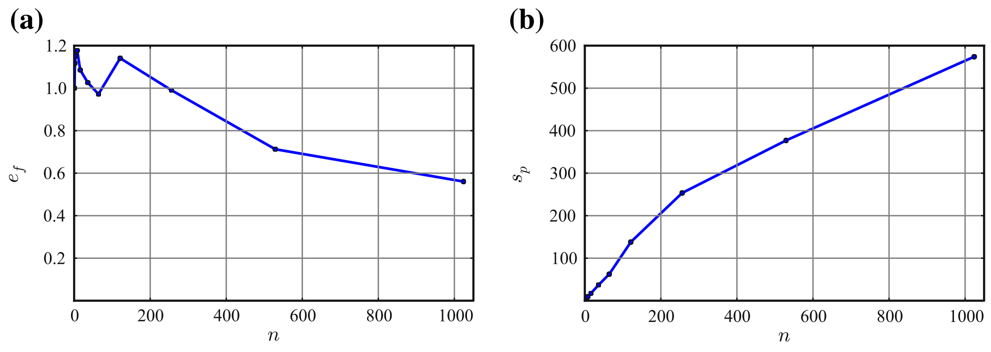

Fig. 16 Pure FEM problem: **a** Efficiency and **b** speed up as a function of the number of processors

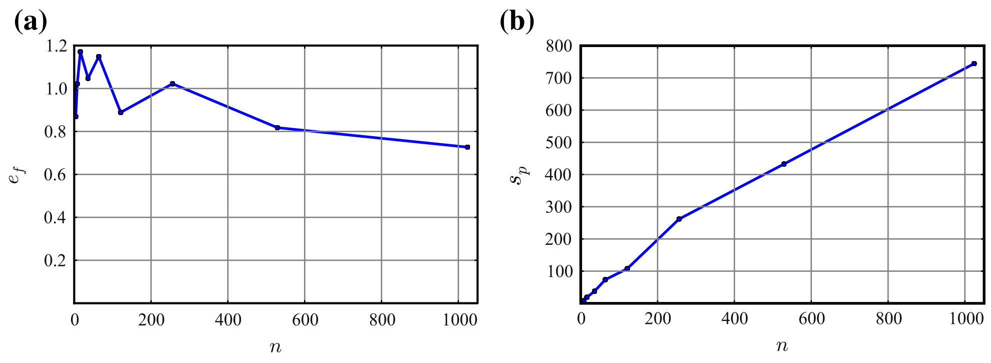

Fig. 17 FDEM problem: **a** Efficiency and **b** speed up as a function of the number of processors

the fracture patterns at the end of the simulation are shown in Fig. 18.

A typical particle mechanics problem. The third example consists of a $N \times N$ regular raster of triangular fully elastic particles interacting among them inside a rigid contained, as shown in Fig. 19. The input parameters used in the simulations are listed in the Table 1.

Each triangle within the raster is provided with an initial random velocity, which changes from triangle to triangle, i.e. all three nodes of a given triangle have the same initial velocity prescribed to them. For a given triangle, the initial velocity is given by

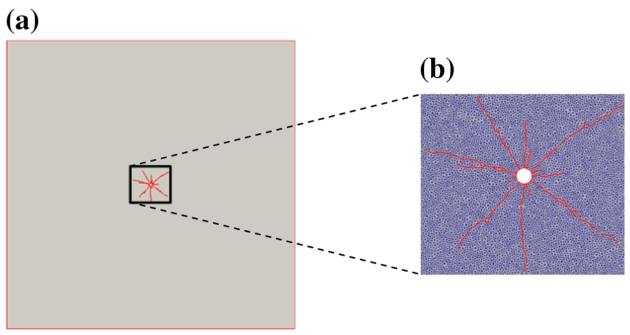

Fig. 18 Fracture patterns at the end of the simulation: **a** general view; **b** detail view

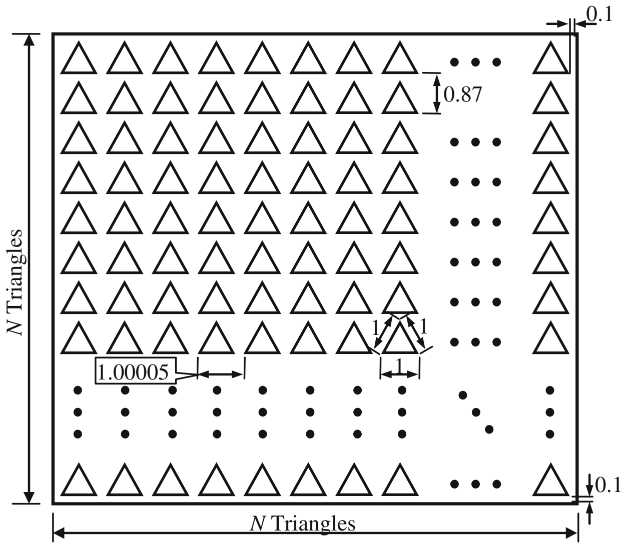

Fig. 19 A raster of N×N elastic triangles enclosed within a *square box*

Table 1 Input parameters

Parameter	Values
Young's modules/GPa	1.0
Poison ratio	0.0
Density/kg/m^3	1,000.0
Contact Penalty/GPa	10
Time step/s	1e-4
Number of steps	500
Maximum Velocity/ms^{-1}	100.0

$$\mathbf{v}_{ini} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} = v_{\max} \begin{bmatrix} 2r_n - 1 \\ 2r_{n+1} - 1 \end{bmatrix} \tag{5}$$

where v_{max} is the maximum velocity and r_n and r_{n+1} are two random numbers obtained from a uniform random distribution, such as

$$0 \le r_n \le 1$$
$$0 \le r_{n+1} \le 1 \tag{6}$$

For a given raster of triangle a single series of random numbers is used [24]. The initialization of the velocities for each triangle is illustrated in Fig. 20. The random number generator used for these examples is the one named "rand3" in Press et al. [24].

The problem has been run using different number of processors on a high performance computing Linux cluster. The performance metrics for this problem are shown in Fig. 21.

5 Further discussion of the performance

In order to test the performance of larger scale simulations, an alternative metric was used to measure the performance of the code. This metric is in the form of a specific time, T_n, which is given by

$$T_n = \frac{T_0 n}{n_{tri} n_{step}} \tag{7}$$

where T_0 is the CPU time elapsed on one of the processors (processor 0), n is the number of processors, n_{tri} is the number of triangles in the system and n_{step} is the number of time steps considered during the analysis.

The performance of the code was measured for systems with one, five and ten million triangles and using 1, 4, 16, 36, 64, 121, 256, 529, 1,024 and 2,116 processors. The results for the performance of the overall code are shown in Fig. 22. A breakdown of the total time, taking into account the four virtual engines for the system comprising one million triangles, is shown in Fig. 23. A breakdown of the total time taking into account the four virtual engines for the system comprising five million triangles is shown in Fig. 24. A breakdown of the total time taking into account the four virtual engines for the system containing ten million triangles is shown in Fig. 25.

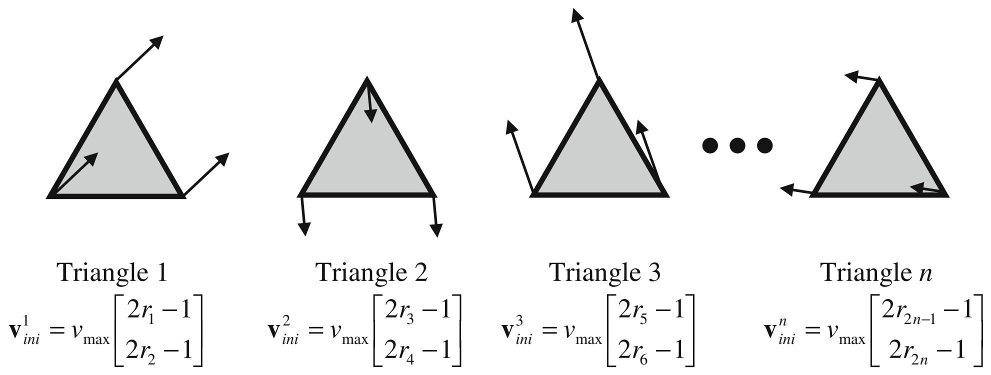
Triangle 1
$\mathbf{v}^1_{ini} = v_{\max}\begin{bmatrix}2r_1-1\\2r_2-1\end{bmatrix}$
Triangle 2
$\mathbf{v}^2_{ini} = v_{\max}\begin{bmatrix}2r_3-1\\2r_4-1\end{bmatrix}$
Triangle 3
$\mathbf{v}^3_{ini} = v_{\max}\begin{bmatrix}2r_5-1\\2r_6-1\end{bmatrix}$
Triangle n
$\mathbf{v}^n_{ini} = v_{\max}\begin{bmatrix}2r_{2n-1}-1\\2r_{2n}-1\end{bmatrix}$

Fig. 20 Prescribing the initial velocity for the *triangles*

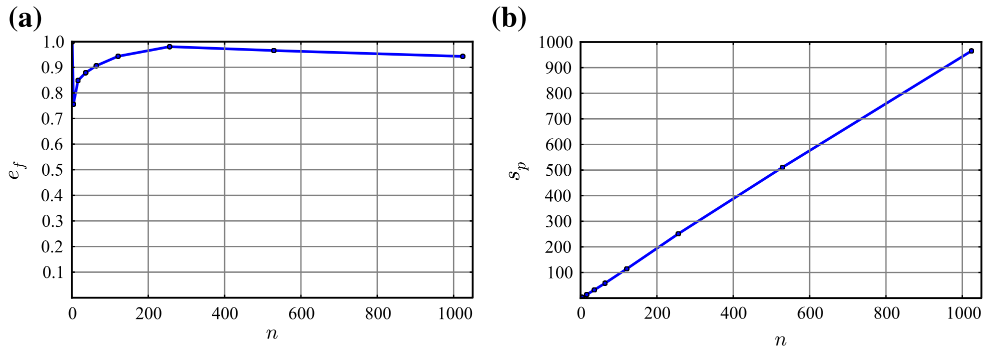

Fig. 21 Particle mechanics problem: **a** Efficiency and **b** Speed up as a function of the number of processors

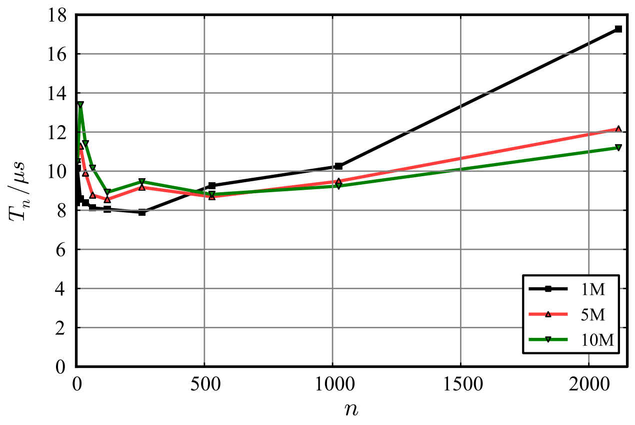

Fig. 22 Specific time as a function of number of processors for systems comprising: one, five and ten million triangles; 1, 5 and 10 M respectively

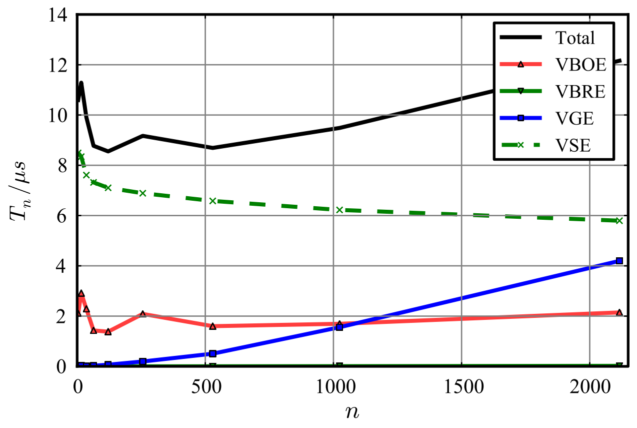

Fig. 24 Specific time for each virtual engine as a function of number of processors for a system comprising five million triangles

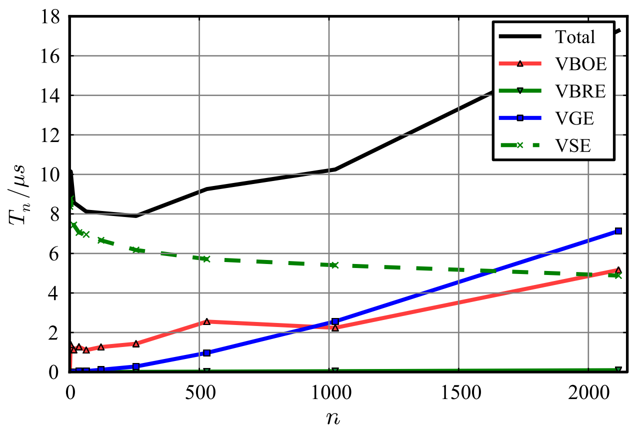

Fig. 23 Specific time for each of the four virtual engines as a function of number of processors for a system comprising one million triangles

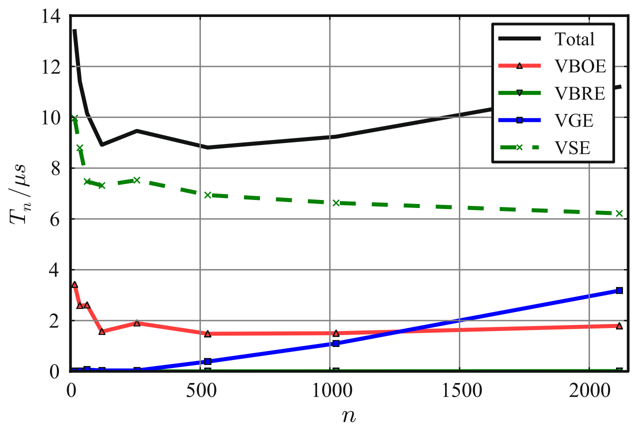

Fig. 25 Specific time for each virtual engine as a function of number of processors for a system comprising 10 million triangles

The evolution of the specific time T_n for the Virtual Space Engine and for the whole code with the number of triangles and the number of processors is shown in Fig. 26a, b respectively.

6 Conclusions

In this work it has been demonstrated that it is possible to build a problem-specific (as opposed to computer architecture-specific) virtual parallel machine for parallelization of existing FDEM software packages. The developed vir-

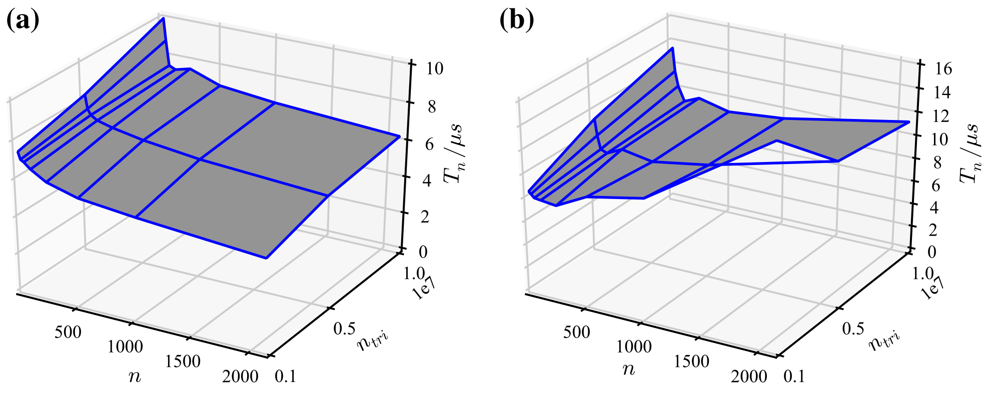

Fig. 26 A 3D graph of the total specific time as a function of the number of processors and number of triangles for the Virtual Space Engine (**a**) and for the whole V-FDEM package (**b**)

tual parallel machine consists of four virtual engines, one of which is the unchanged existing sequential FDEM package.

Porting the parallel software to different parallel computer hardware architectures is thereby reduced to implementing some optimized aspects of the three remaining virtual engines and is in general, a relatively inexpensive operation.

The performance, especially efficiency of the parallelized software, still depends on the particular architecture of the computer hardware being used. However, the test results shown in this paper clearly demonstrate that it is possible to achieve very high efficiency even when a relatively large number of processors is employed, as clearly demonstrated by the results seen in Fig. 22, where only a marginal increase in the specific time occurs despite the number of processors changing from only few to a couple of thousand. It is worth mentioning that in the ideal case the specific time should stay constant—in reality, due to the need to have communication between processors, the specific time always increases and the efficiency of parallelisation is achieved by keeping the specific time as close to constant as possible. For a specific architecture this is done by optimising the three virtual FDEM parallelisation engines. Thus, taking into account the specific hardware characteristics of the particular architecture.

References

1. Rougier E, Munjiza A (2010) MRCK_3D contact detection algorithm. In: Proceedings of fifth international conference on discrete element methods. London, UK
2. Munjiza A, Andrews KRF (1998) NBS contact detection algorithm for bodies of similar size. Int J Numer Meth Eng 43:131–149
3. Munjiza A (2004) The combined finite-discrete element method. Wiley, New York
4. Munjiza A, Knight EE, Rougier E (2012) Computational mechanics of discontinua. Wiley, New York
5. Munjiza A, Rougier E, John NWM (2006) MR linear contact detection algorithm. Int J Numer Meth Eng 66(1):46–71
6. Rockfield, ELFEN (2001) Rockfield Software, UK, http://www.rockfield.co.uk/elfen.htm
7. Munjiza A (1990–1992) RG combined finite discrete element code - C++
8. Munjiza A, Andrews KRF, White JK (1999) Combined single and smeared crack model in combined finite-discrete element analysis. Int J Numer Methods Eng 44(1):41–57
9. Mahabadi OK, Grasselli G, Munjiza A (2009) Numerical modelling of a Brazilian Disc test of layered rocks using the combined finite-discrete element method. In: Diederichs M, Grasselli G (eds) RockEng09: 3rd Canada–US rock mechanics symposium, Toronto–Canada, 9–15 May 2009, Paper 4148
10. Lisjak A, Grasselli G (2010) Rock impact modeling using FEM/DEM. In: Munjiza A (ed) Discrete element methods: 5th international conference on discrete element methods. London–UK, 25–26 August 2010. pp 269–274
11. Lei Z, Rougier E, Knight EE, Munjiza A (2013) Block Caving Induced Instability Analysis using FDEM. 47th US rock mechanics/geomechanics symposium.
12. Knight EE, Rougier E, Sussman AJ, Broome ST, Munjiza A (2013) Split Hopkinson pressure bar experiment simulation using MUNROU. 47th US rock mechanics/geomechanics symposium
13. Latham JP, Xiang J, Belayneh M, Nick HM, Tsang C-F, Blunt MJ (2012) Modelling stress-dependent permeability in fractured rock including effects of propagating and bending fractures. Int J Rock Mech Min Sci 57:100–112
14. Latham JP, Guo L, Wang X, Xiang J (2011) Modelling the evolution of fractures using a combined FEMDEM numerical method. In: Proceedings of the 12th congress of the international society for rock mechanics
15. Barla M, Piovano G, Grasselli G (2012) Rock slide simulation with the combined finite discrete element method. Int J Geomech 12(6):711–721
16. Xu D, Kaliviotis E, Munjiza A, Avital E, Ji C, Williams J (2013) Large scale simulation of red blood cell aggregation in shear flows. J Biomech 46(11):1810–1817
17. Latham JP, Xiang J, Harrison JP, Munjiza A (2010) Development of virtual geoscience simulation tools, VGeST using the combined finite discrete element method, FEMDEM. In: Proceedings of the 5th international conference on discrete element methods, London, 25–26 August 2010.
18. Mahabadi O, Lisjak A, Munjiza A, Grasselli G (2012) Y-Geo: new combined finite-discrete element numerical code for geomechanical applications. Int J Geomech 12 SPECIAL ISSUE: Advances in Modeling Rock Engineering Problems, pp 676–688

19. Wang L, Li S, Zhang G, Ma Z, Zhang L (2013) A GPU-based parallel procedure for nonlinear analysis of complex structures using a coupled FEM/DEM approach. Math Problems Eng. vol 2013, Article ID 618980
20. Lukas T, Munjiza A (2010) Parallelization of an Open-Source FEM/DEM Code Y2D. In: Proceedings of the 5th international conference on discrete element methods, London 25–26 August 2010
21. Bangay S (1993) Parallel implementation of a virtual reality system on a transputer architecture, PhD Thesis, Rhodes University
22. Hillis WD, Tucker LW (1993) The CM-5 connection machine: a scalable supercomputer. In: Communications of the ACM 36(11)
23. Wasniewski J (1996) Applied parallel computing. Industrial computation and optimization: third international workshop, PARA'96, Lyngby, Denmark, 18–21 August 1996, Proceedings. Springer, Berlin
24. Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2002) Numerical Recipes in C, the art of scientific computing, 2nd edn. Cambridge University Press, Cambridge