Queen Mary
University of London

Computational and Algorithmic Solutions for Large Scale Combined Finite-Discrete Elements Simulations

by

Guillermo Gonzalo Schiava D'Albano

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy of Queen Mary University Of London

School of Engineering And Material Science

2013

I dedicate this thesis to my grandfather Pocho (i still miss you), My granny Lila, mum Dora (I am because of you), my father Guille, my brother Seba, my sister Lore and my uncle Beto.

Declaration of originality

The material presented in this thesis is entirely the result of my own independent research under the supervision of Professor Antonio Munjiza. All published or unpublished material used in this thesis has been given full acknowledgement.

Name: Guillermo Gonzalo Schiava D'Albano Date: 24th April 2014

Signature:

List of Publications

Journal Papers:

MUNJIZA, A. ; XIANG, J. ; GARCIA, X. ; LATHAM, J. P. ; **SCHIAVA D'ALBANO, G. G.** ; JOHN, N.W.M. : The Virtual Geoscience Workbench, VGW: Open Source tools for discontinuous systems. In: Particuology 8 (2010), S. 100–105

Conference Proceedings:

SCHIAVA D'ALBANO, G. G. ; MUNJIZA, A. LUKAS, T.: Novel MS (MunjizaSchiava) contact detection algorithm for multi-core architectures. In: ECCOMAS Stuttgart, 2013

SCHIAVA D'ALBANO, G. G. ; MUNJIZA, A.: FEM/DEM simulations on Multicore PC. In: 6th International Conference on DEM Colorado, 2013

BARGIACCHI, M.; **SCHIAVA D'ALBANO, G. G**. ; REVELL, A. ; SMITH, K.: Towards Electrospray Modelling using Lattice Boltzmann Method. In: ICMMES Oxford, 2013

SCHIAVA D'ALBANO, G. G. ; MUNJIZA, A. ; ROUGIER, E. ; LATHAM, J. P. ; JOHN, N.W.M. : Fluid driven fracture process in FEM/DEM analysis. In: DEM Beijin, 2008, S. 242–248

MUNJIZA, A. ; XIANG, J. ; GARCIA, X. ; LATHAM, J. P. ; **SCHIAVA D'ALBANO, G. G**. ; JOHN, N.W.M. : The Virtual Geoscience Workbench, VGW: Open Source tools for discontinuous systems. In: DEM Beijin, 2008, S. 113–121

Abstract

In this PhD some key computational and algorithmic aspects of the Combined Finite Discrete Element Method (FDEM) are critically evaluated and either alternative novel or improved solutions have been proposed, developed and tested. In particular, two novel algorithms for contact detection have been developed. Also a comparative study of different contact detection algorithms has been made. The scope of this work also included large and grand scale FDEM problems that require intensive use of CPU; thus, novel parallelization solutions for grand scale FDEM problems have been developed and implemented using the MPI (Message Passing Interface) based domain decomposition. In this context a special attention is paid to the rapidly developing multi-core desktop architectures. The proposed novel solutions have been intensively validated and verified and demonstrated using various problems from literature.

Acknowledgements

I would like to give a special acknowledgment to my supervisor Ante Munjiza, for teaching me how to be a researcher, and supporting me through the entire research process. I will start the rest of the many acknowledgements in reverse chronological order: Katie Smith, Cristian Sutter, Tomas Lukas, Mariano Sallito, Julian Mindel, Gustavo Gonzalez, Osnat Hakimi, Pete (Pepe) Collins, Joaquin Palmero, Noelia Diaz Vicedo, Manuel Sebastial Guelfi, Cristobal Perez Torres Granaaaa, Stephanie Lynam, Maria Nieves Curras, Ernesto Pineda Leon, Carlos Hidalgo, Pedro Baiz, Beata Manthey, Dominica Luszpinska, Viviana Castana, Carlos Conde Solares, Horacio Barrabasqui, Rafael Ramborger, Simon Vargaz Russo, Hector Vargaz Russo, Martin Cabesas,Jayne Hawkins, Elena di Mascio, Jonathon Hills, Jean Hefford, M Raymond Lam, Alem Million. Mario Derrico, Carlos Sacco, Esteban Gonzales Garcia, Kiko, Gustavo Scarpin, Esteban Rougier. Vanesa Hernanz, Pericles, Vivi, Sandra Ciampone, Shaz, Omid, Marco Marino, Tato Saenz, Marta Delgado, Tono Delgado, Diego Obere, Sergio Feldman, Alvaro Diez, Beata Kilkiewicz, Carlos Fernando Cruz Blanco, Sara Donnarumma, Helga Elvira László, Cristina Peneña kit, Marita Kilroe, Belen Ruggeri, Luz Maria Romero Nuñes, Alan Parish, Sicilia, Asturias, Londra, Cordoba, Toro y Pampa, Joaquin Sabina and the debugger of Visual Studio 2008. To the best cafe in London Town “Pellicci’s”: mamma Maria, Anna, Nevio and Tonny . Finally, these are not my words, but they allow me to summarize to all of you what I could not articulate here; “the path to becoming a doctor is littered with distractions. I would like to thank those distractions for making me the person I am”

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

SCOPE AND LAYOUT OF THE THESIS

1.1 Scope of the thesis

The Combined Finite-Discrete Method (FDEM) was developed in the mid ’90s by Munjiza. In this formulation each body, entity, particle is discretized into N =1, 2, 3, ..., m finite elements (FEs) such as triangles and tetrahedra among others. Finite elements are “glued” together using joints; there is no need to build a global system of equations as for the case of a traditional FEM method or as the case of the Discontinuous Deformation Analysis (DDA). Instead all the equations to be solved are local, facilitating the development of parallel solutions.

One of the key aspects in any DEM software is the contact detection (CD) algorithm. There are many algorithms previously developed in the literature, however deciding which to use for a particular simulation among them is a challenging task. In this context a detail comparison of two of the well known linear algorithms for bodies of similar size Non Binary Search (NBS) and Munjiza Rougier (MR) alongside a flavour of the MR algorithm developed in this work denominated MR-Schiava and novel binary tree (BT) algorithm Balance Binary Tree Schiava (BBTS) is presented.

Interaction between particles, bodies and entities defines the type of CD that will be employed. Interactions are not, however, always between bodies of similar size. When a CD algorithm optimized for discrete elements of the same size is employed for a system with a wide range of sizes the performance measure in CPU time will suffer. To simplify interaction (tetrahedra/line, tetrahedra/surface, tetrahedra/triangle, tetrahedra/tetrahedra, etc) and the CD algorithm associated with it in this work all interactions are point/body, regardless of where the point is or who the point belongs to (surface, triangle, tetrahedra, etc). A novel algorithm for point/body CD has been developed in this work denominated MunjizaSchiava (MS). This algorithm is described and tested in sequential and parallel simulations.

FDEM simulations ran on desktop PCs are generally limited by the CPU power of current processors. The increase in clock speed in the last 5 years has reached a barrier due to thermal constraints in the dissipation of heat energy. The industry is gradually moving towards the use of more processors with slower clock speeds; nowadays almost all commercial off-the-shelf PCs are Multicore.

Conventional sequential programs cannot take advantage of the extra power available in multicore systems, as they can only use one processor at a time. The only alternative for the industry and research community is to embrace parallelization on a scale never seen before. To this end a novel algorithm using MPI (Message Passing Interface) has been developed and tested using Multicore PC.

1.2 Layout of the thesis

This thesis is divided into several chapters detailing the various algorithms and test undertaken. The contents of each chapter are as follows:

- Chapter 2 shows the different methods of discontinua.
- Chapter 3 introduces the FDEM method alongside different aspects of parallel programming.
- Chapter 4 provides a detailed description of the novel BBTS CD algorithm for bodies of similar size.
- Chapter 5 presents a short description of NBS, MR and the new MR-Schiava algorithms.
- Chapter 6 explains and tests a novel CD algorithm for point/body.
- Chapter 7 presents comparisons between NBS, MR, MR-S and BBTS
- Chapter 8 features an algorithm for non-elastic normal interaction.
- Chapter 9 details and tests a novel parallel solution for the FDEM.
- Chapter 10 presents some application examples.
- Chapter 12 presents the conclusions and makes suggestions for further research.

Chapter 2

METHODS OF DISCONTINUA IN GENERAL

2.1 Introduction

Methods of Discontinua are a relatively new set of tools revolutionising the scientific community. The core of these methods is the intrinsic discontinuity present in nature, where complex phenomena are the result of independent entities interacting with each other. For example the local interaction of grains forming a metal bar, influences the macroscopic properties of it, such as ultimate strength, fracture strength, etc. The first part of this chapter contains a brief account of the development of the Methods of Discontinua. Following on from this, the second part provides a summary of the different Computational Methods for Discontinua:

- Discrete Element Method (DEM)
  - Spherical particles
  - Oblique and super quadratic particles
  - Polygon model
  - Real shape particles
- Combined Finite-Discrete Method (FDEM)
- Discontinuous Deform Analysis (DDA)
- Molecular Dynamics (MD)

- Smoothed Particle Hydrodynamics (SPH)

Science and Engineering disciplines have experienced a significant advance during the past forty years. Nano-devices[57] and nano-structures[192] are being used for information technology, Micro-Electro-Mechanical Systems[69, 68, 43] (MEMS) have been implemented in sensors, heat pumps, engines, etc. Smarter materials (such as carbon-fibre) are being used in automobiles[62, 26] and airplanes, changing the paradigm of what it is possible to build.

These wide ranges of sizes and physical characteristics have produced a similar wide range of tools to solve each particular case. In the case of fluid dynamics, the two main groups[69] are:

- Molecular Models
- Continuum Models

Each of the method has its limitations and a universal tool, which can be used to perform fluid dynamic simulations across the entire spectrum of engineering problems, has yet to be created (i.e. from nano-devices[175] to human scale engineering).

A standard tool to discretise continuum systems is the Finite Element Method[32, 211] (FEM). In FEM the domain is subdivided into a "finite" set of smaller domains where a series of partial differential equations are solved considering the boundary conditions. Each of these subdomains has the same continuum properties as their main domain.

Not all systems can be simulated using FEM as there is not always guarantee of continuity throughout the entire domain. A common feature in mountains is called fault.[87] Fault presents a distinctive "jump" in the continuity of the rocks. This discontinuity poses a difficult problem for FEM.[82, 129] It is possible to think that a mountain is made of independent yet interacting bodies with the fault being the boundary between them, where there is not one domain but a series of interacting smaller domains.

Another area in which classical FEM is not well suited is the calculation of granular flows[205] where the intrinsic discrete nature of the flow makes the continuum assumption an impossibility. These flows have great importance in industrial applications such as mills[28] and pharmaceutical industries.[197]

The so called Methods of Discontinua were developed to deal with the discontinuities of the domain: problems in which the interaction of atoms,[175] particles,[184, 36, 101, 56] bodies,[133, 209, 149] rock joints,[130] granular flows,[28, 30] rock cutting[174] and rock braking[127, 126] play an influential role in the behaviour of the system. An important characteristic of these methods is the capacity of dealing with continuous changes in the interfaces solid/solid[131] and solid/fluid.[143]

In 1971, Cundall[34] developed the Distinct Element Method to analyse rock mechanics problems of which the main characteristic was the discontinuity in the domain. From Cundall and his early application on rocks mechanics there were further developments in other areas of engineering and science during the 1970s and early 1980s. In 1977, Lucy, Gingld and Monaghan, developed Smooth Particle Hydrodynamics (SPH) to solve "astrophysical problems".[106] The first industrial application was in particle flows in 1979[35] with further developments by Campbell in 1985[23] and Haff and Werner in 1986.[66] At the end of the '80s, Shi[182] developed the Discontinuous Deformation Analysis (DDA) in his PhD thesis. In the mid '90s Munjiza introduced the FDEM (FEM/DEM)[145] method with further progress in contact detection in 1998[136] and 2006.[148]

The complete list of developments during this important period of the Methods of Discontinua is out of the scope of this thesis. Further reading can be found in the review papers by Campbell,[22,21] Goldhirsh,[58] Maclaughlin[116] and the books, by Jing and Stephansson[82] and Munjiza et al.[141]

As stated by Cundall[36,130] the characteristics of the DEM method, which can also be applied to any Method of Discontinua are:

- "Allows finite displacements and rotations of discrete bodies, including complete detachment"[36]
- "Recognizes new contacts automatically as the calculation progresses"[36]

These two characteristics and the necessity of running simulations with a high number of elements, implies a high computational cost.[201,212,49,81] This constraint in the simulations it is possible to run, has produced the development of new algorithmic solutions called parallel algorithms.[160] Depending on the hardware architecture, there are different application programing interfaces (API), that can be used. On clusters[85,152] it is possible to produce code using MPI,[159,61] OpenMP,[25] in multicore MPI, OpenMP, port base techniques[76,202] and in Graphics Processing Units (GPU), CUDA[91,153] and the new developed OpenCL.[24,59]

The use of GPU processors not only to compute graphics but also to solve DEM problems presents great computational potential. For example one of the latest Nvidia GPU hardware, the TESLA C2070, has 448 CUDA cores.

2.2 Computational Methods of Discontinua

The expression "Computational Methods of Discontinua" is the general term used to define any kind of numerical method dealing with a distinct, independent population of objects that represent the domain.[82] This general term is applied to:

- Discrete Element Method (DEM)
- The Combined Finite-Discrete Element Method (FDEM)
- Discontinuous Deformation Analysis (DDA)
- Molecular Dynamics (MD)
- Smoothed Particle Hydrodynamics (SPH)

One of the most important characteristics is the concept of emerging properties[147, 172] resulting from Virtual Experimentations. Each entity poses a particular law of interaction with other entities and physical quantities such as pressure, temperature and density will arise as a consequence of the Virtual Experimentation; as is in the case of MD, where droplets of fluids[175] are formed without any kind of model for surface tension, or in the case proteins[95] where new chemical compounds are emerging from the simulations.

The mathematical roots[82] of these methods are the Finite Difference Method[100] (FDM), the Finite Element Method[211] (FEM) and the solution of the equation of solid mechanics.[204, 144] The integration scheme depends generally on the kind of problem to be solved and can be explicit for the cases of rigid body systems[82, 133] and FDEM,[133, 141] or implicit for DDA.[82]

This wide range of methods[156] and applications, has only been possible with the development of numerical algorithms and hardware capable of dealing with the kind of intensive (in terms of CPU time) methods that were only dreamed of a few decades ago. There are some overlapping algorithmic solutions between the different methods as is the case with contact detection (CD) algorithms that can be used in DEM, MD, etc. The remainder of this section describes each one of these methods.

2.2.1 Discrete Element Method

There are two main areas in DEM: one is the contact detection and the other is the contact interaction.[144] Contact detection is the search for all the possible contacts between all the entities in the same domain. For contact detection, a wide range of algorithms have been proposed such as Alternating Digital Tree (ADT),[18] 3D-DDA,[16] MR,[148] NBS[136] and many others. For contact interaction there are two major approaches:[75] one is the so-called "soft particle" and the other one is the so-called "hard particle". In the soft particle algorithms a small overlap between the different entities is allowed and the forces are calculated from this overlap.[134] The hard particle approach does not allow for any overlap. Usually the "soft particles" are applied for mechanical problems while the "hard particles" are used in Molecular Dynamics.

There is a large volume of published studies describing the role of particle shapes in the behaviour[97,48,64] of the system. Properties such as inter-grain forces, velocities,[131] breakage rate,[133,75] packing density[65,28,64,121] and permeability[54] are particle shape dependent.[38] In the case of shear flows,[29] the importance of the shape cannot be relegated as final values of flow rates, flow temperature and boundary interaction varies dramatically. These changes can be observed in real and virtual experiments.[123]

The some of the most common particle shapes for the DEM are:

- Spherical particles
- Oblique and super quadratic particles
- Polygon model
- Real shape particles

For a more complete list please refer to the book by Munjiza et al.[141] and the review paper of Dziugus and Peters.[40]

2.2.1.1 Spherical particles

The first particles used in the Distinct Element Method (Discrete Element Method) were circular particles in 2D. The typical spherical particle[141] shown in Figure 2.1 a is described only by its position, radius, velocity, radial velocity, mass and inertia.

Simulations of tens of thousands and more of these particles[175] are possible as their computational cost (contact detection, interaction, etc) is small compared with other kinds of particles. The biggest restriction is what physical problems can be addressed. An intermediate solution between a spherical particle and a more complex particle is the use of "virtual vertex" to take into account the rotational resistance.[207] The difference between this model and the 2D disc is in the calculation of the momentum;

when two particles are in contact the calculated force is applied to the nearest virtual vertex, as shown in Figure 2.1b.

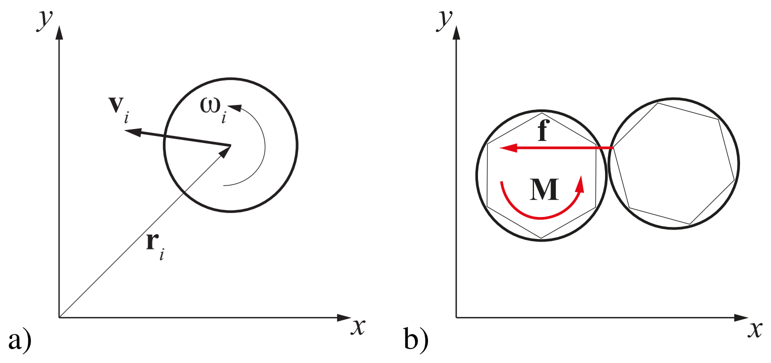


Figure 2.1: Circular particles a)Description. b)Contact with virtual vertex. Figure modified from Yamada et al.[207]

2.2.1.2 Oblique, super quadratic particles

In 1991, Williams et al. published a paper[203] in which they described a new kind of particle for DEM. These particles were still rigid but their shapes were not longer limited to circles in 2D or spheres in 3D.

The equation to produce them in 2D[141] is

$$\left(\frac{x}{a}\right)^m + \left(\frac{y}{b}\right)^m = 1 \tag{2.1}$$

where m is a positive number, a is the size in the x axis and b is the size in the y axis. In 3D the equation to define them is

$$\left[\left(\frac{x}{a}\right)^{\frac{2}{m}} + \left(\frac{y}{b}\right)^{\frac{2}{m}}\right]^{\frac{m}{n}} + \left(\frac{z}{c}\right)^{\frac{2}{n}} = 1 \tag{2.2}$$

Some examples of 3D super-quadric particles are shown in Figure 2.2.

New applications of super-quadric particles are coming to light as they are applied to solve comminution processes (industrial particle breakage). Delaney et al.[38] define the super-quadric particles as

$$\left(\frac{x}{a}\right)^m + \left(\frac{y}{b}\right)^m + \left(\frac{z}{c}\right)^m = 1 \tag{2.3}$$

Their novel technique consists in the way that they treat the particles once they break. When the algorithm detects the breaking of a particle, it will exchange it with a set of smaller super-quadric particles that occupy the same volume. The size and the distribution of these new particles can be modified, depending of the particular material and problem to solve.

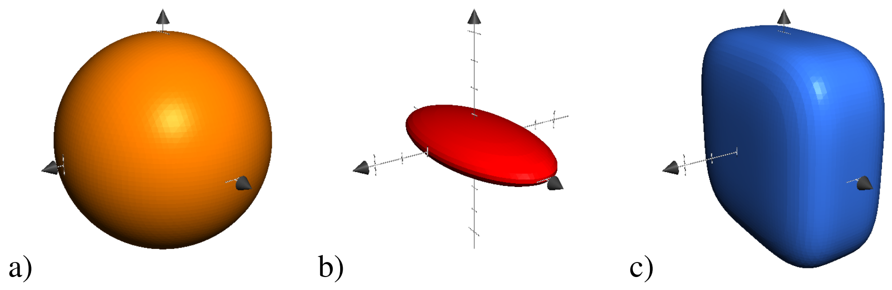


Figure 2.2: 3D quadric particles. a)$a = 1.0$, $b = 1.0$, $c = 1.0$, $m = 1.0$, $n = 1.0$ b)$a = 1.0$, $b = 0.2$, $c = 0.5$, $m = 1.0$, $n = 1.0$ c)$a = 1.0$, $b = 1.0$, $c = 0.5$, $m = 0.5$, $n = 0.5$

2.2.1.3 Polygon model

Fraige et al.[51] developed this particle model to simulate vibration flow in hoppers. The polygon is made of vertices in which the outer angle is always $>$180 degrees. The union between the vertexes is made with a disk as shown in Figure 2.3. This rounded vertex helps the calculus of the DEM problem, as real particles "do not contain perfect vertices".[51] The particles are rigid and cannot be deformed.[197]

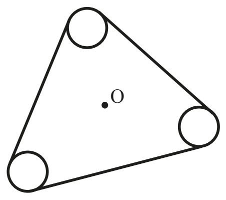


Figure 2.3: Polygon DEM particle of 3 edges. The point of reference O will not always coincide with the centre of gravity. Figure modified from Fraige et al.[51]

2.2.1.4 Real shape particles

In recent years, there has been an increasing amount of literature on the use of more realistic particle shapes. As the computational power becomes available, working with these kinds of particles is becoming less prohibitive in CPU and RAM terms. It is possible to categorize real shape particles into:

- Super-quadric[194] shape particle
- Pseudo[5, 123, 51] shape particle

- Real[179,97,37,48] shape particles

Super-quadric. It is one of simple particles. To create a super-quadric object from a real particle is necessary to:

1. Get an equivalent super-quadric object
2. Set the same mass, and moments of inertia[194]

Even these quasi-real particles share some characteristics with the original particle, their lack of similar morphology affects the final results.
Pseudo shape particles. Real particles are approximated as a series of *N* non-intersecting spheres, grouped together.[5,123] This has the advantage of simplicity for the contact detection and interacting algorithms, as spheres are simple to calculate. Limiting the amount of different particle shapes for a determinate size distribution simplifies the process of extracting properties. Two of these particles are shown in Figure 2.4

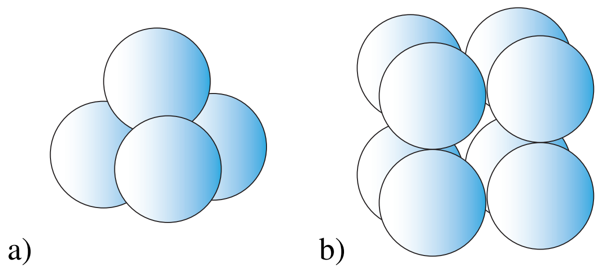


Figure 2.4: Different particles shapes. Figure modified from Abou-Chakra et al.[5]

Real shape particles. These are the next logical step in the DEM simulations. The general steps to obtain real shape particles are:

1. Analyse specimen (laser scan, X-Ray, tomography, etc)
2. Obtain the principal characteristics (shape, moments of inertia, mass)
3. Build DEM model

Non-destructive methods such as image analysis in 3D,[5] laser scan[97,96] and X-Ray tomography,[194,120] can be applied depending of the kind of sample to be analysed. Principal moments of inertia can be obtained from the computational surface model for uniform density samples.[97]

Other parameters are not as simple to obtain. Even for a well-known property such as Young's modulus, the relationship between micro-parameters and macro-parameters will depend on the particle size, the contact modulus and the stiffness ratio.[208] If the particles are allowed to break,[128] other considerations must be taken into account. Extracting fracture properties from samples is far from simple though inverse analysis[90, 89] has been applied in the past with some promising results.

How to represent and build a 2D or a 3D real particle depends on the method. For non-deformable discrete bodies the simplest way to build a particle is to assemble disks in 2D. Asamawy et al.[179, 11] developed an algorithm called Overlapping Discrete Element Cluster (ODEC). The ODEC algorithm in 2D creates a series of overlapping rigid discs (discrete elements), that fill the void of a real particle's "outline shell", as shown in Figure 2.5. By combining the ODEC with PFC2D[1] Asamawy et al.,[179, 11] were able to simulate real soil particles[2].

The extension of ODEC[37] into 3D overlaps rigid spheres. To optimize the calculation time once the particle has been modeled with spheres, the algorithm replaces groups of smaller spheres with bigger ones.

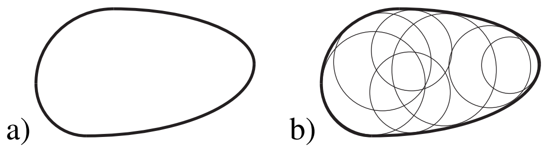


Figure 2.5: Fraser river sand a) Particle shell. b)DEM model using ODEC. Figures modified from Sallam[179]

A new method to simulate real shape bodies using rigid spheres was proposed by Ferellec et al.[48] In this method all the masses of all the spheres are set to the same value regardless of their size to reproduce the particle inertia. Some errors will still occur, but they are smaller than the ones produced if all the spheres have the same density.

2.2.2 The Combined Finite-Discrete Element Method

FDEM was developed in the mid 90's by Munjiza[145] who wrote the first comprehensive book in this area in 2004,[133] followed by a second book in 2011.[141] In this formulation, each entity is discretized into $N = 1, 2, 3, .., m$ finite elements (FEs) such as triangles in 2D, tetrahedron in 3D, etc. This allows the calculus of deformations, fracture and other properties.

[1]Commercial DEM software from Itasca

[2]The PFC2D software has an option to consider a group of circles as a rigid body do not performing contact detection or interaction between them

There is no need to build a global system of equations as for FEM and DDA. In FDEM there is no integration of the equation of motion for the continuum medium because there is no continuous medium. Instead the position, velocity, rotation, etc. are calculated using the equation of motion of each FE. Independent FEs are "glued" together using joints, as shown in Figure 2.6.

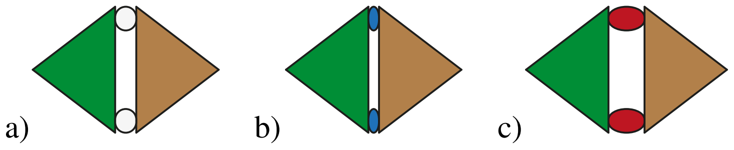


Figure 2.6: Two FEs glued forming a complex FDEM entity. a)Joint forces equal to zero. b)Joints in compression. c)Joints in extension.

If the particles are modelled using FDEM, and the elements are triangles or tetrahedra, there is the advantage to use standard open source/commercial codes (such as GID-Cimne, etc.) to generate the mesh. This makes the creation of real particles libraries simpler[97].

2.2.3 Discontinuous Deformation Analysis (DDA)

The Discontinuous Deformation Analysis (DDA) was developed at the end of the 80's.[182,82] It is used to simulate rocks systems,[182] rock blasting,[63] stability of rocks slopes,[74] stability on tunnels,[12] fluid solid interaction on breakwaters,[84] etc. Validating a new solution for DDA is not trivial, and some of the challenges faced are similar to the ones encountered during FDEM validations. MacLaughlin et al. wrote a comprehensive review on the different validations cases for DDA.[116]

DDA is an implicit method, similar in some aspects to FEM method, as the system to solve is the "so-called energy minimization principle".[82] If total energy in the system, $\prod$ is calculated as

$$\prod = \sum (\mathbf{U}_i) + \mathbf{K} + \mathbf{W} \tag{2.4}$$

where $\mathbf{U}_i$ is the is potential energy, $\mathbf{K}$ the kinetic energy, and $\mathbf{W}$ the dissipated energy. If $\prod$ is differentiated respect the displacements $\mathbf{d}$ as

$$\frac{\partial \Pi}{\partial \mathbf{d}} = \left[\sum \partial (\mathbf{U}_i) + \partial \mathbf{K} + \partial \mathbf{W}\right] / \partial \{\mathbf{d}\} = 0 \tag{2.5}$$

it will produce the system of equations to be solved.[82] The assembly of the system's matrix is a non-trivial algorithmic task. The equations change as the simulation progresses and new contacts are produced, new fractures are created etc. Once the system of equation has been assembled, it is solved implicitly in the same way as many FEM methods.

2.2.4 Molecular Dynamics

Molecular simulations are performed using two methods,[7] one is the "Monte Carlo" (MC) method[8, 178] and the other is the Molecular Dynamics Method (MD)[178, 171]. In 1953, Metropolis et al[125] made the first Monte Carlo simulation.

Monte Carlo simulations calculate the properties of the system by evaluating the equations at random positions **r**. For this reason, it is denominated "Monte Carlo".[67] It is a stochastic method and there is no need to calculate the properties of all molecules in the system.[178] The integrals evaluated are of the form[67]

$$\langle A\rangle = \frac{1}{\mathbb{Z}}\int\ldots\int \exp\left[-\beta U\left(\mathbf{r}_1,\mathbf{r}_2,...,\mathbf{r}_N\right)\right] A\left(\mathbf{r}_1,\mathbf{r}_2,...,\mathbf{r}_N\right) d\mathbf{r}_1\ldots d\mathbf{r}_N \tag{2.6}$$

$$\mathbb{Z} = \int\ldots\int \exp\left[-\beta U\left(\mathbf{r}_1,\mathbf{r}_2,...,\mathbf{r}_N\right)\right] d\mathbf{r}_1\ldots d\mathbf{r}_N \tag{2.7}$$

where $\langle A\rangle$ is the property to be evaluated, $d\mathbf{r}_1 = dx_1\,dx_2\,dx_3$, $\beta = {}^1\!/_{kT}$, k is the Boltzmann's constant and $\mathbb{Z}$ is the configurational integral. This produces great savings in terms of RAM and CPU power, compared with MD.

Molecular dynamics[98] is the study of the interaction between the atoms and/or molecules on an atomic scale. The scale and length of the simulations that can be achieved nowadays with the computational power available is too limited to simulate any kind of real-life problem in engineering.[109] However, there are some kinds of problems where MD represents a good tool and sometimes the only tool, for the calculus of thermo-physical properties like shear viscosity, etc.

If the potential energy of the system is a derivative of the molecular coordinates, the 'emerging' properties such as pressure, temperature, etc, can be obtained[147] utilizing MD simulations.

In Molecular Dynamics, the evolution of the system is calculated by integrating the Newton's equations of motion over all the particles.[189] There are two suppositions:

- The atoms are described as interacting points
- There is no mass change in the system

The interaction between atoms is calculated by a general potential function[109] as

$$U(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N) = \sum_i V_1(\mathbf{r}_i) + \sum_{i,j>i} V_2(\mathbf{r}_i, \mathbf{r}_j) + \sum_{i,j>i,k>j} V_3(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k) + ... \tag{2.8}$$

where U is the potential, $\mathbf{r}_n$ are the position vector of the nth particle, and the function V_m is the m-body potential. The V_1 is the potential from external sources (i.e. gravity, etc), V_2 is the pair-wise interaction between the atoms i, j. V_3 is the interaction between the atoms i, j, K etc. The use of functions of order bigger than two will be expensive in terms of CPU time. Moreover, any complex interaction between atoms or molecules is incorporated inside V_2. In general, the external potentials are not considered.

Usually the potential U is decomposed into the classical intermolecular and intramolecular potential[189] as

$$U = U_{int} + U_{ext} \tag{2.9}$$

When MD is applied to fluids, U_{ext} is decomposed as

$$U_{ext} = U_{d-r} + U_{el} + U_{pol} \tag{2.10}$$

where U_{d-r} is the dispersion-repulsion interaction, U_{el} is the electrostatic interaction, and U_{pol} is the polarized interaction between molecules.

One of the most popular mathematical models for U_{d-r} (V_2) is the Lennard-Jones potential. This potential is the dominant for low polarity systems, like alkanes,[189] and it is calculated as

$$V(\mathbf{r}_1, \mathbf{r}_2) = V(\mathbf{r}) = 4\varepsilon \left[\left(\frac{\sigma}{\mathbf{r}} \right)^{12} - \left(\frac{\sigma}{\mathbf{r}} \right)^{6} \right], \mathbf{r} = |\mathbf{r}_{ij}| = |\mathbf{r}_i - \mathbf{r}_j| \tag{2.11}$$

where ε and σ are constants and the first term accounts for the repulsive potential and the second term accounts for the attractive potential. This potential force is given by

$$f(r) = -\frac{\partial e_p}{\partial r} = \frac{24\varepsilon}{\sigma}\left[2\left(\frac{\sigma}{r}\right)^{13} - \left(\frac{\sigma}{r}\right)^{7}\right] \tag{2.12}$$

This means that all the atoms are interacting with each other. To compute all the iterations among atoms using the Lennard-Jones potential will imply quadratic calculations. There are several methods[67] to truncate the potential function[52] and limit the amount of contact couples. The most common methods are:

- Simple truncation
- Truncation and shift
- Minimum image convection

Simple Truncation Implements a cut-off distance.[67]

$$e_{ps}(r) = \begin{cases} e_p(r) - e_p(r_c) & : r \le r_c \\ 0 & : r > r_c \end{cases} \tag{2.13}$$

The typical value of $r_c = 2.5\,\sigma$ and the value of $e_p(r_c) = -0.0163168911360000\,\varepsilon$ and the force $f_s(r) = -0.0389994774528000\,\varepsilon/\sigma$. This cut-off will influence properties such as pressure and internal energy. Some compensation formulas are proposed by Haile[67] to correct the long range interactions. The total energy will not be constant as the simple truncation produces a "jump" in the potential energy.

Truncation and shift. To avoid the jump in the potential energy[175,67,60] the cut is applied to the force and from that to the potential energy. This potential preserves the total energy[3].

$$f_s(r) = \begin{cases} f(r) - f(r_c) & : r \le r_c \\ 0 & : r > r_c \end{cases} \tag{2.14}$$

If the force is integrated, then the shifted potential[67] with the cut-off radius is given by

$$e_{ps}(r) = \begin{cases} e_p(r) - e_p(r_c) + f(r_c)[r - r_c] & : r \le r_c \\ 0 & : r > r_c \end{cases} \tag{2.15}$$

[3]This is a good tool to debug a MD code.[175]

where r_c is the cut-off distance.

This potential still presents a discontinuity in the force and may produce some "instabilities".[52]

Smoothing function. Frenkel and Smit[52] presented an alternative model for the Truncation and Shift potential where a smoothing function $S(r)$ is defined in order to decrease linearly the potential from $r_l \leq r \leq r_c$ where r_l is the distance where the smoothing function starts and r_c is the truncation distance. The new potential is given by

$$e_p(r) = 4\varepsilon S(r) \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{2.16}$$

and

$$S(r) = \begin{cases} 1 & : r \leq r_l \\ 1 - (r - r_l)^2 (3r_c - r_l - 2r) / (r_c - r_l)^3 & : r_l < r < r_c \\ 0 & : r > r_c \end{cases} \tag{2.17}$$

Molecules present a greater challenge, as their potential cannot be evaluated using the equation 2.11. There are different models that can be used to simplify the molecular potential:

- All atoms
- United atoms
- Anisotropic atoms

If each one of the different atoms are considered as a force centre the model is called "All Atoms". This form of potential is CPU expensive, and other alternatives have been proposed. If the molecule is treated as a single unique point, it will become a single "atom". Depending on the position of this point the model is called "United Atoms". If the point is in the centre of the molecule, and if the centre is located in an intermediate position, the model is called "Anisotropic United Atoms"[189] as shown in Figure 2.7.

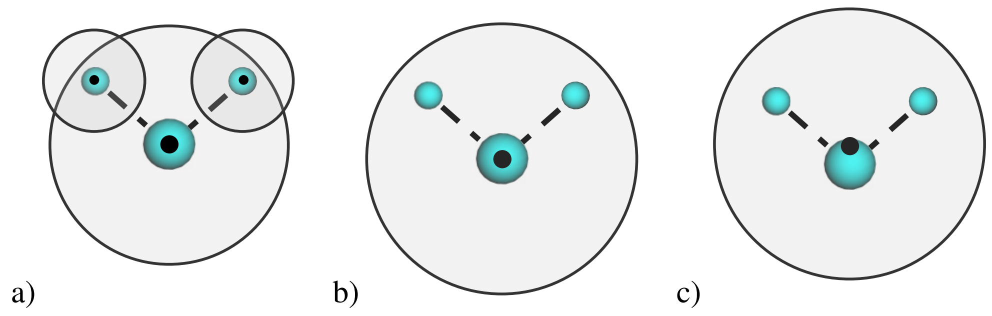


Figure 2.7: Force centre model. a)All atoms. b)United atoms. c)Anisotropic United Atoms

Molecules also present an intramolecular potential U_{int}. As the atoms inside a molecule change their relative position, there will be changes in the stretching, bending, torsion and dispersion-repulsion potentials. The intramolecular potential contributions are neglected if the molecule is considered rigid.

2.2.5 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) was developed at the end of the 1970s, in order to simulate astrophysical problems.[106] The use of mesh methods was not suitable to solve this kind of phenomenon where the mass of the system is located in specific points and the rest of the system is empty. Basically, SPH is a Lagrangian grid-less method in which all the information is located in some defined points, these points will move according to the conservation of energy equations.

Latter SPH was developed to solve different kinds of problems and nowadays is used in diverse areas of science and engineering such as; fluid dynamics,[191] impacts[190, 124] explosions,[183, 107, 169, 108] and granular media.[31, 103]

SPH is becoming one of the most popular mesh-less methods for fluid dynamics,[103] commonly applied to real free-surface flows.[108] The most important characteristic of SPH is the capacity of adaptability to an arbitrary bulk set of particles at each time step. This converts SPH in a natural method to solve problems with important deformations like impacts, and explosions.

SPH presents an harmonic combination of the Lagrangian formulation and particle approximation. SPH particles move according to internal and external forces carrying material properties.[108] All of these quantities evolve according to governing equations, which are written in term of fluxes between particles.[108, 45]

A function $f(\mathbf{r})$ where $\mathbf{r}$ is the position in the domain can be written as

$$f(\mathbf{r}) = \int_{\Omega} f(\mathbf{r}')\, \delta(\mathbf{r} - \mathbf{r}')\, d\mathbf{r}' \tag{2.18}$$

In Smoothed Particle Hydrodynamics any function f can be written as a convolution product with an interpolation kernel function $\omega(\mathbf{r}, h)$ where h is the smoothing length

$$f(\mathbf{r}) = \int_{\Omega} f(\mathbf{r}')\, \omega(\mathbf{r} - \mathbf{r}', h)\, d\mathbf{r}' + O(h^2) \tag{2.19}$$

The approximation of the equation 2.19 can be written as

$$f(\mathbf{r}) = \sum_b \frac{m_b}{\rho_b} f(\mathbf{r}_b)\, \omega(\mathbf{r} - \mathbf{r}', h)\, d\mathbf{r}' + O(h^2) \tag{2.20}$$

where b is referring to each particle in the present domain, and $d\mathbf{r}'$ has been replaced by the volume m_b/ρ_b of the particle b. If the error is dropped, and considering the case of a spherical kernel (i.e. $\omega(\mathbf{r}, h)$ only depends on the distance between particle pair) a typical 2D Kernel function in 2D looks like the one shown in Figure 2.8.

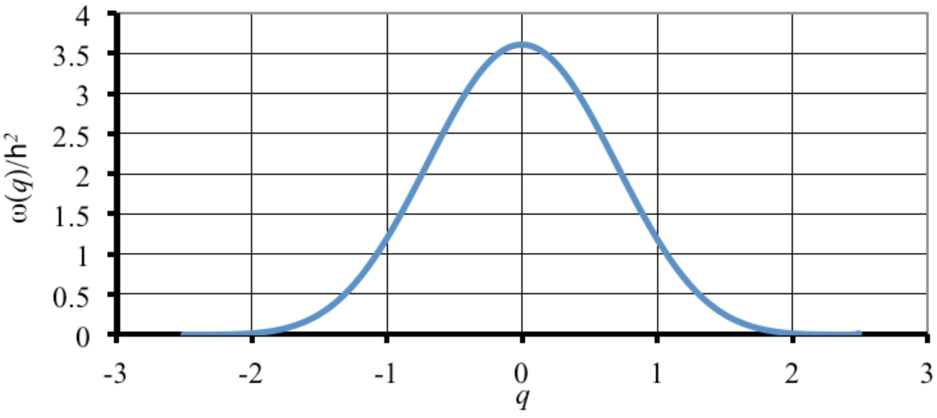


Figure 2.8: 2D Kernel function.

The Figure 2.9 shows a particle a and the nearest particles within a radius h_r that contributes to the kernel of a.

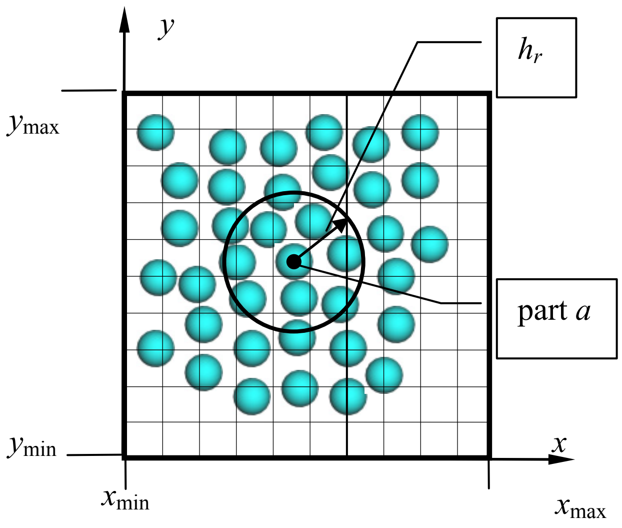


Figure 2.9: Particle a and all the nearest particles that contribute to the kernel of a.

Chapter 3

INTRODUCTION TO THE COMBINED FINITE-DISCRETE ELEMENT METHOD

3.1 Introduction

In this chapter different aspects of the FDEM method are introduced, such as; contact detection, deformation, joints and parallelization. DEM is one of the most significant current developments[142] in computational mechanics. During the past 40 years this method has been adapted to solve increasingly complex problems. From its humble beginnings where the particles were simple rigid bodies[34, 8282], to more complex algorithms able to deal with deformable particles.[133, 141, 129] There are a variety of methods, where the particles are allowed to deform under internal and external forces. The simplest of them rely on 2D circles that can deform/overlap depending on the loads.[92, 157] Rattanadit[92] investigated the "Dynamic Analysis of Granular System In Bending" coupling DEM particles with FEM to simulate an elastic container.[1] One of the limitations of this method is the absence of complex behaviour on the particles.

If a more complex behaviour of each particle is required, the natural thing to do is to discretise them into smaller finite elements, and then "glue" them together to simulate the original particle. The method that allows this is called The Combined Finite-Discrete Element Method. It was developed in the mid 90s[145] by Ante Munjiza.

For the FDEM method different finite elements (FEs) have previously been proposed such as beam element,[13, 137] linear triangle, tetrahedra,[133] quadratic tetrahedra,[206]

[1]Do not confuse with FDEM method

shell,[150] etc. All these FEs, share the same main issues[157,206,141] that have to be solved in order to simulate physical problems:

1. Particle deformation
2. Contact interaction
3. Contact detection
4. “Joints” between particles / Fracture
5. CPU time / parallelization

Some of this issues are shown in Figure 3.1.

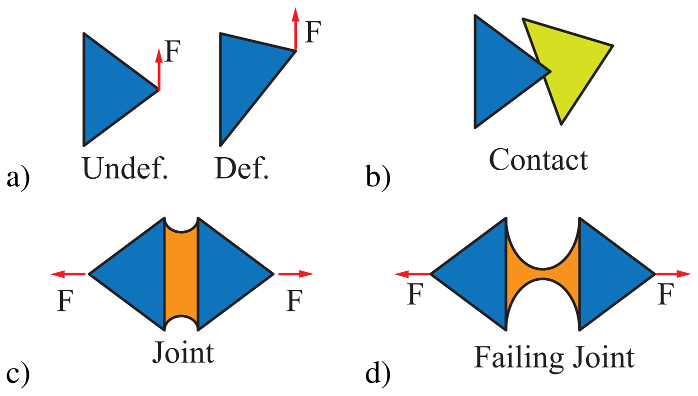


Figure 3.1: Some typical issues in FDEM. a)Deform. b)Contact. c)Transmission of forces between independent entities (FEs). d)Fracture

3.2 Equations for the Discrete Element Method

The governing equation[194,53] for a rigid discrete element is:

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{C}\dot{\mathbf{u}} - \mathbf{F}_{ext} - \mathbf{F}_{con} = \mathbf{0} \tag{3.1}$$

where $\ddot{\mathbf{u}}$ is the acceleration, $\dot{\mathbf{u}}$ is the velocity, $\mathbf{M}$ is the mass matrix, $\mathbf{C}$ is the damping matrix, $\mathbf{F}_{ext}$ is the external force vector and $\mathbf{F}_{con}$ is the contact (interaction) force vector. If the particle is not rigid a new term has to be added to the previous equation

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{C}\dot{\mathbf{u}} + \mathbf{F}_{int} - \mathbf{F}_{ext} - \mathbf{F}_{con} = \mathbf{0} \tag{3.2}$$

where $\mathbf{F}_{int}$ is the internal force vector. The equation 3.2 can be applied to a variety of situations, however is not enough to describe more complex phenomena. For the FDEM method it is necessary to add one more term to the equation 3.2, as the particles are "glued" between each other through joints, as shown in Figure 3.1c. The resulting equation is calculated as

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{C}\dot{\mathbf{u}} + \mathbf{F}_{int} - \mathbf{F}_{ext} - \mathbf{F}_{con} - \mathbf{F}_{jnt} = \mathbf{0} \tag{3.3}$$

where the new term $\mathbf{F}_{jnt}$ is the joint force vector.

3.3 Contact detection

Contact detection (CD) is one of the most important aspects of the DEM Method. It is the process of finding all interaction couples i, j at a particular time and in some cases could take more than 60% of the total CPU time.[133] These couples can be made of objects that are currently interacting or that may interact in n_{CD} steps as shown in Figure 3.2.

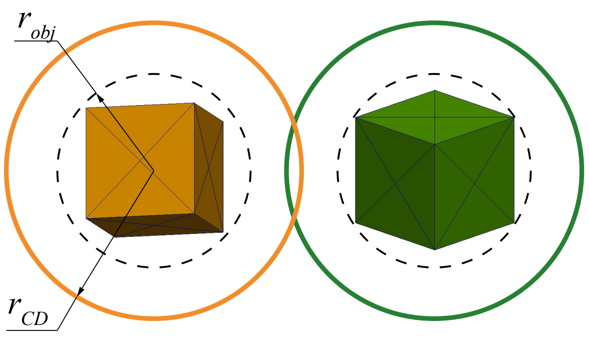


Figure 3.2: Two discrete elements that may interact in n steps.

If the RAM memory is not a limitation, the contact couples can be saved on a data base (arrays, list, binary tree, etc.), avoiding the need for performing contact detection at each time step, as shown in Figure 3.3. There is always a trade-off when the contacting couples are saved as more couples will increase the interaction time (computing false contacts).

The size of each element, for the purpose of CD, is increased by the maximum distance that any object can travel in n_{CD} steps. This distance is defined as $\triangle_{buf}$ and takes into account all the possible positions where the element can travel. The value of r_{CD} is calculated as

$$r_{CD} = r_{obj} + \triangle_{buf} \tag{3.4}$$

and $\triangle_{buf}$ is given by

$$\triangle_{buf} = n_{CD} v_{max} \triangle t \tag{3.5}$$

where r_{obj} is the radius of the object, n_{CD} represents the number of steps between contact detection, v_{max} is the maximum velocity of any object, and $\triangle t$ is delta time. The value of n_{CD} is obtained basically by trial and error. On the limit of $n_{CD} \rightarrow \infty$ the buffer $\triangle_{buf} \rightarrow \infty$ and the data base is no more than a quadratic interaction, of everybody interacting with everybody.

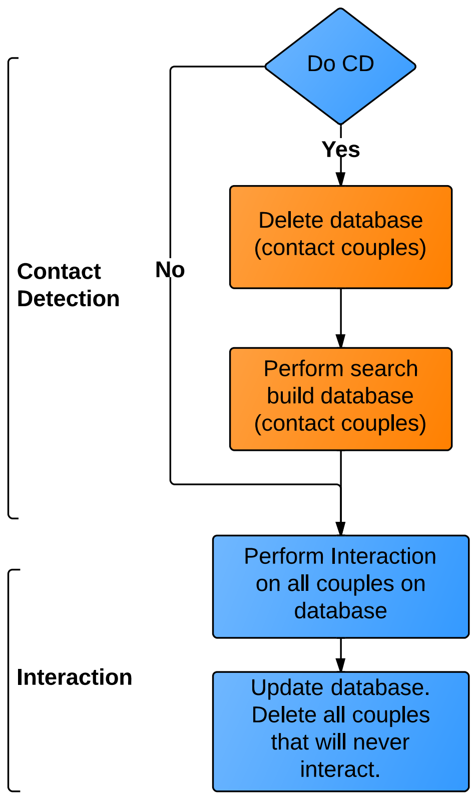

Do CD
Yes
No
Delete database (contact couples)
Perform search build database (contact couples)
Contact Detection
Perform Interaction on all couples on database
Update database. Delete all couples that will never interact.
Interaction

Figure 3.3: Flow diagram CD and interaction using a database of contact couples.

The Direct Search (DS), also known as quadratic search, is shown on the Algorithm 3.1. As its name states, the time increases quadratically with the number of elements as

$$T \propto N^2 \tag{3.6}$$

where T is the time, and N the total number of elements. In general this algorithm is impractical for dynamic simulations with any significant amount of elements. On the other hand, for semi-static simulations with few searches are performed and contact couples stored it could be an alternative, avoiding the need to implement a more sophisticated algorithm.

Algorithm 3.1 Direct search (DS)

```
1: integer $iLv\_N$                                   ▷ Total number of elements
2: integer $iLv\_i$, $iLv\_j$                           ▷ Element $i$, element $j$
3: boolean $qLv\_con$                                 ▷ Contact
4: for $(iLv\_i = 0; iLv\_i < iMv\_N; ++iLv\_i)$ do
5:     for $(iLv\_j = iLv\_i + 1; iLv\_j < iMv\_N; ++iLv\_j)$ do
6:         $qLv\_con = qS\_ConDet\,(iLv\_i, iLv\_j)$
7:         if $qLv\_con$ then
8:             $vS\_Interaction\,(iLv\_i, iLv\_j)$
9:         end if
10:    end for
11: end for
```

For general DEM simulations, the quadratic search imposes a limitation on the number of objects that can be simulated. Different algorithms have been proposed in the past to improve the CPU time of the quadratic search, namely Alternating Digital Tree(ADT),[18] 3D-DDA,[16] MR[148]. The main improvements in CD were made by Munjiza[136] in the mid 90s, who developed the first linear CD algorithm. It was called Non Binary Search-Munjiza (NBS), where the time is given by

$$T \propto N \tag{3.7}$$

In the Chapters 4, 5, 6 and 7, different issues related to CD are explained in detail.

3.4 Deformation 3D

An object subject to external forces will experience internal forces.[162] Each small volume of this body will experience in general a combination of normal stresses σ_i and shear stresses τ_{ij} as shown in Figure 3.4

To describe the stress at a point inside the body, it is necessary to know three normal stresses σ_x, σ_y, σ_z and six shear stresses τ_{xy}, τ_{yz}, τ_{zx}, τ_{yx}, τ_{zy}, τ_{zy}. These form the stress tensor matrix $\mathbf{T}$[115, 80]

$$\mathbf{T} = \begin{vmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{vmatrix} \equiv \begin{vmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{vmatrix} \equiv \begin{vmatrix} \sigma_{1} & \tau_{12} & \tau_{13} \\ \tau_{21} & \sigma_{2} & \tau_{23} \\ \tau_{31} & \tau_{32} & \sigma_{3} \end{vmatrix} \tag{3.8}$$

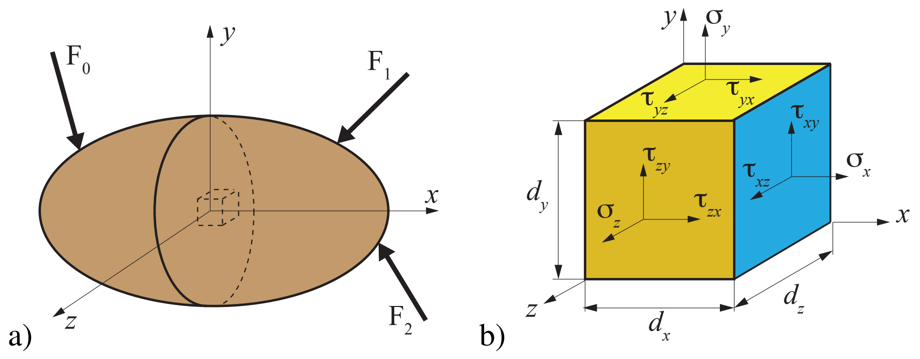


Figure 3.4: a)General body and the external forces acting on it. b)General stresses acting on a small region of the body, with volume $=d_x d_y d_z$. Figure adapted from Parker[162] and Collins[33]

The Cauchy's Stress Theorem, states that the stress vector $\mathbf{t}$ on a point with a normal $\mathbf{n}$ is obtain by[80]

$$\mathbf{t} = \mathbf{T} \cdot \mathbf{n} \tag{3.9}$$

Forces on a Tetrahedron for FDEM. One of the most simple and well documented finite element (FE) for the Finite Element Method (FEM) is the tetrahedron.[110] Countless open source (OpenFoam, Code-Aster, etc.) and commercially (Ansys, ABAQUS, etc.) programs already work with tetrahedra. This is a great advantage when complex 3D objects have to be meshed and boundary conditions set.

The capacity of simulating complex and non-trivial phenomena in FDEM demands robust yet not intensive algorithms. In his book Munjiza[133] developed a formulation for the constant strain tetrahedron finite element. The detailed explanation of the method is beyond the scope of this work.

The solution proposed by Munjiza, is for small strains. It is based on the Green-St Venant tensor, with the Hook's law for the relationship between stress and strain. The stress tensor $\mathbf{T}$ in global coordinates is given by

$$\mathbf{T} = \frac{E}{(1+\nu)} \frac{1}{(|\mathrm{det}\mathbf{F}|)^{2/3}} \tilde{\mathbf{E}}_d + \frac{E}{(1-2\nu)} \frac{1}{(|\mathrm{det}\mathbf{F}|)^{2/3}} \tilde{\mathbf{E}}_s + \frac{2\overline{\mu}}{(|\mathrm{det}\mathbf{F}|)} \mathbf{D} \tag{3.10}$$

where E is the elastic modulus, ν is the Poisson's ratio, $\mathbf{F}$ is the deformation gradient matrix, $\tilde{\mathbf{E}}_d$ is the Green-St Venant matrix due to shape change, $\tilde{\mathbf{E}}_s$ is the Green-St Venant matrix due to volume change, $\bar{\mu}$ dissipative damping of the material and $\mathbf{D}$ is the velocity gradient matrix .

The stress on each face of the tetrahedron can be calculated using the equation 3.9. Replacing the normal $\mathbf{n}$ by $\mathbf{m}$ (half of the cross product of the triangle that form the surface), the force on the surface is

$$\mathbf{f}_{srf} = \mathbf{T}\,\mathbf{m}_{srf} \tag{3.11}$$

Finally the force to add to each of the nodes of the surface is

$$\mathbf{f}_{nod} = \frac{\mathbf{f}_{srf}}{3} \tag{3.12}$$

3.5 Fracture 3D

The joints shown in Figure 3.1c and Figure 3.1d, transmit the load from one finite element to its neighbours. In FDEM formulations fractures are only produced at joints as the finite elements cannot fracture. Fracture is an open fill in constant development where new materials impose the need of new solutions[55] such as multiscale fracture.[83]

The solution employed in this work is based on the smeared fracture model, developed by Munjiza.[133, 143, 140, 99, 135] The model takes into account fractures of Mode I and Mode II, shown in Figure 3.5.

The fracture model is based on the rock fracture under tension from which a typical stress-displacement curve is shown in Figure 3.7. For any stress less than the tensile strength f_t there is no damage and all the energy stored as potential energy is restored to the system. This is hightailed in Figure 3.7a. For any stress bigger than the tensile strength a fracture will be produced, and the joint will start to fail on the softening branch, as shown in Figure 3.7b. The area under the softening branch is equal to the energy release rate G_f[135]. In FDEM this is "implemented through the single crack model"[133] shown in Figure 3.6.

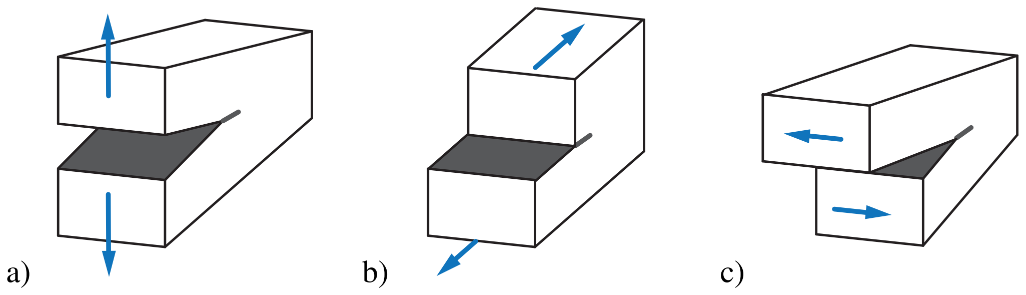


Figure 3.5: Fracture Modes. a)Mode I. b)Mode II. c)Mode II. Figure adapted from Collings[33]

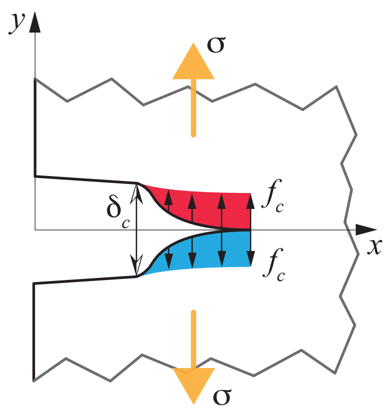


Figure 3.6: Crack model: colour represents the bounding stress. Figure adapted from Munjiza[133]

In theory should be no separation before the tension f_t is reached. In the single crack model there is always a small separation even before f_t is reached. This separation and the displacement δ_t depends on the penalty of the joint.

In this work joints cannot fail in pure tensile compression. The relationship between the compression displacement and the force is linearly proportional to the joint's penalty as shown in Figure 3.8a

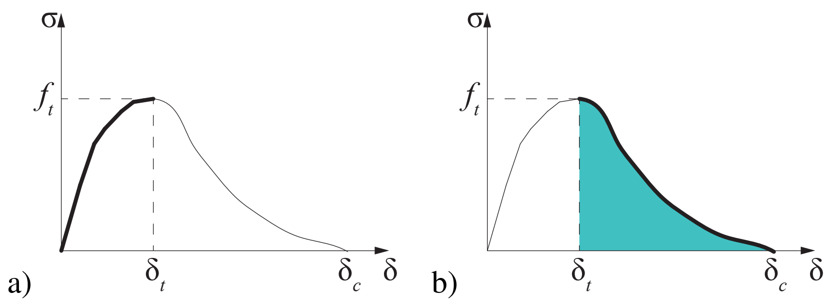


Figure 3.7: Stress curve defined as function of displacements. a)Hardening arm highlighted. b)Softening arm highlighted. The area under the curve is hightailed in light green, and is equal to the fracture energy G_f. Figure adapted from Munjiza[133]

In general the fracture process will be produced in steps, with the displacements positive and negative. One such process is shown schematically in Figure 3.8b. Where the curve is walked in A, B, C, C, D order: at the end of the curve A ($\delta = \delta_t$), the pick stress f_t is reached, and any further displacement will produce permanent damage. At the end of B, the displacement is lowered to zero on the curve C. It is worth to mentioning, that for any displacement on the curve C $\delta \leq \delta_{\mathrm{CB}}$ there is no further loss of energy. Finally the displacements are increased on C, until $\delta = \delta_{\mathrm{CB}}$ is reached. As the displacement increases the joint fails on the curve D.

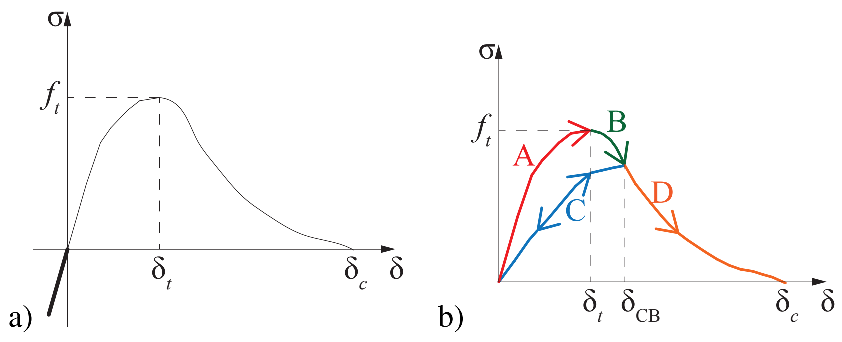


Figure 3.8: Stress curve defined as function of displacements. a) Compression arm is highlighted. b)The curve is “walked” in A, B, C, D order.

The value of the stress σ in the softening branch for any displacement $\delta > \delta_t$ is calculated as

$$\sigma = z f_t \tag{3.13}$$

where z is a mathematical model based on experimental data for concrete failure under tension[133] and is calculated as

$$z = \left[1 - \frac{a+b-1}{a+b} \mathrm{e}^{D\left(a+cb/((a+b)(1-a-b))\right)}\right] \left[a\left(1-D\right) + b\left(1-D\right)^{c}\right] \tag{3.14}$$

where a, b, c are parameters chosen to fit a particular material and D is the damage given by

$$D = \begin{cases} 0 & \text{if} \quad \delta \leq \delta_t \\ 1 & \text{if} \quad \delta > \delta_c \\ \frac{\delta-\delta_t}{\delta_c-\delta_t} & \text{otherwise} \end{cases} \tag{3.15}$$

The next example illustrates the main attributes of the joint where two regular tetrahedra, shown in Figure 3.9, have their position imposed function of time. The left tetrahedron is fixed throughout all the simulation, while the right tetrahedron moves in different directions to simulate pure tensile fracture, pure shear fracture, and a combination between tensile and shear failure.

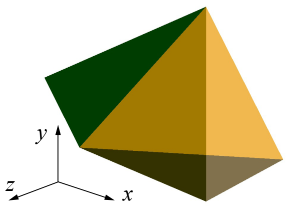


Figure 3.9: Two regular tetrahedron, of A=173.205 mm^2

The material properties of the joint are $f_t = f_s = 5$ MPa, fracture energy $G_f = G_t$ =30.0291 N/m, joint penalty 26.6 GPa. The parameters for the curve z are $a = 0.63$, $b = 1.8$ and c=6. The area under the softening branch z is given by

$$A_{sof} = \int_{D=0}^{D=1} z(a,b,c,D)\ dD = 0.387974 \tag{3.16}$$

and the displacement in the softening branch are

$$\triangle_t = \delta_c - \delta_t = \frac{G_f}{A_{sof} f_t} = 15.48\mu\text{m} \tag{3.17}$$

$$\triangle_s = \delta_c - \delta_s = \frac{G_f}{A_{sof} f_s} = 15.48\mu\text{m} \tag{3.18}$$

and the maximum forces are given by

$$\mathbf{F}_t = A f_t = 866.025\,\text{N} \tag{3.19}$$

$$\mathbf{F}_s = A f_s = 866.025\,\text{N} \tag{3.20}$$

The forces for the Mode I, Mode II and the combined Modes I and II are shown in Figure 3.10. The comparison between expected results and the test are shown in table 3.1.

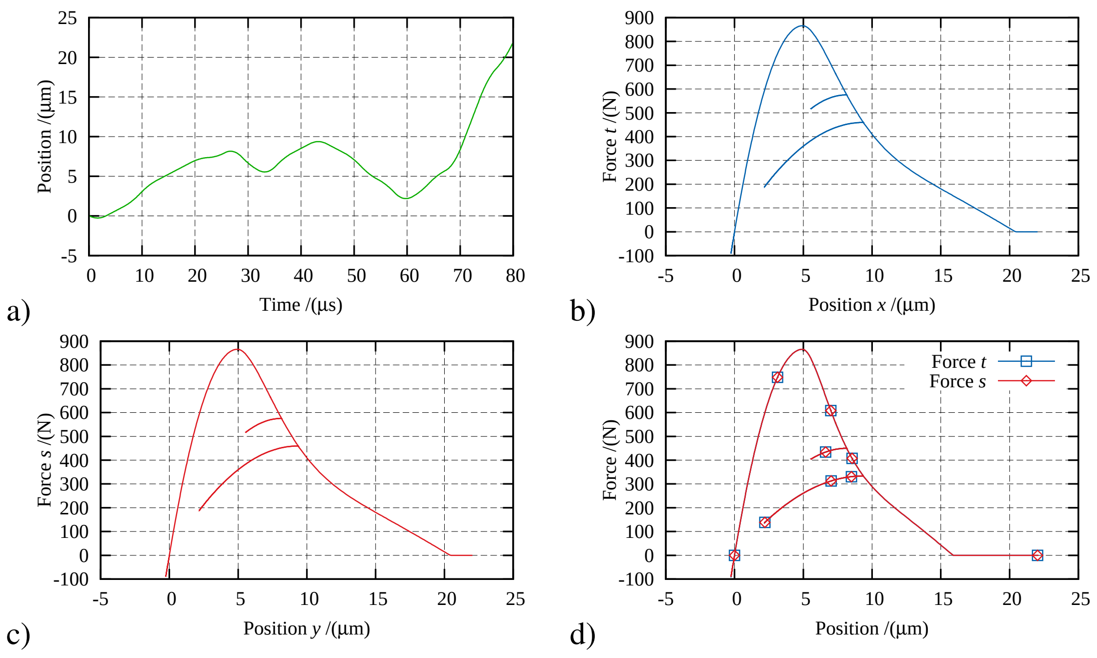


Figure 3.10: Failure modes. All forces are scaled by -1. a)Position function of time. For mode I is position x, for mode II is position y, and for the last case is position x and y. b)Mode I, total force x. c)Mode II, total force y. d)Mode I and Mode II, total force x (normal) and y (tangential)

Case	Expected	Test
Mode I: $\mathbf{F}_t$ (N)	866.03	866.40
Mode I:$\triangle_t$ (μm)	15.48	15.45
Mode II: $\mathbf{F}_s$ (N)	866.03	866.02
Mode II:$\triangle_s$ (μm)	15.48	15.56
Mode I & II: $\mathbf{F}_t$ (N)	866.03	866.02
Mode I & II: $\mathbf{F}_s$ (N)	866.03	866.02

Table 3.1: Joint test

3.6 Parallelization

The modern definition of Computational Mechanics started with the creation of the transistor in the earlier 70's, and relied upon a continuous growth of performance. The three pillars of the transistor speed have been:[19]

- **Scaling:** By scaling the transistor, the performance increases, and the demand of energy is reduced.

- **Micro-architecture:** By rearranging the transistors, “pipelining”,[19] “branch prediction”,[19] etc.

- **Memory:** Using more than one level of cache memory, reducing the delay time between the processor and the memory.

These advances in processor speed have not been matched in the memory architecture. Differences between processor and the Dynamic Random Access Memory (DRAM) are substantial as shown in Figure 3.11. This has led to the use of more than one level of cache to compensate for the DRAM velocity.[19]

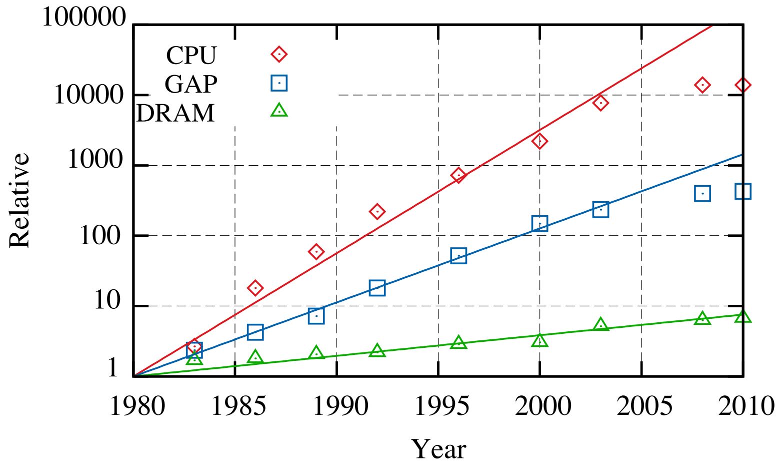


Figure 3.11: Relative comparison of velocities between CPU and DRAM. Figure adapted from Borkar et al.[19] .

CPU velocities have not changed significantly in the past few years and will not change that much in the near future.[1] There is not a single reason for this, but a series of complex factors acting together. Reducing the size of the transistor, is not the simplest method for increasing the velocity, as was the case until the early 21st century. As the size decreases the voltage at which the transistor start to conduct (the threshold voltage) decrease as well. This decrease in the threshold voltage increases exponentially the amount of leakage on it, increasing the power consumption.[19] The main operating voltage of the transistor is also “not expected to change in the near future”.[15]

Another of the reasons for this slowdown in the velocity, is the amount of power consumption. Simply adding more cores and making then work at the maximum frequency will produce high power demanding processors.[19] The total power that can be economically dissipated per die is around 100+ watts.[15] Once that has been reached another way to reduce the power is to slow down the clock speed.

3.6.1 Hardware processors

As previously mentioned, we are facing a time where it is no longer possible to expect the next generation of processors to outperform the previous generation in clock speed. Demand for more computational power has led to hardware customisation of big server centres in companies like Google, Facebook, HP and Dell. Even open hardware projects are starting to gain momentum[2]. Servers are power demand structures, and in some cases the kind of tasks that are performed are restricted. It makes sense to customise the processors, switching off some of the capacities, and increasing others, saving energy and increasing performance.[4]

Not all applications require extreme computational velocities and interconnected servers. For applications that can be split into separate independent tasks such as the SETI[167] project, a "colossal amount of computing power could be assembled",[94] building what is in fact a "super computer" with desktop CPU velocities.

Regardless of the path chosen to obtain more computational power, all solutions will rely upon one or more of the three main groups of hardware processors:

- Graphic Processor Unit (GPU)
- Central Processor Unit (CPU)
- Low power processors

In recent years, a new class of low power processors, such as ARM, have become more important as their use in mobile devices becomes more and more common.

GPU. These processors have been used and driven mostly by the gaming industry and its continuous demand for more powerful processors. Nowadays almost all desktop computers have one GPU processor unit.[158]

Before the mid 2000s , the main applications of GPU processors have been for the calculus of graphics. The classical "stream programming model"[180] used by the GPU is showed in Figure 3.12. In this model, each processing core, reads a well-defined area in the memory and produces the same size output result. Other more complex models have been propose in recent years with variable size input and non standardised size output.[180]

[2]Projects like OpenCompute

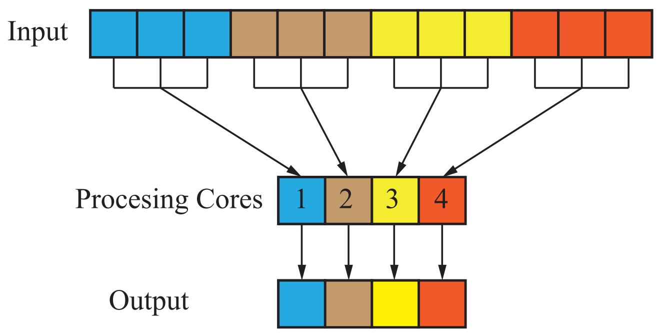


Figure 3.12: GPU stream programming model. Figure modified from Sengupta et al.[180]

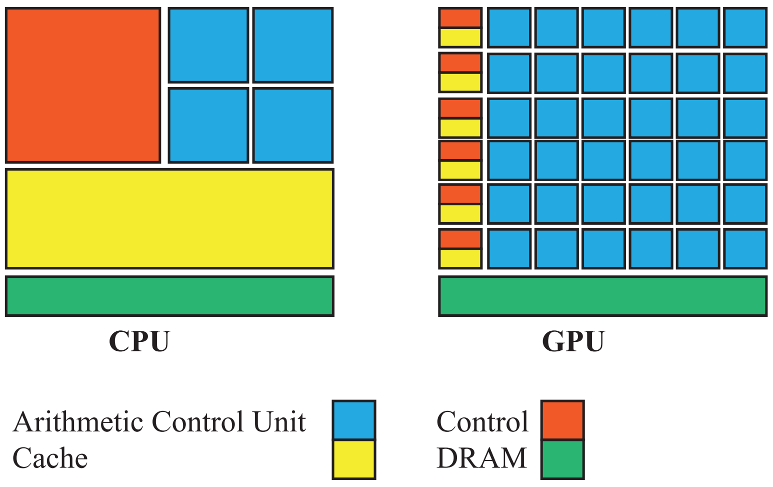


Figure 3.13: GPU and CPU philosophy design. Figure modified from Nvidia[155] .

What differentiates between CPU and GPU, or we could say between Multicore and Many-Core processors are the problems they are designed to solve. On one hand Multicore processors are expected to perform well while running legacy sequential codes but, more importantly, these multicore processors possess non-simple controls, that “allow single thread to execute in parallel or even out of order while maintaining the appearance of sequential execution”.[91] Meanwhile, Many-Core processors are designed to run simple parallel applications.[91] They are “single-instruction-issue”[196] and posses “simpler memory models”.[91] This different philosophy is shown in Figure 3.13.

With the development of program tools like CUDA[91,153] and OpenCL,[155,151,24] the scientific community is taking advantage of these powerful processors.[170]

The simplicity of GPU with respect to CPU, ensures relative speed-ups but at the same time increases the complexity of code, as the program has to be structured into calculus blocks[158,19] to take advantage of this power.

CPU. The time we were expecting each generation of processor to be faster than the previous one, doubling the clock speed every 24 months, stopped almost decade ago.[15] The Moore law, establishes that the increase in the number of transistors/area, will continue to happen in the future,[15] but does not have a relationship with the velocity of the chips.

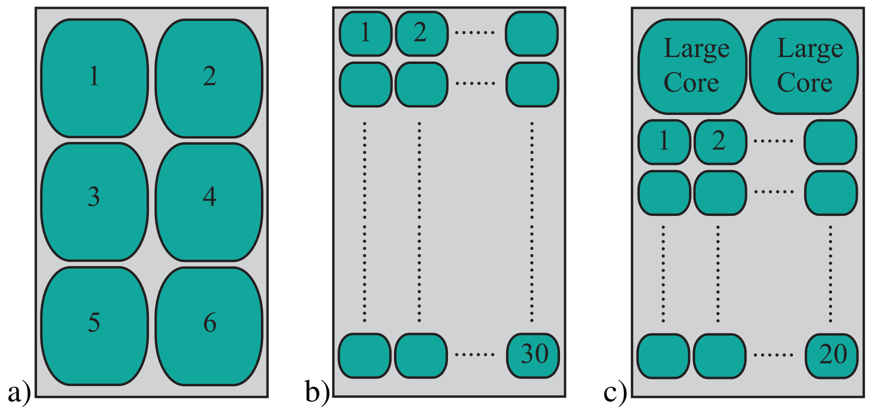


Figure 3.14: Arrangement of cores. a)6 equal cores. b)30 smaller cores. c)A mix of the previous solutions. Figure modified from Borkar et al.[19]

"Energy-proportional computing"[19] is the last aim of chip designers. One big single core is energy inefficient, and the evolution of processors is moving into multicore. Non-parallel programs cannot benefit from multicore processors, but parallel programs can. In theory a parallel program running in 2 processors should increase its velocity by 2x, and so son. Borkar et al.[19] proposed three different homogeneous arrangements for an hypothetical 150 million transistor. A six large cores, really good for single thread applications with an theoretical 6x increase in parallel is shown on Figure 3.14a. 30 smaller cores, better for parallel applications is shown in Figure 3.14b. Two larges cores and 20 smaller cores are a combination of the previous cases is shown in Figure 3.14c.

In a new era where the power used is defined by features in chip design, new revolutionary solutions have to be implemented in order to speed up applications. Parallelization is the most commonly solution proposed but alone it will be not enough. The traditional approach of 90/10, which designs the chip for the 90% of the cases, is facing power barriers.

Chien et al.[27] proposed a new approach. In their 10x10 design proposal, the most common tasks should be packed into 10 different groups. For these groups a custom design micro-engine on the chip will run the specific task. The main advantage is there will be about 10x saving in energy for each domain. This proposal is taking what is already happening in the silicon industry one step further, as the interaction of these micro engines are made in a way that the data is shared in the L1 cache, speeding up the access to the memory. The idea is that a particular application may use one or more engines at the same time, saving energy by using only what is needed, therefore using particular engines efficiently. This is shown schematically in Figure 3.15.

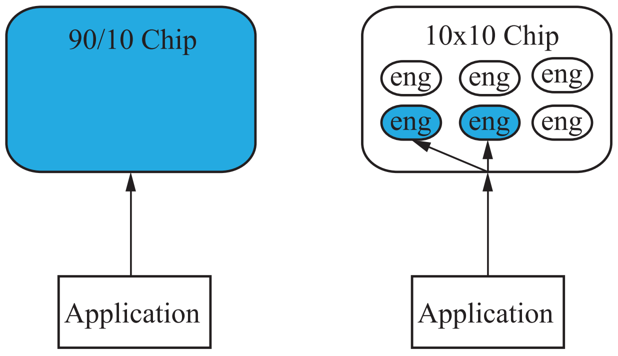


Figure 3.15: Schematic view of a traditional 90/10 chip (right), and the proposed the 10x10 chip (left).

Low Power Chips. The key for the future may be in low power chips such as the ARM and the Epiphany[3]. The most common of these are the ARM. ARM have been designed from the beginning with power consumption as one of the main features. Companies like Apple, Nokia, Samsung and many more are using ARM chips for their flagship products.

The most interesting characteristic of the ARM is not in the type of products where they can currently be found but the ones that are coming to the market. For the first time one of the biggest companies in the server segments, HP, is using really low power chips in its servers,[77] using Calxeda flavour of ARM in its project Moonshot Servers[78] which is expected to deliver

- 89% less energy
- 94% less space
- 63% less cost

Canonical with its product Ubuntu server is supporting ARM technologies and in particular these servers from HP.[10] These have implications not only for the Internet Providers, but for the scientific community who can benefit in the near future from running

[3]Epiphany (from Adapteva) chips have been designed with multiprocessing as their main characteristic.

simulations on less expensive servers. A small step in this direction demonstrated by a cluster[3] made by Cox in the university of Southampton using 64 RaspberryPi (ARM computer) computers to run MPI simulations.

3.6.2 The computer as heterogeneous system

Today, computers can be analysed as an heterogeneous mix of different systems,[151] as shown in Figure 3.16

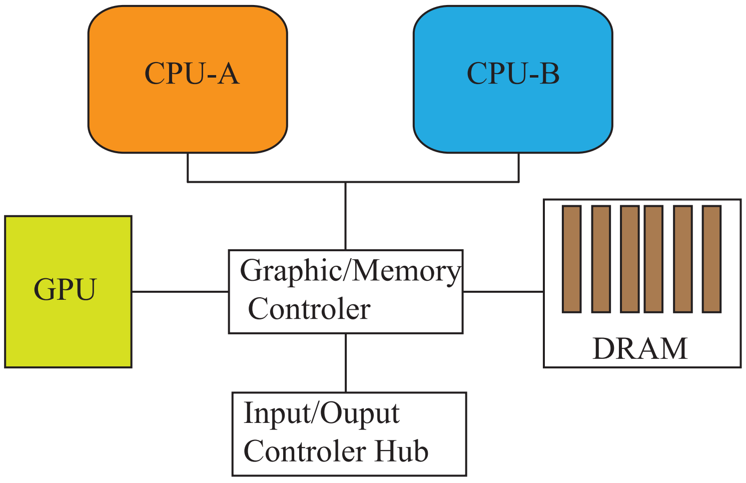


Figure 3.16: Heterogeneous system, with two CPU units, that may be different, and one GPU unit. Figure modified from Munshi et al.[151] .

It is expected that in the future, applications will take advantage of these two particular design processors, running part of the application on the CPU and other parts on the GPU.[91] CUDA[91,153] and OpenCL[24,39,155,151] allow for the develop of software that seamlessly uses CPU and GPU.

This heterogeneity is becoming a common word in todays computers. There are already some chips that introduce “heterogeneity” in their design such as the Intel Sandy Bridge. Future developments have to take them into account.

3.6.3 Hardware memory

Today’s computers can be divided in two main groups from a memory point of view.[160] These groups are shown in Figure 3.17.

The first group, called is called “Share Memory”, is encountered in Multicore Computers. In this group each processor has the ability to read and write all the memory. OpenMP provides tools to code for these systems.

The second group is called “Distributed Memory”, and is the typical encounter in a High Performance Computing (HPC)-Cluster. Nowadays clusters have nodes with more than one processor, making them hybrid systems.[160] Each node can only access other node’s memories through the network. A typical language to program for Distributed Memory is MPI.

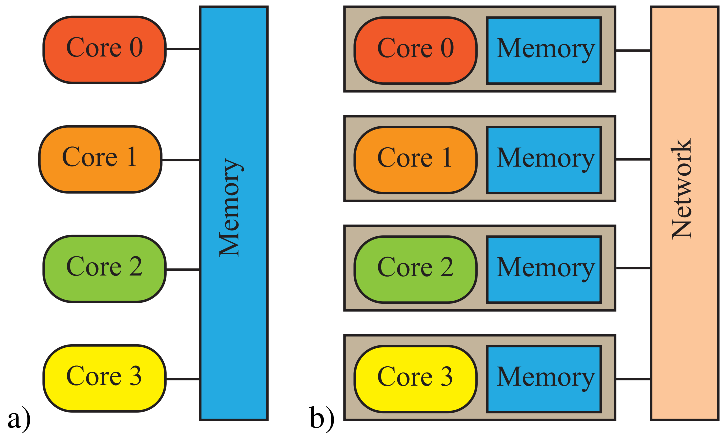


Figure 3.17: Memory distributions. a)Share Memory System b)Distribute Memory System with 4 processors (each processor could be a multicore processor). Figure modified from Pacheco[160]

3.6.4 Software: parallel language programming

As the increase in transistor speed of single processors, is slowing down[19] with each new generation, the development of new software is shifting to parallel methods.

Different solutions have been proposed and implemented[160,71] to speed up the calculus of scientific and engineering programs. With the revolution of the last 30 years in hardware more and more programs are being developed taking into account this new CPU power, and legacy codes are been adapted to take it. Depending on the hardware architecture there are different options to implement these solutions. On clusters[85,152] it is possible to code using MPI,[159,61] in Multicore computers OpenMP[25,168] and MPI, finally in many cores computers (GPU), CUDA[91,153] and OpenCL.

New tools have been developed in the last 5 years, such as the Open Computing Language[24,39,155,151] (OpenCL). Capable of compiling and running independently of the processor and platform. OpenCL was developed by Apple and NVidia and was presented to the non profit group Khronos in 2008. OpenCL allows a program to take the power of the CPU (single core, multicore) and GPU processors individually or as an interacting team.[59]

As multicore computers become more and more common, tailor-made solutions are needed. Holmes et al.[76,202] have developed a new port base technique for Multicores. The new algorithm is called H-Dispatch. The size of the *domain* is much smaller than that the traditionally used in MPI. These smaller domains and the careful use of the shared memory allows the distribution of workloads in a much more efficient way. Not based on the the spatial distribution of objects into domains (classical parallel method), but on the workload of the processors, as shown in Figure 3.18.

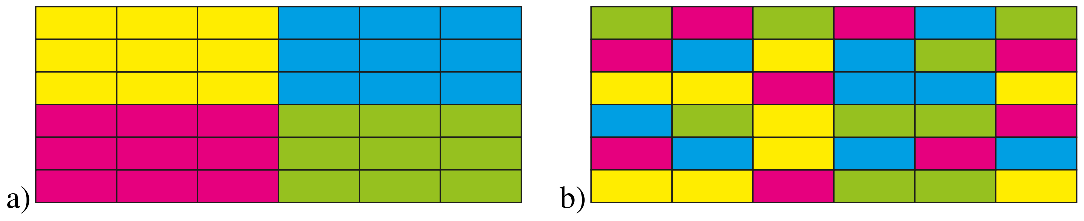


Figure 3.18: Each colour represent the domain analysed by a particular processor. a)Classical, domain decomposition. b)Fine domain decomposition. Figure modified from Holmes et al.[76]

All these different solutions, called explicit-parallel, share the same intrinsic set of problems, as the amount of effort and knowledge needed by the programmer is great. The programmer not only needs to understand the physics of the problem to solve, but how a particular hardware operates. The time and money that needs to be invested in developing new solutions can be prohibitive.

The cost to develop and debug a parallel solution for FDEM (to this author's knowledge) is at least three to four times higher when is compared to similar sequential solution. The main question is where the future lies and as the programs become more and more complex the amount of resources and money required increases. Usually porting one parallel application from one kind of hardware to other one is not a straightforward task, and some of the optimisations have to be painstakingly repeated. As the programs increase in complexity and cost, the future may lay in Implicit Parallel Programming. This hides most of the complexities from the programmer, allowing the development of more complex programming, as well as reducing the test and debugging times.[79,196]

All the different parallel programming languages could be grouped into three main categories:[187]

- **Explicit:** Has the advantage of tailor-made solutions, as the programmer takes full control of the software and the hardware that will be running the application. For example MPI, OpenMP, etc

- **Implicit**: Hides the complicity "inside libraries or APIs",[187] for the programmer.

- **Automatic parallelization**: This works by *reading* a non parallel application written for example in Fortran, and transforming it into a parallel application. It is not yet really working.

3.6.5 Some parallel general considerations.

Speed-Up. The most common factor, utilised to measure the performance of a parallel program, is called the “speed-up factor”. It is equal to

$$S = \frac{t_{seq}}{t_{par}} \tag{3.21}$$

where t_{seq}, is the total time employed by the the best sequential algorithm,[160, 198, 188] and t_{par} is the total time used by the parallel algorithm. In the practice, t_{seq} usually is the algorithm upon which the parallelization was made.[160]

The total time t_{par} is equal to

$$t_{par} = t_{exe} + t_{com} + t_{upd} \tag{3.22}$$

where t_{exe} is the execution time, t_{com} is the communication time between processes[4] and t_{upd} is the update time of the objects affected by the parallelization. Replacing t_{par} in the equation 3.21with the equation 3.22 then

$$S = \frac{t_{seq}}{t_{exe} + t_{com} + t_{upd}} \tag{3.23}$$

if $t_{exe} \approx {}^{t_{seq}}/_{n_{proc}}$, where n_{proc} is the number of processes then

$$S = \frac{t_{seq}}{{}^{t_{seq}}/_{n_{prc}} + t_{com} + t_{upd}} \tag{3.24}$$

For an ideal parallelization algorithm $t_{com} = 0$ and $t_{upd} = 0$ and t_{par} is equal to

$$t_{par} = t_{parIdeal} = \frac{t_{seq}}{n_{pro}} \tag{3.25}$$

[4]Depending on the library more than one process may reside in one processor

Replacing the equation 3.25 into the equation 3.21, leads to the maximum speed up factor equal to

$$S_{max} = \frac{t_{seq}}{t_{parIdeal}} = \frac{t_{seq}}{{t_{seq}}/{n_{pro}}} = n_{proc} \tag{3.26}$$

This means that the maximum speed-up factor is linear and equal to the number of processes. For some cases S could be bigger than S_{max} these cases are defined as "superlinear"[198] . This may be due to, poor sequential algorithms, hardware issues, or a particular behaviour of the algorithm.[198]

One important characteristic of S is its dependence on the number of objects that are simulated. This is best illustrated by the algorithm 3.2 where this algorithm is applied to two processes. In this simple algorithm there are two instances of communications, between the processes and in each of them updates have to be made to the system's data archive. As the total amount of objects N increases usually the communication ratio

$$R_{com} = \frac{N_{com}}{N} \tag{3.27}$$

where N_{com} is the total amount of objects to communicate, decreases. This implies that the ratio

$$R_{com\_t} = \frac{t_{com}}{t_{par}} \tag{3.28}$$

decrease as well. As most of the time is spent doing calculations and not communicating between different processes.

Algorithm 3.2 Simple parallel DEM algorithm for rigid bodies. "$\rightleftharpoons$" are operations performed with communication between processes.

```
while $stp < numStp$ do
    $Set\,zero\,forces$
    $Do\,contact\,detection\,\&\,interaction$
    $\rightleftharpoons$ $Update\,contact forces$
    $Update\,positions$
    $\rightleftharpoons$ $Send\,originals\,\&\,proxies$
end while
```

In the case of OpenMP, the communication between processors is faster than MPI as it takes advantage of the share memory architecture of the system, not having to copy, and delete data from the RAM. Nevertheless there have to be updates of the objects in the system. For a small number of objects the over-burn of the parallelization could be so high, that the parallel solution could end up being slower than the sequential, as

$$t_{com} + t_{upd} > \frac{t_{seq}}{n_{prc}} \tag{3.29}$$

Amdahl's Law. Parallelize a code is a costly and difficult task, that requires complete knowledge of how the sequential code operates. As not all the routines in a code are equally demanding in terms of CPU it would be really tempting, to identify the subroutine or the group of subroutines that are more demanding and decide to only parallelize just a portion of the code, leaving the rest to run sequentially. For an FDEM general code, it will make sense to parallelized the contact detection and interaction routines. The Figure 3.19 shows schematically how such an algorithm will look like for the algorithm 3.2.

One question that needs to be asked, however, is whether this new algorithm will improve the CPU times or not. Modifying the equation 3.24, to take into account the partial parallelization of the algorithm, leads to the equation:[160]

$$S = \frac{t_{seq}}{\frac{\alpha t_{seq}}{n_{proc}} + (1-\alpha) t_{Seq} + t_{com} + t_{upd}} \tag{3.30}$$

where α is the percentage of the algorithm that has been parallelized, and $0 \leq \alpha \leq 1$. For an ideal parallelization

$$S = \frac{t_{seq}}{\frac{\alpha t_{seq}}{n_{proc}} + (1-\alpha) t_{Seq}} = \frac{1}{(1-\alpha) + \frac{\alpha}{n_{proc}}} \tag{3.31}$$

and for infinite processes

$$S_{n_{proc} \to \infty} = \frac{1}{1-\alpha} \tag{3.32}$$

Even if 50% of an algorithm is parallelized and is run on a cluster of 1000 cores, the $S = 1.998$ and for an ideal cluster with infinite machines $S = 2.0$. Amdahl's Law[160] says that for a partial palletised code there is a limit to the amount of speed-up that can be obtained.

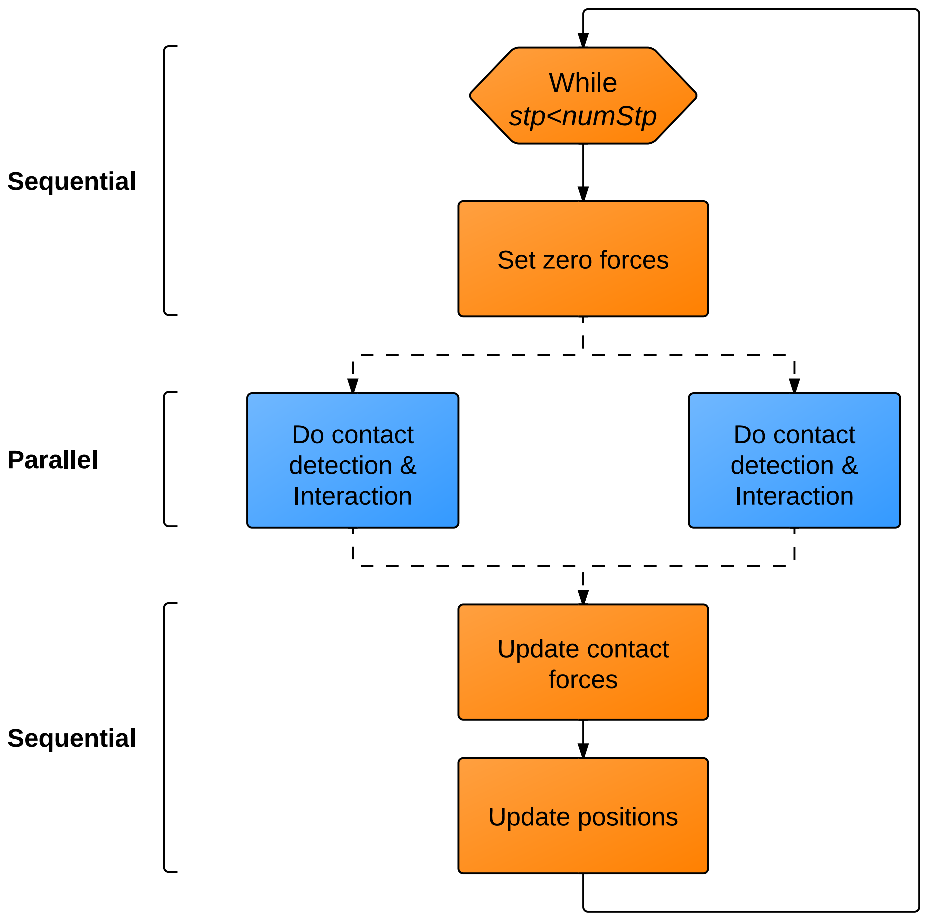


Figure 3.19: Partial parallelized algorithm.

Rounding Error (the evil of real numbers). It would be expected to obtain the same results in a sequential program, compared to a parallel version of the same program, as both programs use the same equations to solve the unknowns in the system.[112] And only the objects, that are on the boundary[5] between processes are affected by the updates of forces, positions, etc.

As is common in engineering, the results are not as expected. One of the reasons for this discrepancy, between sequential and parallel codes, lies in a well-known computational error called "rounding error". The representation of a real number in the computer is limited by the amount of memory dedicated to it. Just to represent the

[5]In the case of a parallelization with domain decomposition

number π would need infinite memory, as is for the representation of $1.0/3.0$. The error in the representation of a real number is limited by machine epsilon ε_M.

It would be tempting just to use the biggest precision possible for a particular combination of hardware-software. But more precision does not always means better results. One well documented example is the Rump's function[177]

$$f = 333.75b^6 + a^2\left(11a^2b^2 - b^6 - 121b^4 - 2\right) + 5.5b^8 + \frac{a}{2b} \tag{3.33}$$

where $a = 77617.0$ and $b = 33096.0$. The only errors in this functions are the rounding errors. It would be expected that as the precision increases the final result should approach the exact solution $f = -0.82739605994682_4^3$.[177] The results computed on a Intel workstation[132] are shown on the table 3.2.

Precision	f
Single	2.0317×10^{29}
Double	5.960604×10^{20}
Double-extended	$-9.38724 \times 10^{-323}$

Table 3.2: Rump's function results. Reproduced from Muller et al.[132]

The intensive use of 64-bit and 128-bit slows down the solver,[44] and may not lead to a more accurate solution. This became more appealing, when coding in CUDA, as the majority of the GPU hardware only works efficiently[44] in 32-bit.

If a computer complies with the IEEE standards, the machine epsilon[188] is $\varepsilon_M \approx 1.11 \times 10^{-16}$. There are many routines and classes to compute the ε_M, amount them MACHAR.[2] A simple way to calculate ε_M is shown in the algorithm 3.3.

Algorithm 3.3 Calculate Machine Epsilon. Algorithm reproduced from Karniadakis et al.[86]

```
double $dLv\_eps$                                  ▷ Machine epsilon
double $dLv\_tst$                                  ▷ Test
$dLv\_eps = 1.0$
$dLv\_tst = 1.0 + dLv\_eps$
while $1.0 \neq dLv\_tst$ do
    $dLv\_eps = dLv\_eps/2.0$
    $dLv\_tst = dLv\_tst + dLv\_eps$
end while
```

The effect of ε_M in the discrepancies between parallel and sequential solutions, is better illustrated by a simple example of 4 atoms. The same calculus is performed for a sequential code shown in Figure 3.20a, and for the parallel code shown in Figure3.20b and Figure3.20c. For simplicity only forces in the x direction are considered.

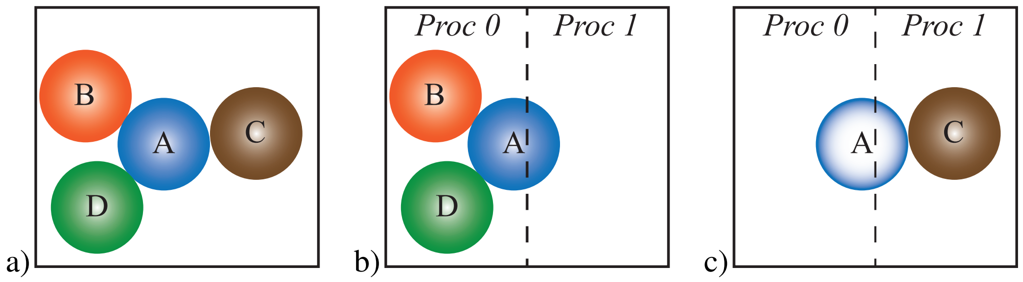
Adding the forces of process 0 and process 1

$$\begin{gathered} F_{A_{par}} = F_{A_{proc0}} + F_{A_{proc1}} \\ F_{A_{par=(1.0+\varepsilon)-1.0}} \end{gathered} \tag{3.34}$$

Taking into account the rounding error, both forces are not equal as shown in the next equation

$$\begin{gathered} F_{A_{seq}} \neq F_{A_{par}} \\ (1.0-1.0)+\varepsilon \neq (1.0+\varepsilon)-1.0 \\ \varepsilon \neq 0.0 \end{gathered} \tag{3.35}$$

For simulations involving Discrete Elements, these small changes can generate a different behaviour of the system.[138] These discrepancies are not in any way only concerning to parallel calculus. As Munjiza[82] pointed out small differences in the initial conditions of a sequential FDEM simulation can lead to different results.



$$F_{A_{seq}} = F_B + F_C + F_D \qquad F_{A_{proc0}} = F_B + F_D \qquad F_{A_{proc1}} = F_C$$

$$F_{A_{seq}} = (1.0-1.0)+\varepsilon \qquad F_{A_{proc0}} = 1.0+\varepsilon \qquad F_{A_{proc1}} = -1.0$$

Figure 3.20: Calculus of the force in the atom A. All the forces are added in alphabetic order of the atom's *id*. a)Sequential calculus. b)Parallel calculus in the process 0. Is important to notice the absence of the atom C, as no part of it is in the process 0 c)Parallel calculus in the process 1. Only The proxy of atom A and atom C are in the process 1.

Chapter 4

Novel CD algorithm for bodies of similar size

4.1 Introduction

Large-scale DEM simulations involve contact between millions of different entities, usually called atoms, molecules, particles, bodies, discrete elements, etc. In dynamics problems where most of the particles are in movement, the CPU time required to search for all the contacting couples (which ones are in contact with each other) can be over 60% of the CPU time in many cases.[133]

There are different algorithms proposed to perform searches for these couples, namely Direct Search(DS), Quick Sort,[165] Balanced Binary Tree(BBT),[93] Munjiza-No Binary Search (NBS),[136] Munjiza-Rougier(MR),[148] Williams C-grid,[163] Alternating Digital Tree(ADT),[18] Augmented Spatial Digital Tree (ASDT),[47] 3D-DDA,[16] Discrete Function Representation,[200] SMB[102] as well as many others.

The simplest algorithm called Direct Search (DS) shown in Algorithm 3.1, performs a direct search between each of the elements in the domain. The contact detection time is given by[136, 133]

$$T \propto N^2 \tag{4.1}$$

The Balanced Binary Tree is a logarithmic search algorithm, i.e. the contact detection (CD) time needed to find a matching item is given by[93]

$$T \propto \log(N) \tag{4.2}$$

and the contact detection time needed to find N items is given by[93]

$$T \propto N \log(N) \tag{4.3}$$

In the case of the MR and NBS algorithms, the CD time increase proportionally with the amount of DEs.[133, 148] The CD time for these algorithms is given by

$$T \propto N \tag{4.4}$$

The MR,[148] NBS,[136] and Balanced Binary Tree (BBT)[93] algorithms, have been studied previously as independent algorithms. The decision of which algorithm to apply to a particular case is not obvious as there is not enough evidence or tests of these algorithms in the literature.

This work aims to identify the algorithm that best suits a particular problem (i.e. 1D, 2D, 3D Discrete Elements simulations) for bodies of similar size. Most of the work was written in the form of C++ object oriented programming.[104] There is a special variation of the original MR algorithm also using formatting more suitable for languages such as C[88] and Fortran[166] i.e. using arrays instead of objects and pointers. This algorithm is called MR-S (MunjizaRougier-Schiava).

In this chapter the modification to the BBT for use in contact detection is presented. This novel algorithm, which has been named BBTS (Balanced Binary Tree Schiava), has as its main algorithms: BBTS-load and BBTS-search. Modifications have been included to make it suitable for contact search in 1D, 2D and 3D, combining the BBT with space cell decomposition.[148, 136, 133] Also, a novel "binary bush" visualization has been exploited as originally proposed by Munjiza.[139] In this visualization the Binary Tree is shown without its root, placing the first node of the tree in the centre, with its sons placed concentrically as shown in Figure 4.1.

In chapter 5 short descriptions of the Munjiza-NBS, MR and MR-S are presented, while the comparison of performance between MR, MR-S, NBS and BBTS for structured distributions and random distributions in 1D, 2D and 3D is presented in Chapter 7.

4.2 Balanced Binary Tree

The processes involving the storage and retrieval of data in a balanced binary tree are described in depth by Gary D.Knott.[93] One of the differences between Knott[93] and this method is the way in which nodes are considered. In this formulation the nodes are objects; this means that there is no necessity to include an array which points from one node to another. This information is stored in each node instead of an array.

In a binary tree each node a is a right son of b only if $a < b$, and a is a left son of b only if $a > b$. For a tree that complies with these two rules and in “which the items are randomly-received”,[93] the search time required to find a matching item is generally a logarithmic function of N. However, this only occurs when the tree is balanced, which occurs when “no excessively short or long path exists”[93] . Such paths tend to increase the average number of comparisons required.

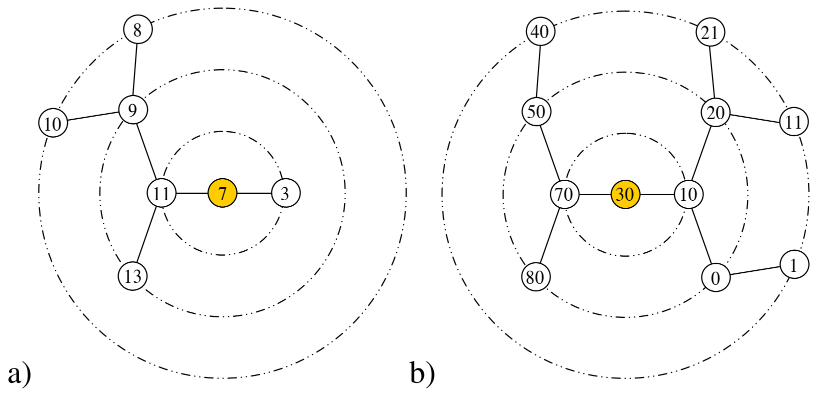


Figure 4.1: Binary Bush[139]. a)Unbalanced tree $B_7 = -2$. b)Balanced tree $B_{30} = 0$.

A binary tree is balanced only if all its nodes are balanced. The balance[93] B of a given node i is given by equation 4.5

$$B_i = h(R_{si}) - h(L_{si}) \tag{4.5}$$

where $h(R_{si})$ and $h(L_{si})$are the numbers of nodes in the longest paths of the sub-trees located to the right and left of node i respectively. A given node i is said to be balanced if

$$-1 \leq B_i \leq 1 \tag{4.6}$$

The balance of the node 7 in the tree shown in Figure 4.1a is given by

$$\begin{aligned} h(R_{s7}) &= 1 \\ h(L_{s7}) &= 3 \\ B_7 &= -2 \end{aligned} \tag{4.7}$$

which means the tree is unbalance. An example of a balanced tree is shown in Figure 4.1b where the balance of node 30 is given by

$$\begin{aligned} h\left(R_{s30}\right) &= 3 \\ h\left(L_{s30}\right) &= 3 \\ B_{30} &= 0 \end{aligned} \tag{4.8}$$

Space decomposition. There are many ways to tackle the Contact Detection problem using BBT. The most common technique is to bisect the domain into regions and later to bisect these regions further.[47] The division of the domain is the key for the performance of the algorithm. A well-known technique is the quadtree.[161, 181] A simple example of which is shown in Figure 4.2.

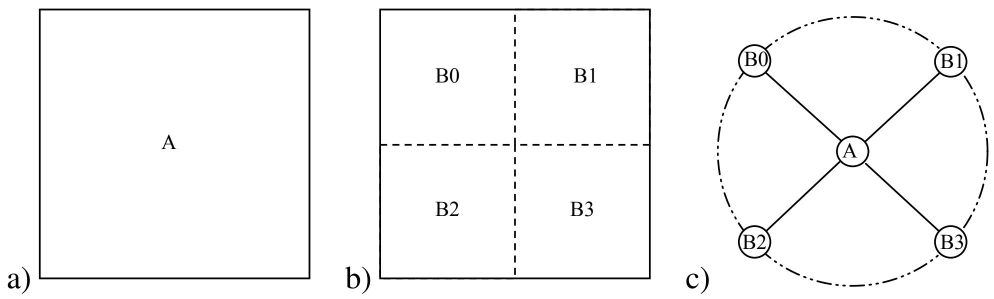


Figure 4.2: Space decomposition. a)Original region A b)Subdivision of the region A c) Tree representation

To avoid this subdivision in the BBTS algorithm all DEs are simplified as a spherical object.[136] The diameter of this object d is equal to the biggest sphere that can contain the biggest DE in the system[136] as shown in Figure 4.3a. These objects are mapped onto a rectangular domain made of cubical subdomains or size d,[133] as shown in Figure 4.3b.

The numbers of subdomains in the x, y and z direction are given by

$$\begin{aligned} n_x &= Int\left(\tfrac{x_{max}-x_{min}}{d}\right)+1 \\ n_y &= Int\left(\tfrac{y_{max}-y_{min}}{d}\right)+1 \\ n_z &= Int\left(\tfrac{z_{max}-z_{min}}{d}\right)+1 \end{aligned} \tag{4.9}$$

where x_{max}, y_{max} and z_{max} are the maximum x, y and z coordinates, x_{min}, y_{min} and z_{min} are the minimum x, y and z coordinates and d is the size of the cells.

The advantage of this space decomposition is its simplicity as there is no need to subdivide the space at running time. Only a simple mapping of objects into subdomains (cells) is performed at running time. Also the algorithm is independent of the packing density, in terms of velocity (the velocity for loose packages only depends on the number of elements) and RAM memory (each element is represented by an unique node on the tree).

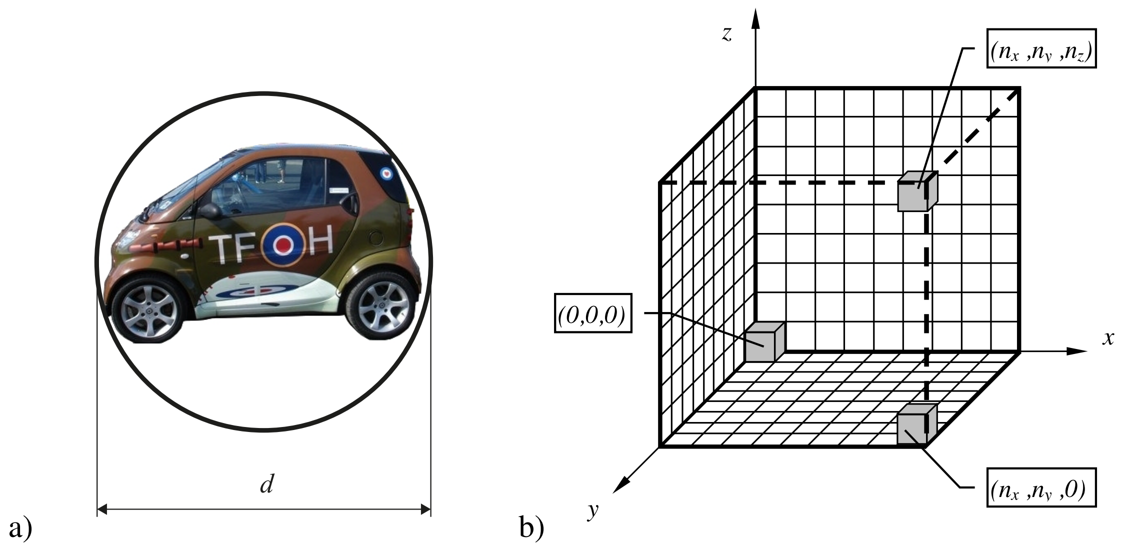


Figure 4.3: Space Decomposition. b)Spherical bounding box. b)Cubical sudbomains (cells). Figures adapted from Rougier.[175]

Mapping of discrete elements into cells. Each DE has only one set of coordinates corresponding to the cell where it is mapped,[148] thus simplifying the contact detection algorithm. The formulae for the mapping are given by

$$\begin{aligned} i_x &= Int\left(\frac{x-x_{min}}{d}\right) \\ i_y &= Int\left(\frac{y-y_{min}}{d}\right) \\ i_z &= Int\left(\frac{z-z_{min}}{d}\right) \end{aligned} \tag{4.10}$$

where x, y and z are coordinates of the centre of the spherical bounding box, x_{min}, y_{min} and z_{min} are the minimum x, y and z coordinates of the system, d is the cell size and i_x, i_y and i_z are the x, y and z integerised coordinates respectively.

4.2.1 Basic structure of the BBTS

In the implementation of the BBTS used in this work, each subdomain (cell) is represented by a node, and the root of the entire tree is formed by an empty node and has only one son to the left as shown in Figure 4.4 .

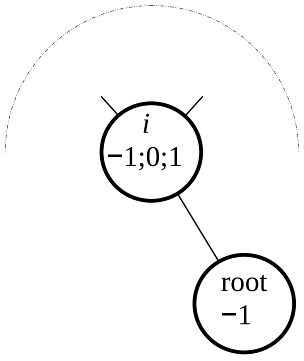


Figure 4.4: Root of the entire tree and its son. Figure adapted from Knott[93]

The information stored inside each node is listed in Table 4.1

Variable Name	Variable Description
L_s	Pointer to the left son of the node
R_s	Pointer to the right son of the node
V	Key Value of the node
B	Balance of the node
O	Void pointer to any object (or it could be a integer *id*)

Table 4.1: Information stored inside each node

The key value of a node is given by

$$V = [i_x, i_y, i_z] \tag{4.11}$$

where i_x, i_y and i_z are the integerised coordinates of the bounding box. The relationship between a given node i and its left son is given by

$$V_{Lsi} > V_i \tag{4.12}$$

where V_{Lsi} is the key value of the left son of node i and V_i is the key value of the node i. In a similar way, the relationship between a given node i and its right son is given by

$$V_{Rsi} < V_i \tag{4.13}$$

where V_{Rsi} is the key value of the right son of node i and V_i is the key value of the node i. To establish the relationship shown in equations 4.12 and 4.13 a spatial ordering criterion is needed.[148] In this work the spatial ordering criterion adopted is the same criterion used in the MR algorithm, and it "states that a bounding box i is greater than a bounding box j if":[148]

$$\begin{gathered}\{[(i_{zi} > i_{zj})]\ or\ [(i_{zi} = i_{zj})\ and\ (i_{yi} > i_{yj})]\} \\ or\ \{[(i_{zi} = i_{zj})\ and\ (i_{yi} = i_{yj})\ and\ (i_{xi} > i_{xj})]\}\end{gathered} \tag{4.14}$$

A simple example of the use of this criterium is shown in Figure 4.5

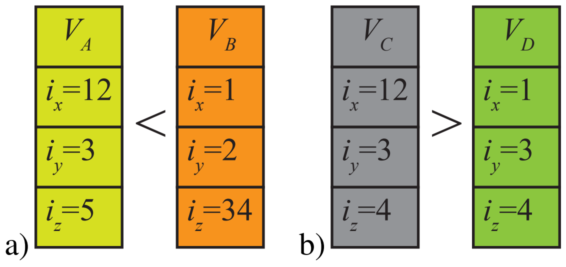


Figure 4.5: Spatial ordering criterium. a)V_A<V_B. b)V_C>V_D.

4.3 BBTS-load

In the case that a new object (triangle, tetrahedra, etc) with a key value V_{new} needs to be added to the tree, a new node is created; its key value set to V_{new} and its pointer O set to the object to add. The remaining question is: where is this new node to be placed inside the BBTS structure? To answer this question the BBTS is parsed starting from the left son of the root.

Algorithm 4.1 Simple add procedure. Add object with key V_{new}. "→" has the C++ meaning

```
1: $Nod_{cur} = BT \rightarrow GetFrs()$    ▷ Get first node on tree. i.e. the left son of the root
2: while ($Nod_{cur}! =$ Null) do
3:     if $BT \rightarrow IsGreThan(Nod_{cur}, V_{new})$ then    ▷ IsGreaterThan i.e. $(V_{cur} > V_{new})$
4:         $Nod_{cur} = BT \rightarrow GetRigSon(Nod_{cur})$    ▷ go to the rigth son $(R_s)$
5:     else if $BT \rightarrow IsLesThan(Nod_{cur}, V_{new})$ then    ▷ IsLessThan i.e. $(V_{cur} < V_{new})$
6:         $Nod_{cur} = BT \rightarrow GetLefSon(Nod_{cur})$    ▷ go to the left son $(L_s)$
7:     end if
8: end while
9: Add element to the empty position
```

The key value V_{new} of the new node is compared to V_{cur} which is the key value of the current node. If the V_{new} is less than V_{cur} then the next node to be checked is set to be the right son of the current node. On the other hand if the V_{new} is greater than V_{cur} then the next node to be checked is set to be the left son of the current node. The process described above is repeated until an empty position is found.[93] When this is achieved the new node is placed on that empty position. This is best illustrated by the Algorithm 4.1.

For example, if the node d is to be added to the BBTS shown in Figure 4.6a, and the key value of the node d is such that

$$V_d < V_c \tag{4.15}$$

then, the node d would be positioned to the right of node c as shown in Figure 4.6b.

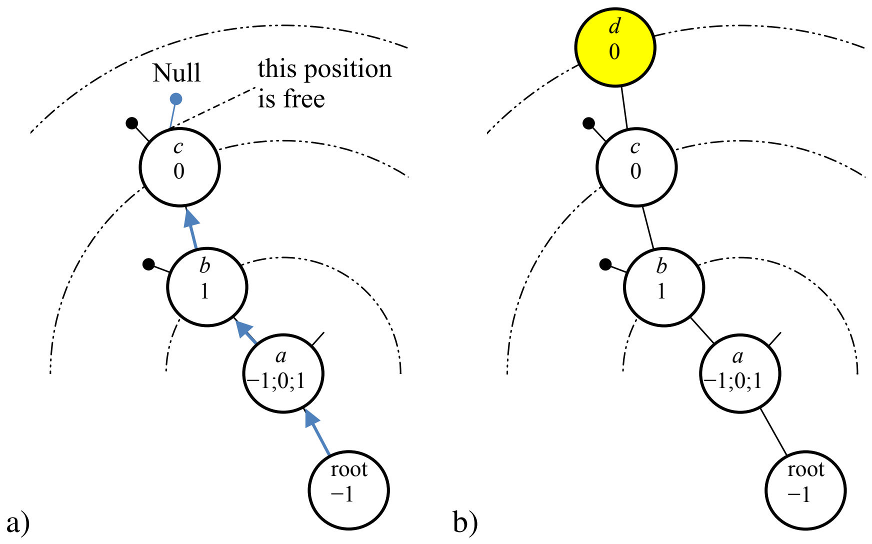


Figure 4.6: a) Search for the leaf node position for $x = d < c$ b) Add new node.

In the original algorithm[93] when a new element with the same key value as one of the existing nodes of the tree is added the same procedure described above is performed. In this way for example, if an element is added to the BBTS shown in Figure 4.7a with key value $V = 30$, then the tree shown in Figure 4.7b would be obtained. In this case, the tree would be balance on terms of height but the search time will not be bounded by a logarithmic function of N.

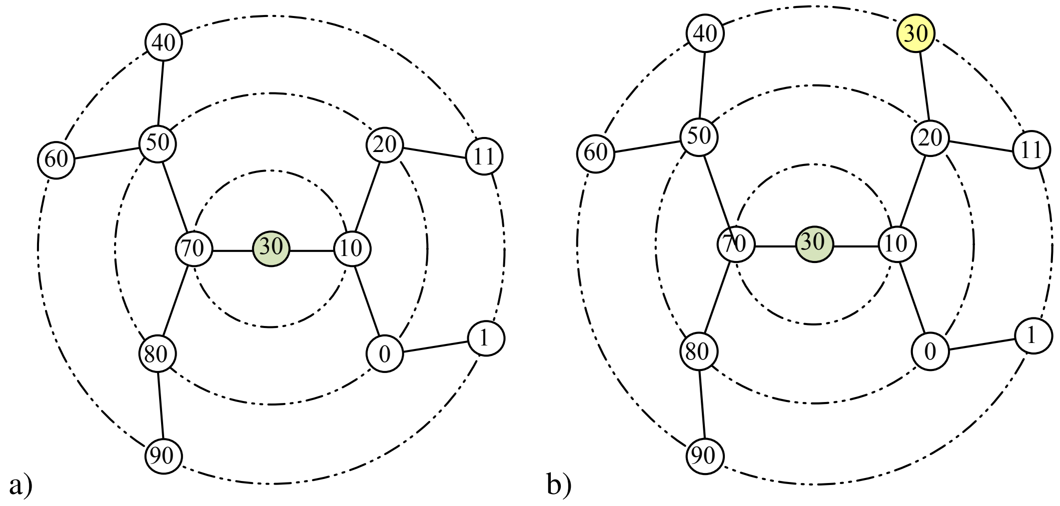


Figure 4.7: Binary Bush a)Before adding “30” b) After adding “30”

This situation would be catastrophic in terms of CPU time spent in searching for an element with a particular key value, because one search procedure would not be enough to find all the elements with a particular key value. Instead an uncertain number of searches would be necessary to perform the contact detection process.

To overcome this problem a new pointer is introduced to each node, as shown in Figure 4.8a. Now in the case that three nodes share the same key value they would be arranged one next to one another as shown in Figure 4.8b while only one of them “is” in the tree structure. This would make the tree structure to look as shown in Figure 4.8a.

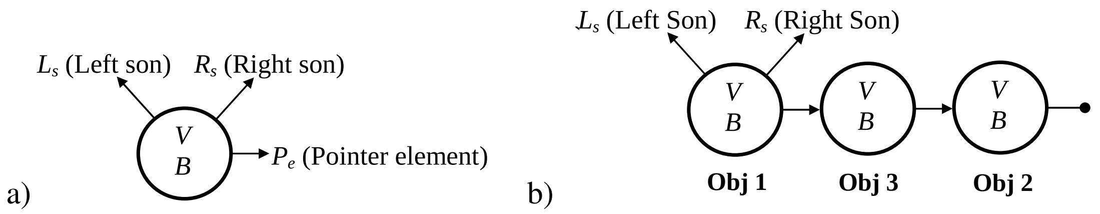


Figure 4.8: Node a)New node with pointer to nodes with same key value V. b)More than one node with the same value V

The new algorithm that takes into account elements that share the same key value; is given by the Algorithm 4.2.

The amount of RAM used by the BBTS algorithm does not depend on the packing density, as each object in the domain is represented by an unique node. The amount of RAM is given by the equation 4.16.

When a new node is added[93] to the tree the B values of each of the nodes must be updated, i.e. the balance of each of the nodes must be recalculated. This is done with the help of two vectors. The first vector contains the pointers to all the elements of the branch of the tree where the new object has been added starting from the root tree. For the case shown in Figure 4.6b this vector is shown in Table 4.2a.

$$\begin{aligned}
V &= 3\,\text{short integer numbers} &&= 6\,\text{bytes}\\
B &= 1\,\text{short integer number} &&= 2\,\text{bytes}\\
L_s &= 1\,\text{pointers} &&= 8\,\text{bytes}\\
R_s &= 1\,\text{pointers} &&= 8\,\text{bytes}\\
P &= 1\,\text{pointers} &&= 8\,\text{bytes}\\
O &= 1\,\text{pointers} &&= 8\,\text{bytes}\\
M &= V+B+L_s+R_s+P+O &&= 40\,\text{bytes}
\end{aligned} \tag{4.16}$$

Algorithm 4.2 Add procedure, elements with the same key value. "$\rightarrow$" has the C++ meaning

```
1: $Nod_{cur} = BT \rightarrow GetFrs()$        ▷ Get first node on tree. i.e. the left son of the root
2: while $(Nod_{cur}! = \text{Null})$ do
3:     if $BT \rightarrow IsGreThan(Nod_{cur}, V_{new})$ then    ▷ IsGreaterThan i.e. $(V_{cur} > V_{new})$
4:         $Nod_{cur} = BT \rightarrow GetRigSon(Nod_{cur})$        ▷ go to the rigth son $(R_s)$
5:     else if $BT \rightarrow IsLesThan(Nod_{cur}, V_{new})$ then    ▷ IsLessThan i.e. $(V_{cur} < V_{new})$
6:         $Nod_{cur} = BT \rightarrow GetLefSon(Nod_{cur})$        ▷ go to the left son $(L_s)$
7:     else
8:         add new node to $P_e$ of $Nod_{cur}$
9:     end if
10: end while
11: Add element to the empty position
```

a)

Component	Pointer to
0	Root
1	a
2	b
3	c

b)

Component	Exit direction
0	-1
1	-1
2	1
3	1

Table 4.2: Path of the vector. a)Pointers to each element in the branch. b) Exiting direction from each of the elements of the branch

The second vector contains a flag that denotes the direction from which each of the elements of the branch of the tree under consideration are exited. For the case shown in Figure 4.6b this vector is shown in Table 4.2b. If an element is exited from the left then the flag is equal to -1 and if the element is exited from the right then the flag is equal to 1.

The whole branch is now parsed starting from the element located immediately before the new element that has just been added to the BBTS. This can be seen in the case shown in Figure 4.6 where starting point is the element c.

The values of the node balance variables are updated according to the exiting direction from each node. For example in the case of node c shown in Figure 4.6b the node balance variable is given by Equation 4.17

$$B_{c\,new} = B_{c\,old} + \Delta B_c \tag{4.17}$$

where $B_{c\,old}$ and $B_{c\,new}$ are the old and new values of the balance variable for node c, and $\triangle B_c$ is the change in the balance of node c due to the addition of node d to the tree. In this case $\triangle B_c$ is given by Equation 4.18

$$\Delta B_c = +1 \tag{4.18}$$

because node c is exited from the right, i.e. theV_c is greater than V_d

Depending on the updated value of B_i, three cases can arise

- Case 1: $B_i = 0$ no further updating of balancing variables is needed because the height of the branches that have node i as a root has not changed.
- Case 2: $|B_i| = 1$ the root of node i needs to have its balance variable updated, i.e. the updating process must continue towards the root of this sub-tree.
- Case 3: $|B_i| = 2$ the sub-tree starting in the node i is unbalanced, and therefore needs to be balanced. This is shown in Figure 4.9a for the node b

4.3.1 Balancing procedure

As was mentioned earlier it is essential to have a BBTS balanced to have bounded search times. For each unbalanced value of the node given by

$$\begin{aligned} B_i &= 2 \\ B_i &= -2 \end{aligned} \tag{4.19}$$

will be a right or left son j of the node i with balance B_j as

$$-1 \leq B_j \leq 1 \tag{4.20}$$

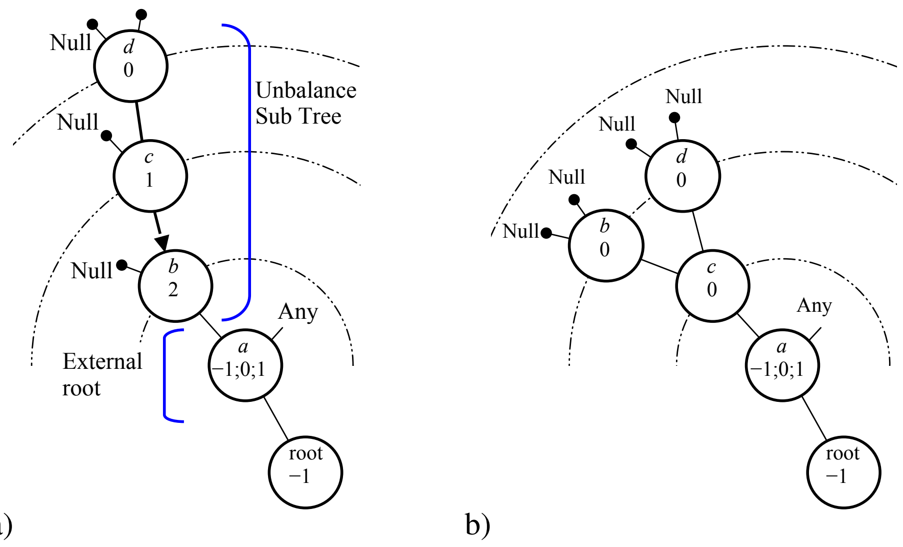


Figure 4.9: a) Retraced of the search path b) Final Balance of the tree. Figures adapted from Knott[93]

Thus there are four possible cases,[93] depending on the position of the nodes, for the balancing procedure as shown in Table 4.3. The basic idea, in each of them is to "rotate" the sub-tree to be balanced in the "direction" of its "shorter side".[93]

Case	B_i	B_j
1	2	≥ 0
2	-2	$0 \leq$
3	2	-1
4	-2	1

Table 4.3: List of possible cases when balancing a sub-tree

Case 1. In this case, the balance of node j can have the following values

$$\begin{aligned} B_{jold} &= 0 \\ B_{jold} &= 1 \end{aligned} \tag{4.21}$$

where B_{jold} is the balance of the balance for node j before the balancing of the sub-tree. The unbalanced sub-tree with root in node h is pivoted around node h to balance it. This is shown for the most general case in Figure 4.10, the possible balance values are listed in Table 4.4.

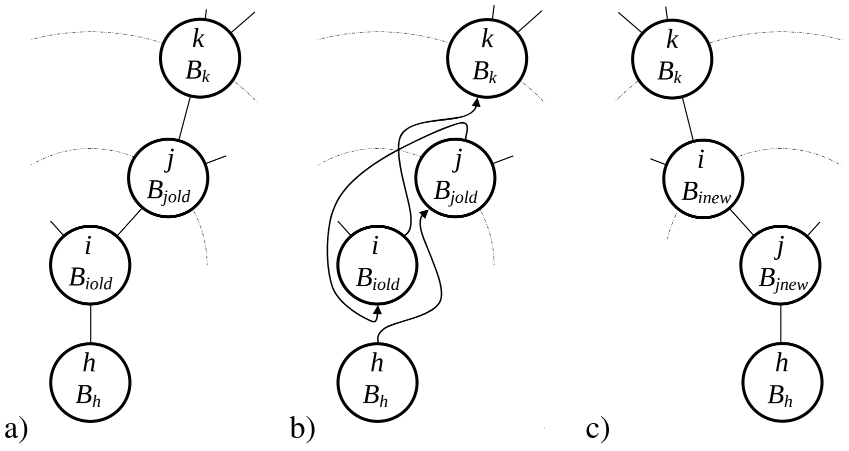


Figure 4.10: Balancing of node i from the tree. a) Initial configuration. b) Necessary rearrangements, c) Final configuration. Figures adapted from Knott[93] .

Balance Variable	Possible Values
B_h	−1,0,1
B_{iold}	2
B_{jold}	0,1
B_k	−1,0,1
B_{inew}	0,1
B_{jnew}	−1,0

Table 4.4: Case 1: possible balance values

The new balance for nodes i and j are given by

$$B_{inew} = -B_{jold} + 1 \tag{4.22}$$

$$B_{jnew} = B_{jold} - 1 \tag{4.23}$$

where B_{inew} and B_{jnew} are the balance for nodes i and j after the balancing is performed.

A particular case where $B_{iold} = 2$, $B_{jold} = 1$, $B_a = 0$ is shown in Figure 4.11a. The values of the balance variables after balancing are given by

$$B_{inew} = -B_{jold} + 1 = 0 \tag{4.24}$$

$$B_{jnew} = B_{jold} - 1 \tag{4.25}$$

this is illustrated in Figure 4.11c.

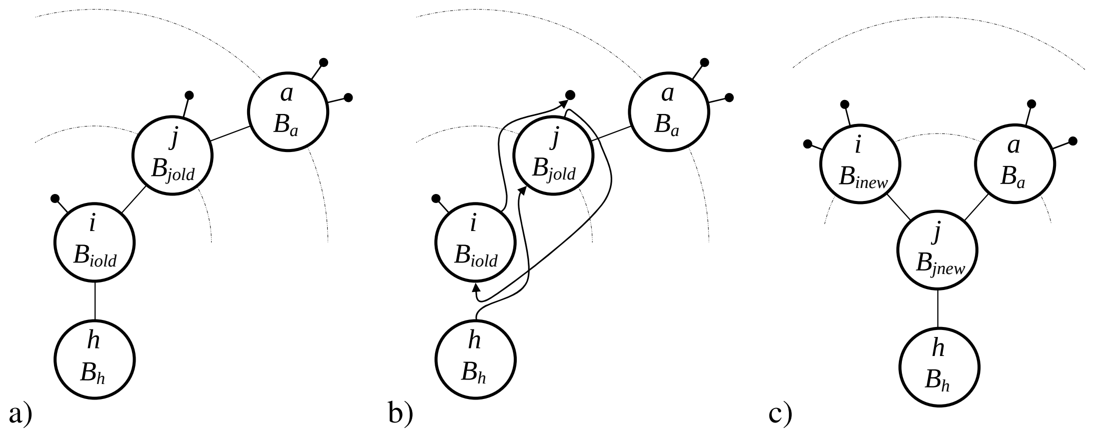


Figure 4.11: Example of balancing of node i with node B_{jold} =1. Figures adapted from Knott[93]

Case 2. In this case the balance of node j can have the following values

$$\begin{aligned} B_{jold} &= 0 \\ B_{jold} &= -1 \end{aligned} \tag{4.26}$$

where B_{jold} is the balance of node j before the balancing of the sub-tree. The most general case is illustrated in Figure 4.12.

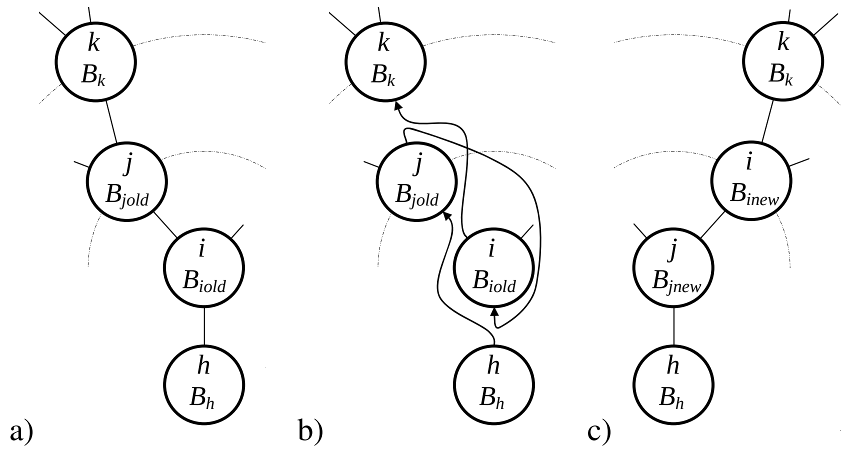


Figure 4.12: Balancing of node i from the tree. a) Initial configuration. b) Necessary rearrangements. c) Final configuration. Figures adapted from Knott[93] .

The new values of the balance variables for nodes i and j are given by

$$B_{inew} = -B_{jold} - 1 \tag{4.27}$$

$$B_{jnew} = B_{jold} + 1 \tag{4.28}$$

The possible balance values are listed in Table 4.5.

Balance Variable	Possible Values
B_h	−1,0,1
B_{iold}	−2
B_{jold}	−1,0
B_k	−1,0,1
B_{inew}	−1,0
B_{jnew}	0,1

Table 4.5: Case 2: possible balance values

A particular case where $B_{jold} = -2$, $B_{jold} = -1$, $B_a = 0$ is shown in Figure 4.13 The balance values after the balancing is performed are given by

$$B_{inew} = -B_{jold} - 1 = 0 \tag{4.29}$$

$$B_{jnew} = B_{jold} + 1 = 0 \tag{4.30}$$

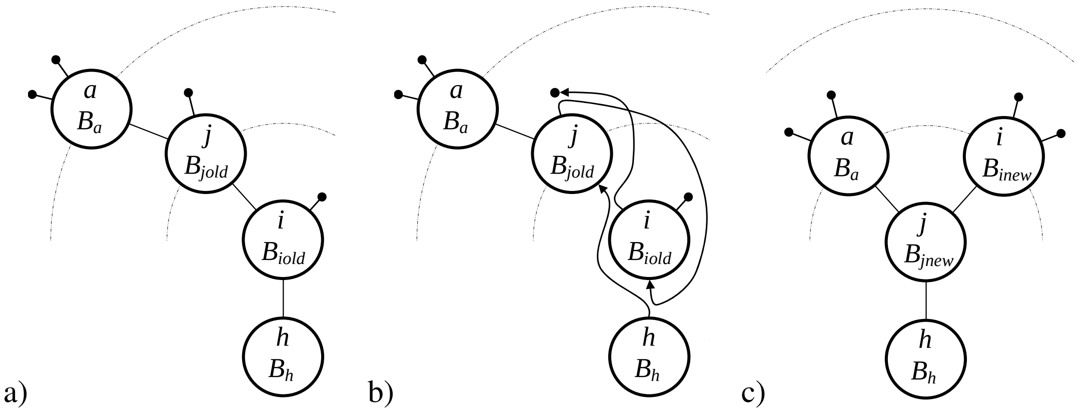


Figure 4.13: Example of balancing of node i with node $B_{jold} = -1$. Figures adapted from Knott[93] .

Case 3. In this case the new values of the balance variables for nodes i, j and k are given by

$$B_{inew} = \begin{cases} 0 & for \quad B_{kold} = 0 \\ -\left(\frac{1+B_{kold}}{2}\right) & for \quad B_{kold} = \pm 1 \end{cases} \tag{4.31}$$

$$B_{jnew} = \begin{cases} 0 & for \quad B_{kold} = 0 \\ \left(\frac{1-B_{kold}}{2}\right) & for \quad B_{kold} = \pm 1 \end{cases} \tag{4.32}$$

$$B_{knew} = 0 \tag{4.33}$$

The balancing of node i is described in Figure 4.14.

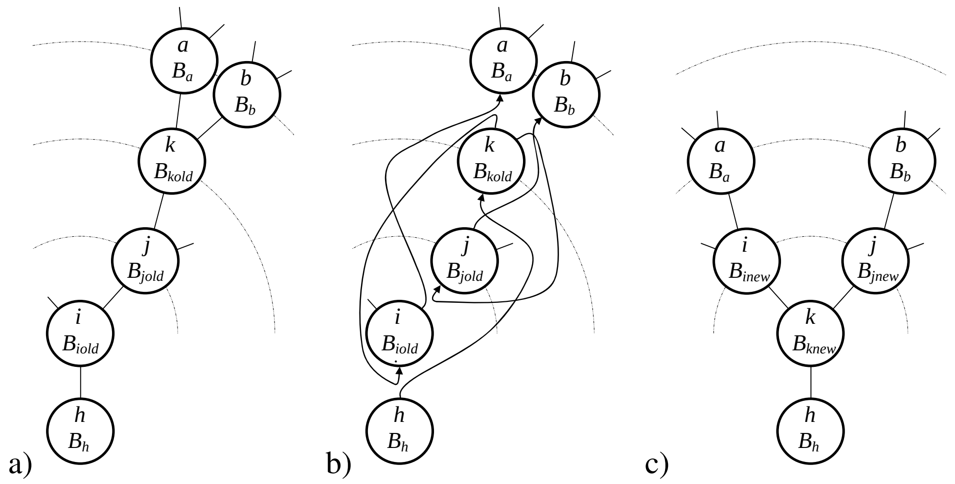


Figure 4.14: Balancing of node i from the tree. a) Initial configuration. b) Necessary rearrangements. c) Final configuration. Figures adapted from Knott[93] .

The the possible balance values are given in Table 4.6.

Balance Variable	Possible Values
B_a	−1,0,1
B_b	−1,0,1
B_{iold}	2
B_{jold}	−1
B_{kold}	−1,0
B_{inew}	0,1
B_{jnew}	0

Table 4.6: Case 3: possible balance values

Case 4. The new balance values for nodes i, j and k are given by

$$B_{inew} = \begin{cases} 0 & for \quad B_{kold} = 0 \\ \left(\frac{1-B_{kold}}{2}\right) & for \quad B_{kold} = \pm 1 \end{cases} \tag{4.34}$$

$$B_{jnew} = \begin{cases} 0 & for \quad B_{kold} = 0 \\ -\left(\frac{1+B_{kold}}{2}\right) & for \quad B_{kold} = \pm 1 \end{cases} \tag{4.35}$$

$$B_{knew} = 0 \tag{4.36}$$

The balancing process for the most general case is shown in Figure 4.15.

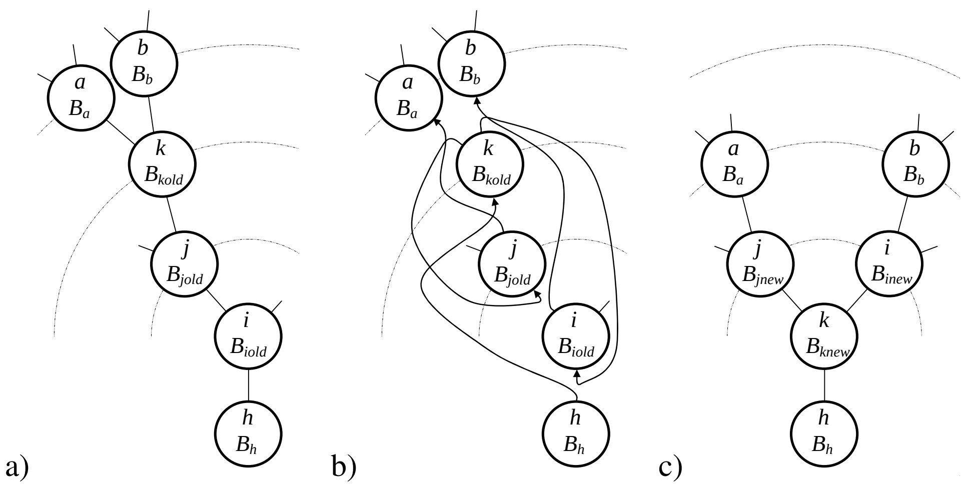


Figure 4.15: Balancing of node *i* from the tree. a) Initial configuration. b) Necessary rearrangements. c) Final configuration. Figures adapted from Knott[93]

The possible balance values are given in Table 4.7.

Balance Variable	Possible Values
B_a	−1,0,1
B_b	−1,0,1
B_{iold}	-2
B_{jold}	1
B_{kold}	−1,0,1
B_{inew}	0,1
B_{jnew}	-1,0
B_{knew}	0

Table 4.7: Case 4: possible balance values

4.4 BBTS-search

When dealing with systems in which all the particles change their coordinates at every time step, it is faster to build the whole tree from the root at each time step than to rearrange a pre-existing binary tree. This is the reason why, in this work, the whole binary tree is built from the root when CD is performed.

Search procedure. The search procedure is started by setting the current node to the left son of the root of the tree. The key value to be searched V_s is compared to the key value of the current node V_c. If the V_c is greater than V_s then the right son of the current node is selected. In a similar way, if V_c is smaller than the V_s then the left son of the current node is selected. This process is repeated until either a node with a key value equals to the searched key value is found or the end of the tree is reached, i.e. there is no element stored in the tree with a key value equals to the searched key value. This process is best illustrated by the Algorithm 4.3.

Algorithm 4.3 Search procedure

```
 1: $Nod_{cur} = BT \rightarrow GetFrs()$          ▷ Get first node on tree. i.e. the left son of the root
 2: while $(Nod_{cur}! = \text{Null})$ do
 3:     if $BT \rightarrow IsGreThan(Nod_{cur}, V_{search})$ then   ▷ IsGreaterThan i.e. $(V_{cur} > V_{sea})$
 4:         $Nod_{cur} = BT \rightarrow GetRigSon(Nod_{cur})$          ▷ go to the rigth son $(R_s)$
 5:     else if $BT \rightarrow IsLesThan(Nod_{cur}, V_{search})$ then ▷ IsLessThan i.e. $(V_{cur} < V_{sea})$
 6:         $Nod_{cur} = BT \rightarrow GetLefSon(Nod_{cur})$          ▷ go to the left son $(L_s)$
 7:     else
 8:         return $Nod_{cur}$
 9:     end if
10: end while
11: return Null
```

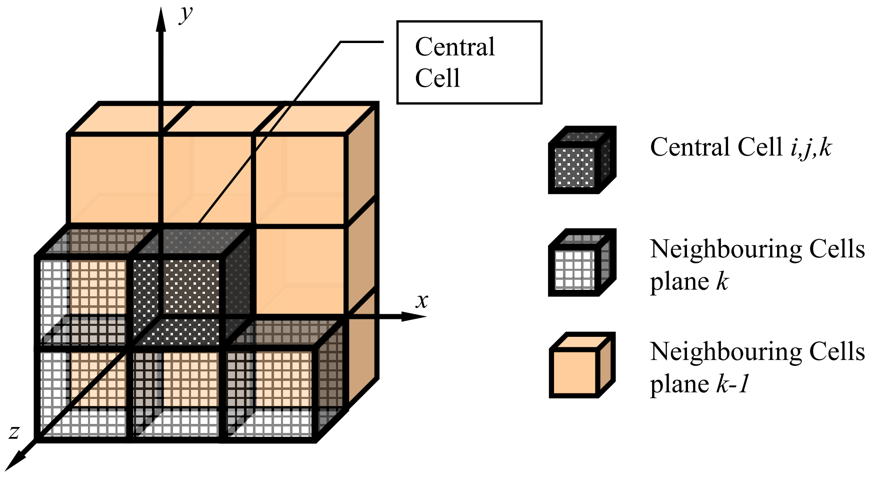


Figure 4.16: Contact mask in 3D. Figure modified from Munjiza and Rougier[148]

Finding contacting couples. As it was mentioned before, a particular discrete element has only one set of coordinates corresponding to the subdomain where it is mapped.[148] Thus, a given DE can be in contact only with those DEs mapped either to the same cell or to any of the neighbouring cells. To avoid repetition of contacting couples a contact checking mask[148, 136, 133] is adopted, as shown in Figure 4.16.

To find all the contacting couples for a given DE mapped to the cell i, j, k all the DEs with integerised coordinates

$$\begin{array}{ccc} i-1,j+1,k-1 & i,j+1,k-1 & i+1,j+1,k-1 \\ i-1,j,k-1 & i,j,k-1 & i+1,j,k-1 \\ i-1,j-1,k-1 & i,j-1,k-1 & i+1,j-1,k-1 \\ i-1,j,k & i,j,k & \\ i-1,j-1,k & i,j-1,k & i+1,j-1,k \end{array} \tag{4.37}$$

must be found. This is done by selecting one by one the cells of the contact checking mask, and parsing the tree from its root to find the elements mapped to that particular cell. In total 14 searches have to be perform.

Chapter 5

MR-S ALGORITHM

5.1 Introduction

In this chapter a description of the MR-S (MunjizaRougier-Schiava) algorithm is presented. The MR-S algorithm is a flavour of the MR (MunjizaRougier) algorithm implemented with arrays instead of objects. This modification makes MR-S more suitable for languages such as C and Fortran.

To explain the MR algorithm it is necessary to describe its roots in the so-called Munjiza-NBS (Non Binary Search) algorithm. A short description of NBS is presented in section 2. In section 3 the MR algorithm is described. Finally, in the last section the MR-S algorithm is explained.

5.2 NBS algorithm

This algorithm was one of the first to present a proportional CD time with respect to the number of DEs[136] N.

$$T \propto N \tag{5.1}$$

It is based on the space decomposition into subdomains (cells), as shown in Figure 5.1a. The size of the square subdomains is defined by the diameter of the biggest spherical bounding box in the domain. The entire domain is represented using arrays of integers. A detailed explanation of this algorithm is given in Munjiza's book.[133] In this section NBS is explained using a simple 2D example.[133]

The first stage of this algorithm is a loop over the entire set of elements, calculate their integerised coordinate in i_y using Equation 4.10, and place the elements into the corresponding single connected list of y_i row, as shown schematically in Figure 5.1b.

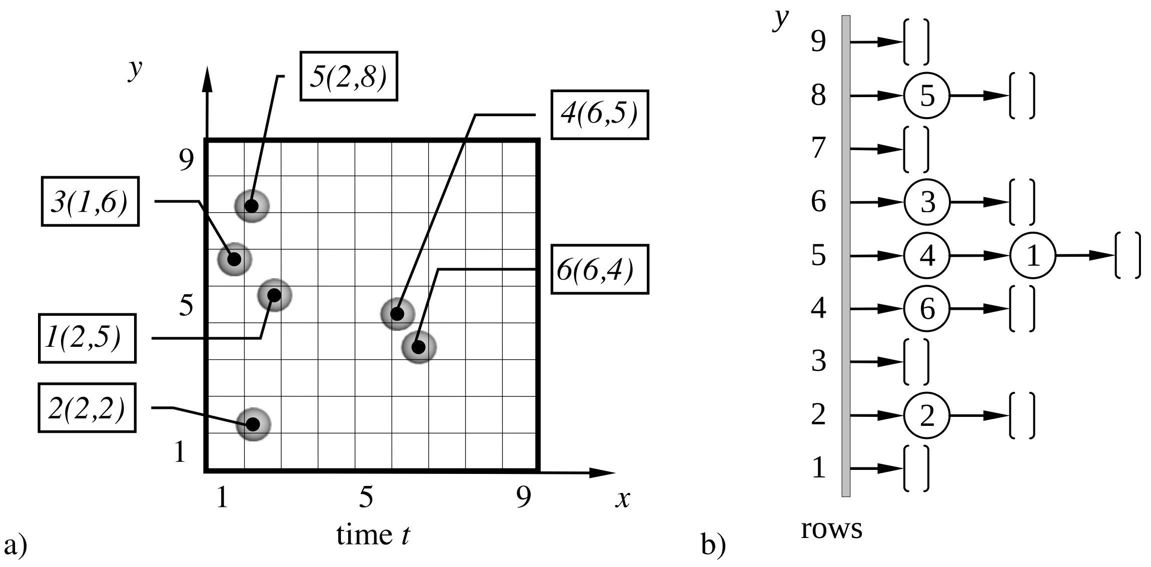
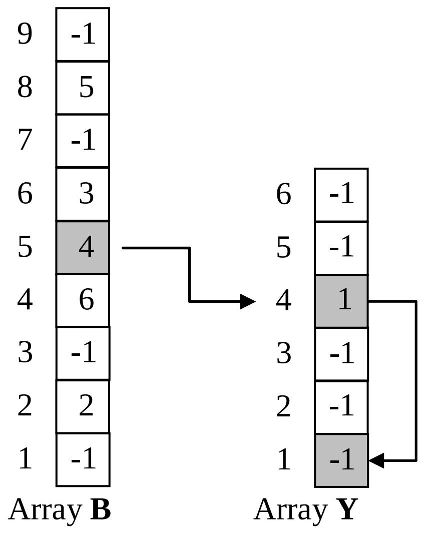


Figure 5.1: a) Discrete elements time step t. Each DE is tagged as $id`(i_x, i_y)$ b) Single connected lists rows. Figures modified from Munjiza.[133]



Array **B** row	Array **B** value
9	-1
8	5
7	-1
6	3
5	4
4	6
3	-1
2	2
1	-1

Array **Y** row	Array **Y** value
6	-1
5	-1
4	1
3	-1
2	-1
1	-1

Figure 5.2: Arrays **B** and **Y.** Figure modified from Munjiza.[133]

This list is stored in two 1D arrays, the first "array **B** size n_y, where n_y is the number of cells in the y direction".[136] An empty row is tagged with -1 otherwise the digit is equal to the id of the last element added on the row. The second array **Y** of size N,[136] is used to point to the next element on the single connected list. If an element is the last on the list, its position on the array **Y** will be a negative number equal to -1 to indicate the end of the list. These two arrays are shown in Figure 5.2 for the particles in Figure 5.1a representing the list shown in Figure 5.1b.

Travelling on the arrays **B** and **Y**, the element four is the first element on the list of the fifth row, and points to the element 1 in the array **Y** because this is the next element on the list as shown in Figure 5.1b and Figure 5.2. On the other hand, the element one is the last element on the list and in its position there is a -1, to indicate this.

In the second stage, the mapping of the all elements into the i_x cells is performed, for each row, as it is shown for row 5 in Figure 5.3.

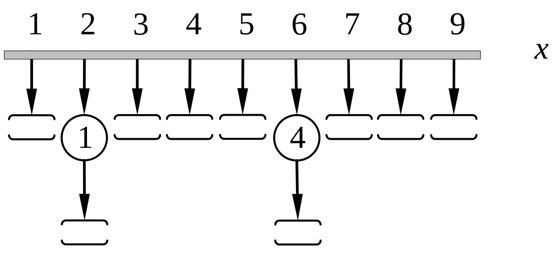


Figure 5.3: Single connected lists of columns for the 5th row. Figure modified from Munjiza[133]

Only cells with at least one DE are checked for interaction. If the row i (element i in the array **B**) is marked as 'new' a loop is performed over all the elements in it and the list is marked as 'old'. For each of the elements in the row, two 1D arrays are written: the "array **A** size n_x, where n_x is the number of cells in the x direction, and the array **X** size N".[136] These arrays are complementary of the **B** and **Y** arrays, and the same role applies to them for empty elements as is shown in Figure 5.4 for the 5th row.

	1	2	3	4	5	6	7	8	9
Array **A**	-1	1	-1	-1	-1	4	-1	-1	-1

	1	2	3	4	5	6
Array **X**	-1	-1	-1	-1	-1	-1

Figure 5.4: Arrays **A** (for the 5th row) and array **X.** Figures modified from Munjiza[133]

The NBS contact search algorithm utilizes a contact checking mask to avoid repetition in the contact search. For an element in column x_i and row y_i only two rows (in 2D) are necessary to perform the search y_i and y_{i-1} as shown in Figure 5.12. The main disadvantage of NBS compared with MR, is the need to rebuild the arrays database every time CD is performed.

5.3 MR algorithm

The contact detection time for the MR[148] algorithm is proportional to the number of DEs and is given by

$$T \propto N \tag{5.2}$$

It comprises of three algorithms a) Quick Sort (QS) b) MR-Sort, c) MR-Search. QS is not a linear algorithm and its CPU time is given by

$$T \propto N \log_2 (N) \tag{5.3}$$

This does not affect the linearity of MR as it is used to sort out the list only during the first time step. The MR algorithm is explained in depth in the boo.k by Munjiza et al.[141] In this section a short description of these algorithms is presented.

The MR domain is subdivided in a similar way to the NBS domain, into cubical cells,[148] as shown in Figure 5.6, in which the integerised coordinate of the elements is calculated using the equation 4.10. Each DE is approximated by a spherical container as was shown in Figure 4.3. The diameter of the biggest spherical container defines the size of the subdomains (cells). Once the elements are mapped they are added to the MR double connected list.

MR was originally implemented in C++ using objects. All the MR-objects in the system are arranged in a closed double connected list. Each one of these MR-objects has a pointer to the next MR-object in the list, a pointer to the preceding MR-object in the list, a pointer to a discrete element (it could also be an integer *id*) and its integerised position *V* using three integerised coordinates, as shown in Figure 5.5.

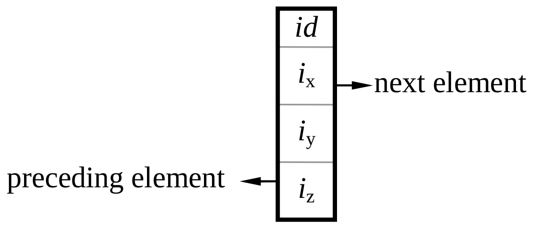


Figure 5.5: MR-object shown with *id*. Figure adapted from Munjiza et al.[148]

All the MR-objects are sorted[148] according to the same criteria mentioned in the previous chapter and given in equation 5.4

$$\begin{gathered}\left\{\left[\left(i_{zi} > i_{zj}\right)\right] or \left[\left(i_{zi} = i_{zj}\right) and \left(i_{yi} > i_{yj}\right)\right]\right\} \\ or \left\{\left[\left(i_{zi} = i_{zj}\right) and \left(i_{yi} = i_{yj}\right) and \left(i_{xi} > i_{xj}\right)\right]\right\}\end{gathered} \tag{5.4}$$

where i_{xi}, i_{yi} and i_{zi} are the integerised coordinates of the DE i and i_{xj}, i_{yj} and i_{zj} are the integerised coordinates of the DE j.

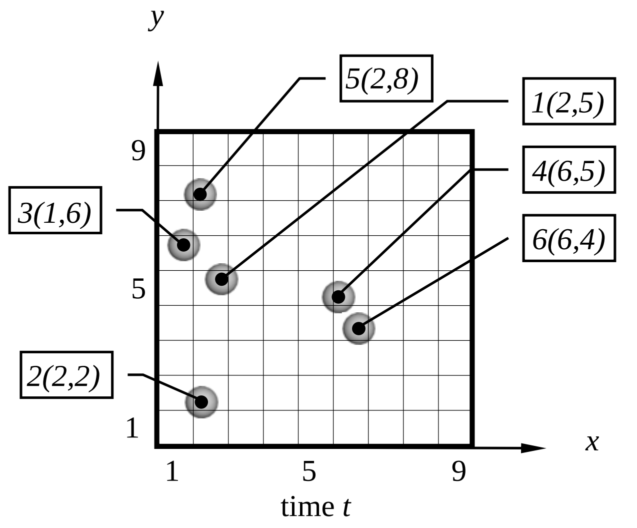


Figure 5.6: System at initial time t. Figure modified from Munjiza et al.[148]

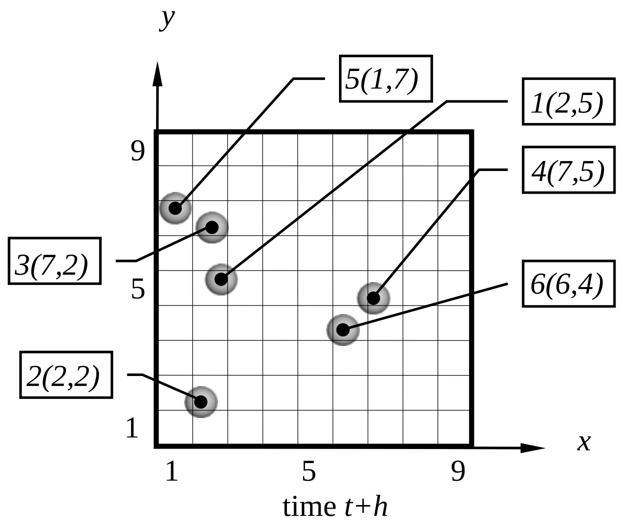


Figure 5.7: System at time $t+h$. Figure modified from Munjiza et al.[148]

The list has one element, List Head, that never changes its position. It is an MR-object whose integerised coordinates are equal to zero. A simple 2D example illustrates better the MR-list where the Figure 5.6 shows the system at time t and Figure 5.7 represents the system at time $t+h$. Initially the double connected list is not sorted as shown in Figure 5.8.

Quick Sort. Quick Sort[148] algorithm is applied in the first step to sort the entire list. The Quick Sort algorithm first identifies the biggest (B_{big}) and the smallest box (B_{small}) as shown in Figure 5.8. Following this, the middle box (B_{mid}) is calculated. Then traveling the double connecting MR-list from the beginning and the end simultaneously, each box is compared with the B_{mid}. If the box that is parsed from the beginning is bigger than B_{mid}, and the box that is parsed from the end is smaller than .the B_{mid}, these two cells are swapped. This first iteration ends when these two boxes are the same.

At this point the list is divided into two lists and the process starts again, "recursively until the size of the list"[148] after the division is one, the sorted list is shown in Figure 5.9. The total sorting time is given by

$$T \propto N \log_2 N \tag{5.5}$$

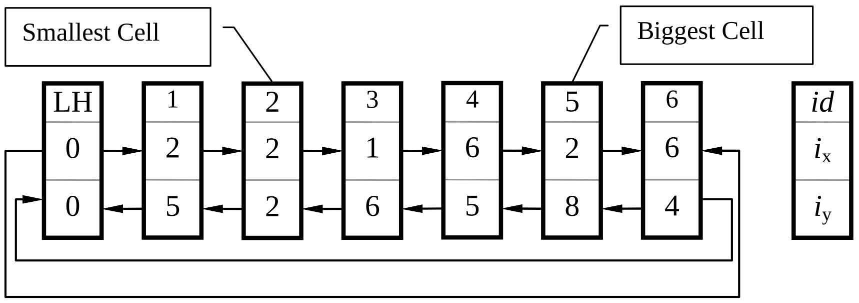


Figure 5.8: Double connected list, not sorted, time step t. Figure modified from Munjiza et al.[148]

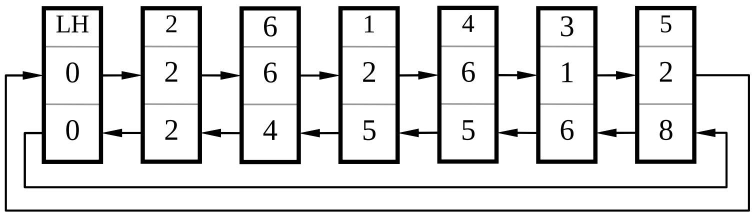


Figure 5.9: Double connected list, after applied Quick Sort algorithm, time step t. Figure modified from Munjiza et al.[148]

MR-Sort. It is based in the stability criteria present in discrete element simulations[133] to avoid unnecessary comparisons[148] in the MR-list. As the list in the time step $t + h$ is nearly sorted as shown in Figure 5.10. For example, element 5 has to swap positions with element 3 because it is smaller than element 3. The sorted list is shown in Figure 5.11. The list is parsed element by element until there is no MR-objects to swap.

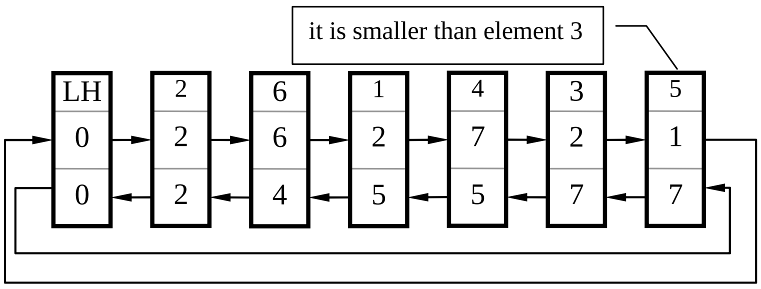


Figure 5.10: Unsorted list in the time step $t+h$. Figure modified from Munjiza et al.[148]

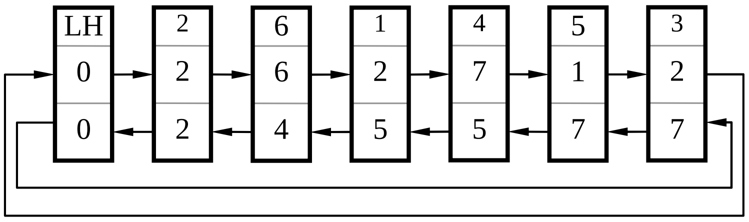


Figure 5.11: Sorted list in the time step $t+h$. Figure modified from Munjiza et al.[148]

MR-Search. "To avoid repetition in the contact detection"[148] MR-Search utilizes a contact checking mask as shown in Figure 4.16. To find the contacting couples between a particular target element, the list is parsed from the beginning until the smallest element in each particular row is found. The last element is the one that is bigger than the last element in the contact row as shown in Figure 5.12 for a 2D mask.

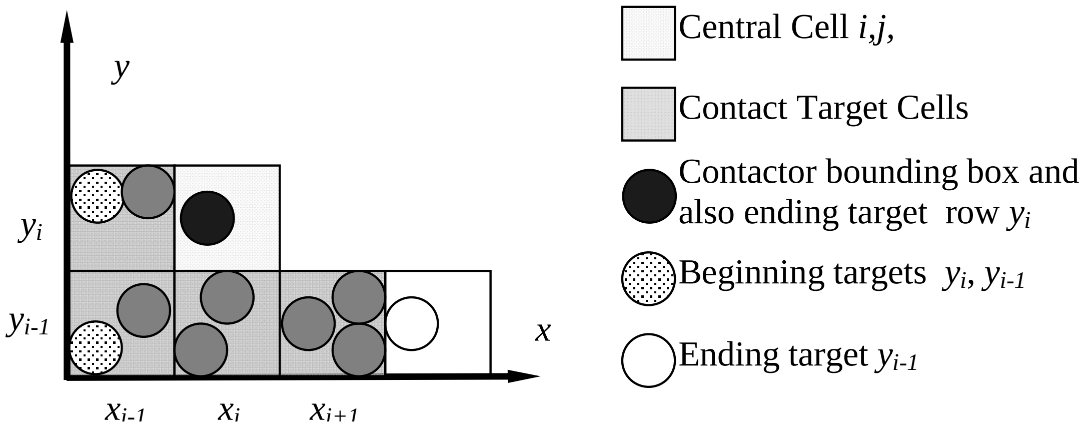


Figure 5.12: 2D contact mask. Figure adapted from Munjiza et al.[148]

The contact interaction will be calculated between the contactor and each one of the rows of the contact checking mask calculated from their beginning element, until the ending element.

$$beginning \leq target < ending \tag{5.6}$$

If the integerised coordinates are short integers numbers then, the total amount of RAM memory used per object is given by:

$$\begin{aligned}
V &= 3\,\text{short integer numbers} &&= 6\,\text{bytes}\\
P_{prv} &= 1\,\text{pointer} &&= 8\,\text{bytes}\\
P_{nxt} &= 1\,\text{pointer} &&= 8\,\text{bytes}\\
O &= 1\,\text{pointer} &&= 8\,\text{bytes}\\
M &= V + P_{prv} + P_{nxt} + O &&= 30\,\text{bytes}
\end{aligned} \tag{5.7}$$

If the pointer O to a discrete object is replaced by an integer id then the total RAM memory is $M = 26$ bytes.

5.4 MR-S algorithm

The principal difference between MR-Schiava (MR-S) and MR is the use of arrays instead of objects. This has the advantage of reducing the amount of RAM memory utilized by the algorithm. The main algorithms used by MR: Quick Sort, MR-Sort and MR-Search are utilized without great changes. In this implementation the data corresponding to the whole set of DEs is rearranged into five one-dimensional arrays of sizes equal to N plus one as shown in Figure 5.13.

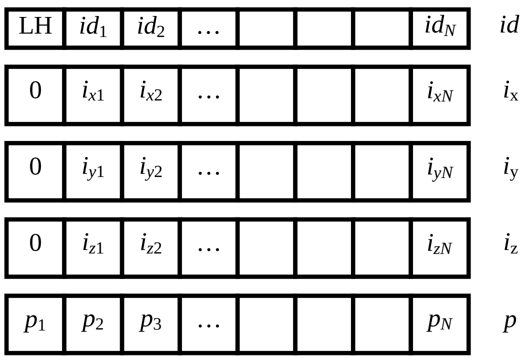


Figure 5.13: Data structure for the implementation of MR algorithms with arrays.

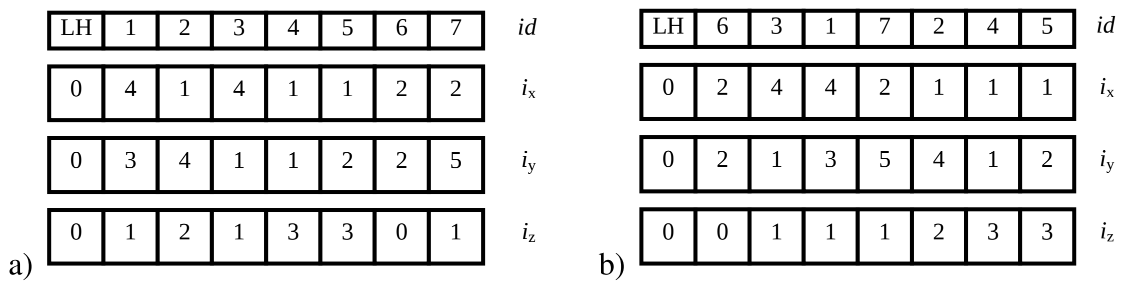
The first array could either be an array of integers corresponding to the *id* of each discrete element or it could be an array of pointers O to each discrete element.

If the integerised coordinates are short integer numbers the total amount of memory per element used is given by

$$\begin{aligned} V &= 3\,\text{short integer numbers} &&= 6\,\text{bytes} \\ O &= 1\,\text{pointer} &&= 8\,\text{bytes} \\ p &= 1\,\text{integer} &&= 4\,\text{bytes} \\ M &= V + O + p &&= 18\,\text{bytes} \end{aligned} \tag{5.8}$$

If the pointer O to a discrete object is replaced by an integer *id* then $M = 14$ bytes. The percentage of reduction for the case of using a pointer O of MR-S respect of MR is 40%.

Quick Sort. The pointer to the preceding MR-object in the MR algorithm shown in Figure 5.5 is only unitised in the first time step when the list is ordered using QS. Using arrays, there is no need to record the previous or the next element. For ordering the arrays using Quick Sort,l only the array corresponding to the *id* (or O) and the three arrays corresponding to V are used as shown in Figure 5.14.



a)

LH	1	2	3	4	5	6	7	*id*
0	4	1	4	1	1	2	2	i_x
0	3	4	1	1	2	2	5	i_y
0	1	2	1	3	3	0	1	i_z

b)

LH	6	3	1	7	2	4	5	*id*
0	2	4	4	2	1	1	1	i_x
0	2	1	3	5	4	1	2	i_y
0	0	1	1	1	2	3	3	i_z

Figure 5.14: Arrays used by Quick Sort. a) Before ordering b) After ordering

After the Quick Sort Algorithm is implemented, all the objects on MR-S are spatially ordered. For MR-Sort and MR-search only a pointer to the object with V equal or bigger to the current element is needed. This is achieved using an array of pointers p which is initialised after the list is ordered as shown in Figure 5.15.

If the elements change their coordinates the values of V are updated to i_x, i_y and i_z arrays. As the list is no longer in order, it has to be updated.

It is possible to avoid using the array p to update the list by moving data stored in *id*, i_x, i_y and i_z arrays from the old position to a new position. This would be CPU inefficient. It is more CPU time-efficient to update only one array p than to update 4 arrays *id*, i_x, i_y and i_z.

a)

LH	6	3	1	7	2	4	5	*id*
0	2	4	4	2	1	1	1	i_x
0	2	1	3	5	4	1	2	i_y
0	0	1	1	1	2	3	3	i_z
1	2	3	4	5	6	7	0	p

b)

$$\begin{tikzcd}
LH \arrow[r] & 6 \arrow[r] & 3 \arrow[r] & 1 \arrow[d] \\
5 \arrow[u] & 4 \arrow[l] & 2 \arrow[l] & 7 \arrow[l]
\end{tikzcd}$$

Figure 5.15: a) Arrays at time step t b) Schematic view of the elements order at time t.

MR-sort. MR-sort updates the array p as shown in Figure 5.16a, every time CD is performed. All the other arrays remain unchanged.

a)

LH	6	3	1	7	2	4	5	*id*
0	2	4	4	2	1	1	2	i_x
0	2	1	3	5	4	1	2	i_y
0	1	1	2	1	2	3	2	i_z
2	4	1	5	7	6	0	3	p

b)

$$\begin{tikzcd}
LH \arrow[r] & 3 \arrow[r] & 6 \arrow[r] & 7 \arrow[d] \\
4 \arrow[u] & 2 \arrow[l] & 1 \arrow[l] & 5 \arrow[l]
\end{tikzcd}$$

Figure 5.16: a) Arrays at time step $t+h$ b) Schematic view of the elements order at time step $t+h$.

MR-search. MR-search is applied in the same way as for the previous algorithm using the contact mask shown in Figure 5.12 on the array p.

Chapter 6

NOVEL CONTACT DETECTION ALGORITHM FOR BODY-POINT

6.1 Introduction

The algorithms covered in the previous chapters are designed to work with bodies of similar size. BBTS, NBS and MR are heavily penalised by the size of the biggest element (size of the biggest boundary box) related to the size of the smallest element, resulting in them not being suited to performing body-point contact. In this chapter a Novel algorithm based on the BBTS, NBS and the MR is introduced. This Novel algorithm is denominated as MS MunjizaSchiava.

Parallelization while present the advantage of discretise the domain into smaller subdomains[1] , the amount of objects in each domain changes function of time. As the objects move from one process to its neighbours, it is not possible to build an MR-array and keep it throughout the simulation.

The basic flow diagram of CD is shown in Figure 3.3 in chapter 3, reproduced here in Figure 6.1 for clarity. In this work, CD is not performed each time step, it is only performed when:

- the number of steps since the last CD ($n_{\triangle CD}$) is equal to the number of steps chosen by the user (n_{CD}) i.e. $n_{\triangle CD} = n_{CD}$
- there is a change in the number of objects in the processor. This event is usually triggered by the parallelization algorithm.

[1]In the case of the MPI parallelization employed in this work. For other types of parallelization please refer to chapter 3

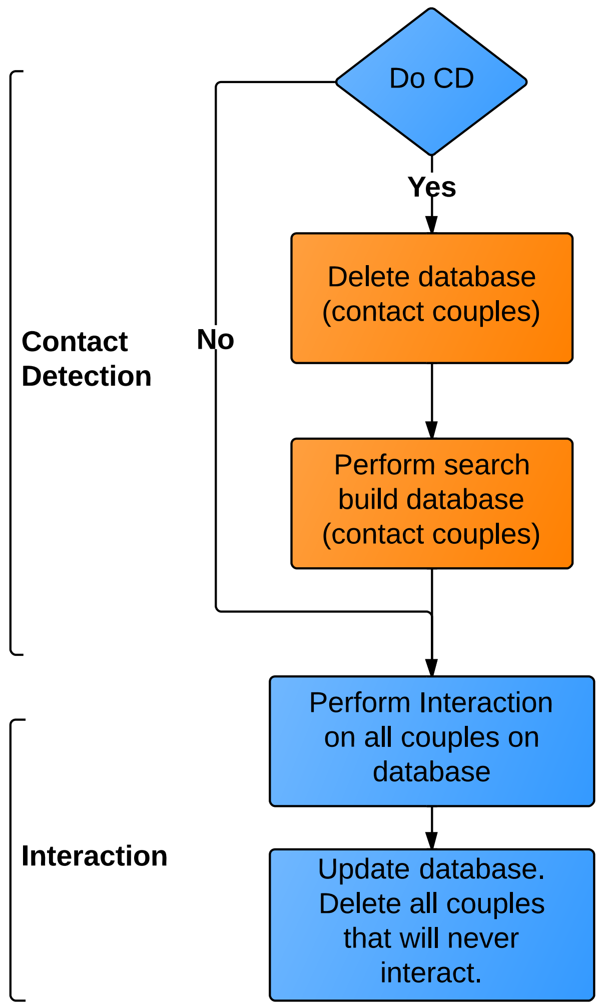

Do CD
Yes
No
Delete database (contact couples)
Perform search build database (contact couples)
Contact Detection
Perform Interaction on all couples on database
Update database. Delete all couples that will never interact.
Interaction

Figure 6.1: Flow diagram CD and interaction using database of contact couples.

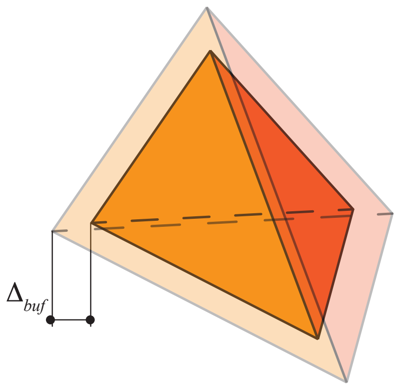


Figure 6.2: FE with its dimensions increased by $\triangle_{buf}$.

The search performed during CD has to find all elements that are in contact, and the ones that may be in contact in n_{CD} steps. To this aim the dimensions of any contactor (FE) are increased by $\triangle_{buf}$ as shown in Figure 6.2.

In the second section of this chapter, a short description of this novel algorithm is presented, followed by the improvements on the BT necessary to perform MS. Then the complete implementation is presented in 2D and 3D. Finally, a sequential (one process) and parallel numerical test is presented.

6.2 Novel CD algorithm

This algorithm is based on the space decomposition of $\mathbb{R}$ into squares in 2D and boxes in 3D, denominated $\mathbb{R}_s$. The size s of the space decomposition does not depend on the biggest element, but is a parameter chosen by the user, usually proportional to $\triangle_{buf}$ where $\triangle_{buf}$ is given by

$$\triangle_{buf} = 2\, n_{CD}\, v_{max} \triangle t \tag{6.1}$$

where n_{CD} represents the number of steps between contact detection, v_{max} is the maximum velocity of any object and $\triangle t$ is delta time. While the size of the space decomposition s is

$$s \propto \triangle_{buf} \tag{6.2}$$

The mapping of $\mathbb{R}$ into $\mathbb{R}_s$ is performed using simplified formulae based on the equation 4.10 and is given by

$$\begin{aligned} i_x &= Int\left(\tfrac{x}{s}\right) \\ i_y &= Int\left(\tfrac{y}{s}\right) \\ i_z &= Int\left(\tfrac{z}{s}\right) \end{aligned} \tag{6.3}$$

where x, y and z are the coordinates in $\mathbb{R}$, s is the subdomain (cell) size and i_x, i_y and i_z are the x, y and z integerised coordinates respectively.

The key idea is to map all interaction points (targets) shown in Figure 6.3a, into $\mathbb{R}_s$ as shown in Figure 6.3b.

Then each of the bodies (contactors) is mapped/transformed into a rectangle in 2D or a rectangular cuboid in 3D. This abstraction from the original shape of the contactor simplifies the algorithm. In 3D only six integerised numbers are required to define any contactor, regardless of its shape. The minimum integerised coordinate and the maximum integerised coordinates are calculated as

$$\begin{aligned} i_{x_{min}} &= Int\left(\frac{x_{min}-\triangle_{buf}}{s}\right) \\ i_{y_{min}} &= Int\left(\frac{y_{min}-\triangle_{buf}}{s}\right) \\ i_{z_{min}} &= Int\left(\frac{z_{min}-\triangle_{buf}}{s}\right) \end{aligned} \tag{6.4}$$

$$\begin{aligned} i_{x_{max}} &= Int\left(\frac{x_{max}+\triangle_{buf}}{s}\right) \\ i_{y_{max}} &= Int\left(\frac{y_{max}+\triangle_{buf}}{s}\right) \\ i_{z_{max}} &= Int\left(\frac{z_{max}-\triangle_{buf}}{s}\right) \end{aligned} \tag{6.5}$$

The area in 2D, or volume in 3D, created by these limits is denominated as $\mathbb{R}_{s\_CD}$ shown in Figure 6.4a. It is worth noticing that in no other way are the bodies mapped into $\mathbb{R}_s$ apart from the *min* and *max* integerised coordinates. Searching for contacting couples requires finding the non-empty cells shown in Figure 6.4b and performing interaction tests between each target (point) and the contactor (FE).

6.3 Implementation using BBTS

Different data bases can be used to store and retrieve interaction points. Arrays and matrices are the most efficient method, in terms of memory and velocity, of storing data[2] . Yet their lack of flexibility make them ill-suited for object-oriented code. Even when it is possible to allocate memory during simulation time[3], it is not possible to add more memory to an existing array. On the other hand, the binary tree coded using objects is flexible, as it is possible to add objects on the fly, providing the benefit of sequential storage.

Each target (contact point) on the tree is added using the spatial ordering criterion.[148] previously introduced in chapter 4. A node i is greater than a node j i.e. $V_i > V_j$ if:

$$\begin{gathered} \left\{\left[\left(i_{zi} > i_{zj}\right)\right] or \left[\left(i_{zi} = i_{zj}\right) and \left(i_{yi} > i_{yj}\right)\right]\right\} \\ or \left\{\left[\left(i_{zi} = i_{zj}\right) and \left(i_{yi} = i_{yj}\right) and \left(i_{xi} > i_{xj}\right)\right]\right\} \end{gathered} \tag{6.6}$$

[2]In a similar way to the implementation used in NBS

[3]Using **malloc/new** function in C/C++, and deleting it using **free/delete**. It is never a good idea to continuously retrieve memory from the system, only to return it later. This could lead to fragmentation memory problems.

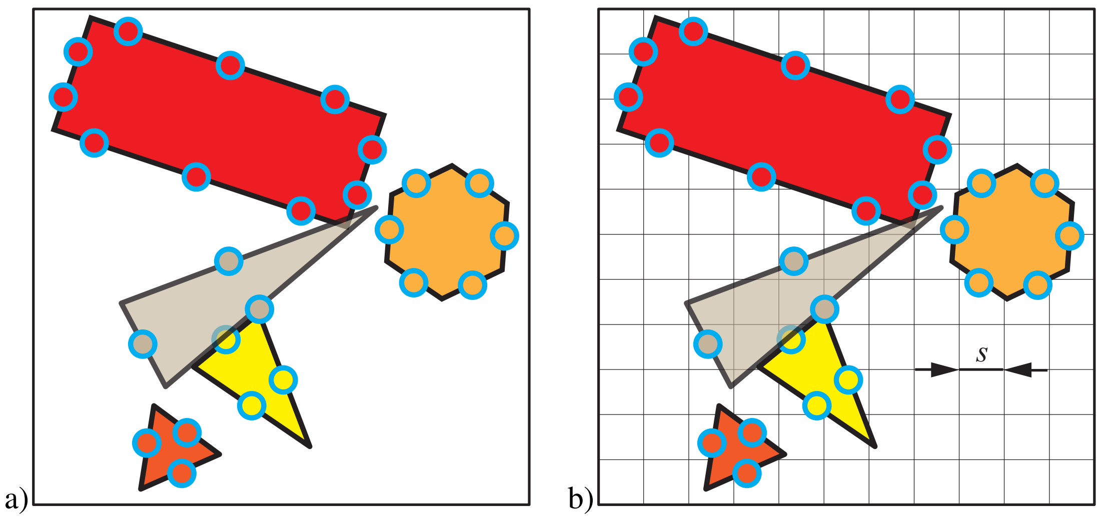


Figure 6.3: From $\mathbb{R}$ into $\mathbb{R}_s$a)Different 2D FEs, with different combinations of contact points. The contact points, are the circles on the border of the FEs b)Mapping of the contact points into $\mathbb{R}_s$

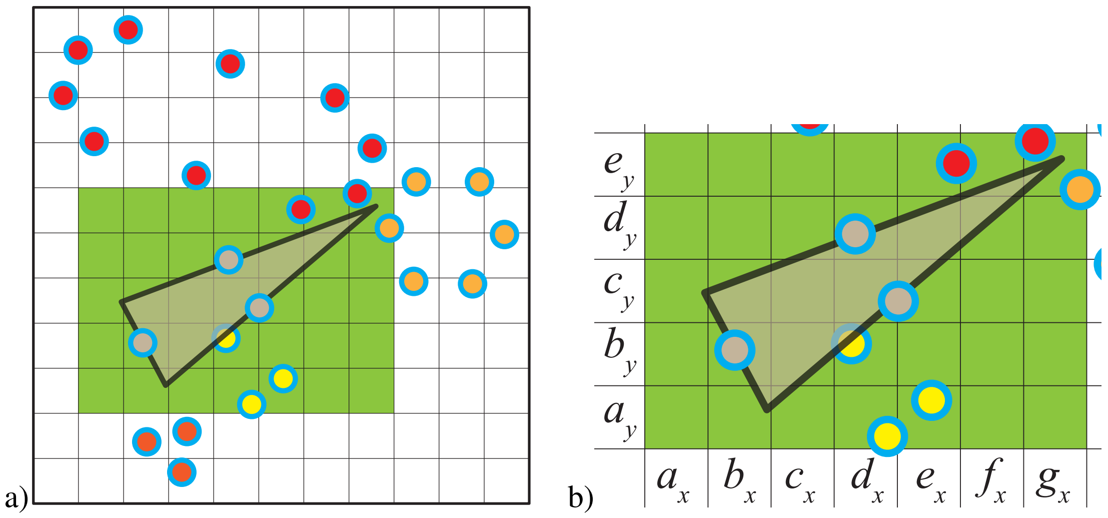


Figure 6.4: Contact detection subdomain $\mathbb{R}_{s\_CD}$. a)Area on $\mathbb{R}_s$ relevant to the FE. b)Zoomed area. Points belonging to the same FE are skip by the interaction routine.

Using an unmodified BBTS to perform CD-search would lead to the algorithm 6.1. For each contactor, it would be necessary to perform N operations equal to

$$N = N_x N_y \tag{6.7}$$

$$N_x = i_{x_{max}} - i_{x_{min}} + 1 \tag{6.8}$$

$$N_y = i_{y_{max}} - i_{y_{min}} + 1 \tag{6.9}$$

where N_x is the number of subdomains (cells) in the x direction, N_y is the number of subdomains in y direction. The extension into 3D is trivial. As was previously discussed the time T to retrieve N objects on a binary tree is

$$T \propto N \log(N) \tag{6.10}$$

Algorithm 6.1 Contact search body/point on $\mathbb{R}_s$ using an unmodified BT. "$\rightarrow$" has the C++ meaning

```
1: integer $iLv\_x_{min}, iLv\_y_{min}$ ▷ Min integerised coordinate of contactor $Con_{Cur}$ on $\mathbb{R}_s$
2: integer $iLv\_x_{max}, iLv\_y_{max}$ ▷ Max integerised coordinate of contactor $Con_{cur}$ on $\mathbb{R}_s$
3: for $(iLv\_y_{cur} = iLv\_y_{min}; iLv\_y_{cur} < (iLv\_y_{max}+1); ++iLv\_y_{cur})$ do
4:     for $(iLv\_x_{cur} = iLv\_x_{min}; iLv\_x_{cur} < (iLv\_x_{max}+1); ++iLv\_x_{cur})$ do
5:         $Tar_{cur} = BT \rightarrow FndEqYX(iLv\_y_{cur}, iLv\_x_{cur})$ ▷ FindEqual. Target:list of pnts
6:         $vS\_Interaction(Con_{cur}, Tar_{cur})$ ▷ Interaction (add to database)
7:     end for
8: end for
```

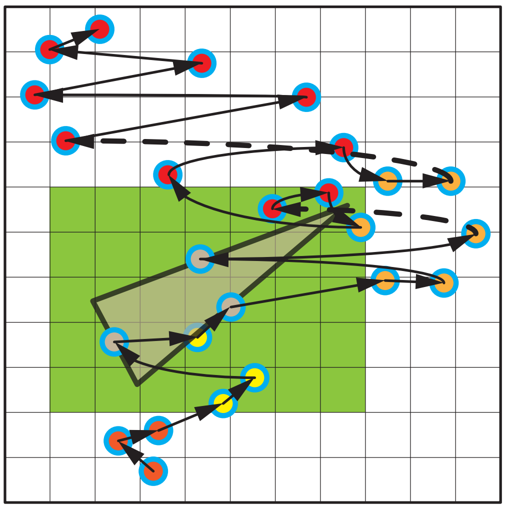


Figure 6.5: Hierarchy of contact points on domain $\mathbb{R}_s$. Starting from bottom to top and left to right, the hierarchy increases.

This algorithm would be highly inefficient and impractical. As the spatial ordering criterion is used, there is a 1D hierarchy of points on $\mathbb{R}_s$ depending on their spatial position in a similar way to the objects on an MR list. This is best illustrated by the

Figure 6.5 where each node i points to the node j greater than or equal to itself in terms of V.

The exploitation of the hierarchy list on the subdomain $\mathbb{R}_{s\_CD}$, thereby avoiding operations on empty cells is the key idea behind this novel algorithm.

6.3.1 Modifications to the BBTS

The natural hierarchy created by the spatial ordering criterion cannot be exploited by a standard BBTS. The modification to the BBTS aims to:

- Reduce the amount of operations required to find the smallest greater node of any node i.e. the *next* node
- Avoid searching for, or operating on, empty cells

In a standard BBTS, finding the smallest greater node t of a node d, implies a binary *parse* from the tree's root as shown in Figure 6.6, and in algorithm 6.2.

Algorithm 6.2 Search smallest greater than V_{search} procedure. "$\rightarrow$" has the C++ meaning

```
 1: $Nod_{cur} = \text{Null}$                                   ▷ Pointer to current node
 2: $Nod_{smaGre} = \text{Null}$                                ▷ Pointer to smallest greater node
 3: $Nod_{cur} = BT \rightarrow GetFrs()$                       ▷ GetFirstNode
 4: while $(Nod_{cur}! = \text{Null})$ do
 5:     if $BT \rightarrow IsGreThan(Nod_{cur}, V_{search})$ then   ▷ i.e. $(V_{cur} > V_{search})$
 6:         $Nod_{smaGre} = Nod_{cur}$
 7:         $Nod_{cur} = BT \rightarrow GetRigSon(Nod_{cur})$       ▷ go to the rigth son $(R_s)$
 8:     else
 9:         $Nod_{cur} = BT \rightarrow GetLefSon(Nod_{cur})$       ▷ go to the left son $(L_s)$
10:     end if
11: end while
12: return pointer to $Nod_{smaGre}$
```

It is necessary to transform the binary tree into a 1D list, avoiding any kind of parse to find the next object. To achieve this a new pointer C_n is added to the smallest greater object (next node) as shown in Figure 6.7. The binary tree built using this new node is denominated as the MS-tree.

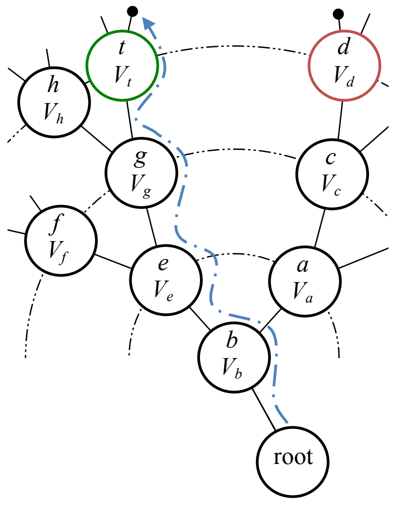


Figure 6.6: Parsing (light blue path) of a binary tree, to find the smallest greater node (green) than node d (red).

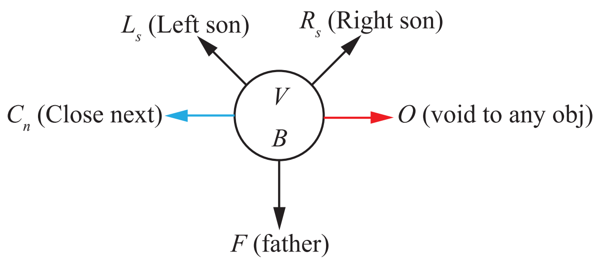


Figure 6.7: New node structures (in black) are highlighted pointers necessary to add, delete and balance the BT. The pointer to the next object C_n is highlighted in blue and a general pointer to a void* object O (in C/C++ void pointers can be casted into other objects) is highlighted in red.

Updating the pointer C_n, when a new object is added to the MS-tree, does not imply a heavy over-burn of CPU time. Actually, it is a simple procedure that takes advantage of the BS-tree logic. The modification to the previous add algorithm 4.1 is shown in the algorithm 6.3.

This is similar to the case when an object is deleted and the C_n pointer has to be updated in the remaining nodes on the tree, while the balancing algorithms remains without changes.

The BTTS algorithm, presented in Chapter 4, does not allow two or more nodes with the same key V, adding nodes with the same key to the same node as shown in Algorithm 4.2. The reason for this is the way in which a BBTS-search is performed, searching one at a time, for all possible boxes on the contacting mask shown in Figure 4.16. This search implies searching for nodes that may not exist.

This novel algorithm, departs from the BBTS as there is no search for particular nodes in MS-search. In the MS-tree there is no limitation for one or more nodes to have the same key value V. Nodes with the same key added using algorithm 6.3 coexist on the MS-tree, building what is in fact a binary tree and at the same time an organised 1D list. The MS-tree is best described by the Figure 6.8.

Algorithm 6.3 Add procedure modifications to set pointer to C_s. Nodes with the same V coexist on the tree. "$\rightarrow$" has the C++ meaning

```
1:  $Nod_{smaGre} = \text{Null}$                                    ▷ Pointer smallest greater node
2:  $Nod_{bigSma} = \text{Null}$                                    ▷ Pointer biggest smaller node
3:  $Nod_{prv} = \text{Null}$                                       ▷ Pointer node previous
4:  $Nod_{cur} = BT \rightarrow GetFrs()$                           ▷ Get first node on tree
5:  while $(Nod_{cur}! = \text{Null})$ do
6:      $Nod_{prv} = Nod_{cur}$
7:      if $BT \rightarrow IsGreThan(Nod_{cur}, V_{new})$ then      ▷ IsGreaterThan i.e. $(V_{cur} > V_{new})$
8:          $Nod_{smaGre} = Nod_{cur}$
9:          $Nod_{cur} = BT \rightarrow GetRigSon(Nod_{cur})$       ▷ go to the rigth son $(R_s)$
10:     else                                                        ▷ i.e. equal or less than $V_{new}$
11:         $Nod_{bigSma} = Nod_{cur}$
12:         $Nod_{cur} = BT \rightarrow GetLefSon(Nod_{cur})$       ▷ go to the left son $(L_s)$
13:     end if
14: end while
15: $Nod_{cur} = Nod_{prv}$
16: create $Nod_{new}$ set variables on node
17: if $(Nod_{bigSma}! = \text{Null})$ then
18:     $Nod_{bigSma} \rightarrow AddNxt(Nod_{new})$
19: else
20:     $Nod_{new} \rightarrow AddNxt(Nod_{smaGre})$
21: end if
```

6.4 CD-MS

The CD-MS general procedure to perform Contact Detection is:

1. Clean database (contact couples)

2. MS-load: Load all targets (interaction points) into MS-tree

3. For each contactor (FE) perform MS-search. Add to database

4. Clean MS-tree

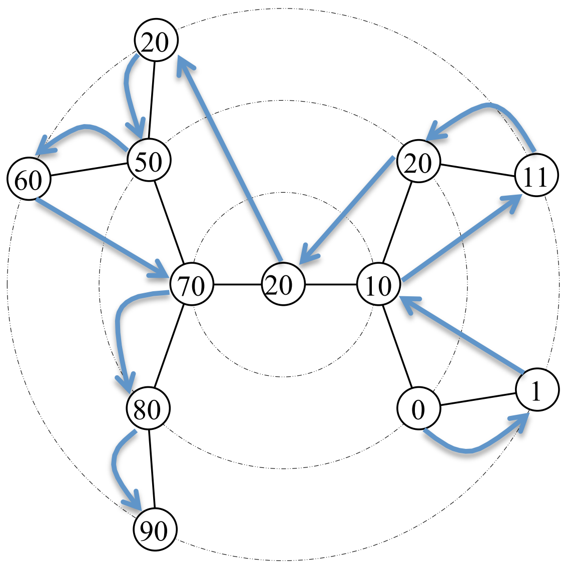


Figure 6.8: MS-tree: pointers to the *next* object are highlighted in light blue,

MS-load. After the contact interaction data base is cleaned, all interaction points are loaded into the MS-tree (algorithm 6.3) using integerised coordinates (equation 6.3) and the spatial ordering criterion (equation 6.6). As the interaction points are loaded the MS-tree is balanced using the balancing procedure described in chapter 4.

Algorithm 6.4 MS-search 2D.

```
1: **integer** $iLv\_x_{min}, iLv\_y_{min}$ ▷ Min integerised coordinate of contactor $Con_{Cur}$ on $\mathbb{R}_s$
2: **integer** $iLv\_x_{max}, iLv\_y_{max}$ ▷ Max integerised coordinate of contactor $Con_{cur}$ on $\mathbb{R}_s$
3: **integer** $iLv\_x_{cur}, iLv\_y_{cur}$    ▷ Current integerised coordinates
4: $Con_{cur}$    ▷ Pointer to contactor (finite element) current
5: $Nod_{cur} = \text{Null}$    ▷ Pointer to current node
6: $Nod_{lst} = \text{Null}$    ▷ Pointer to last node
7: $Nod_{cur} = BT \rightarrow FndSmaGreY\,(iLv\_y_{min} - 1)$    ▷ FindSmallestGreater_Y
8: $iLv\_y_{cur} = BT \rightarrow GetY\,(Nod_{cur})$    ▷ GetYcoordiante of node current
9: **while** $(iLv\_y_{cur} < iLv\_y_{max})$ **do**
10:     $Nod_{cur} = BT \rightarrow FndSmaGreYX\,(iLv\_y_{cur}, (iLv\_x_{min} - 1))$
11:     $Nod_{lst} = BT \rightarrow FndSmaGreYX\,(iLv\_y_{cur}, iLv\_x_{max})$
12:     **while** $(Nod_{cur}! = Nod_{lst})$ **do**
13:         $vS\_Interaction\,(Con_{cur}, Nod_{cur})$    ▷ If there is contact add to data base
14:         $Nod_{cur} = BT \rightarrow GetNxt\,(Nod_{cur})$    ▷ GetNextNodeOfCurrentNode
15:     **end while**
16:     $Nod_{cur} = BT \rightarrow FndSmaGreY\,(iLv\_y_{cur})$    ▷ FindSmallestGreater_Y
17:     $iLv\_y_{cur} = BT \rightarrow GetY\,(Nod_{cur})$    ▷ GetYcoordiante of node current
18: **end while**
```

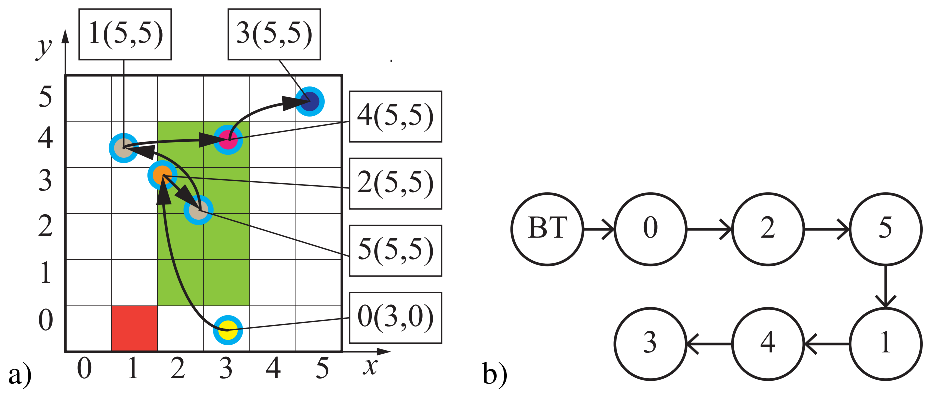


Figure 6.9: Interaction points, belonging to different finite elements. a)In $\mathbb{R}_s$, each arrow points to the next next node. The subdomain $\mathbb{R}_{s\_CD}$ of a generic contactor is highlighted in green. b)1D sorted “list” produced by BT

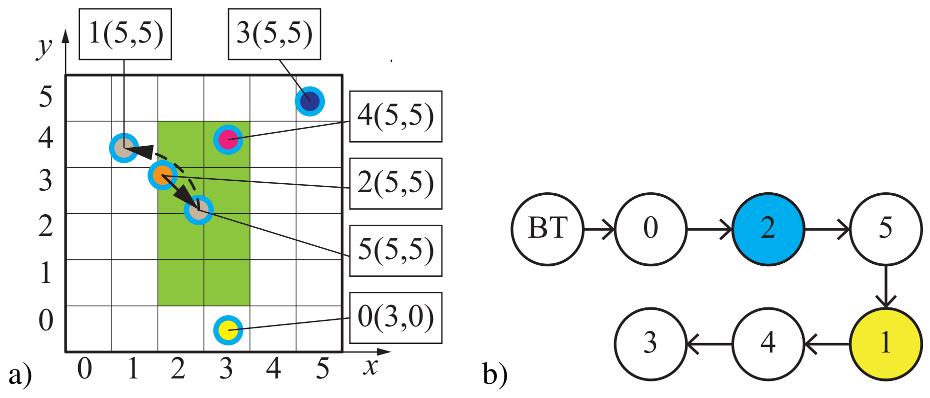


Figure 6.10: First step MS-search. a)Only nodes that will be “travelled” have arrows pointing to the next element. b)The first node to perform an interaction is highlighted in blue, while the first node to be skipped is highlighted in yellow.

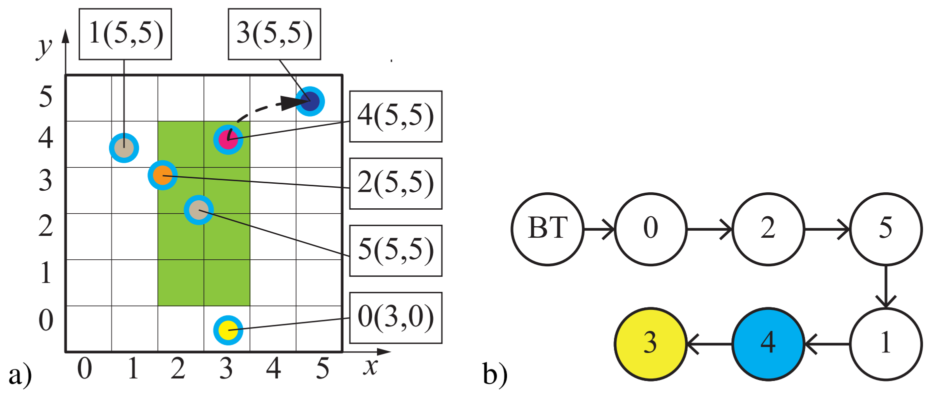


Figure 6.11: Last step MS-search. a)Only nodes that will be “travelled” have arrows pointing to the next element. b)The first node to perform an interaction is highlighted in blue, while the first node to be skipped is highlighted in yellow.

Algorithm 6.5 MS-search 3D.

```
1: **integer** $iLv\_x_{min}, iLv\_y_{min}, iLv\_z_{min}$ ▷ Min integerised coordinate of $Con_{Cur}$ on $\mathbb{R}_s$
2: **integer** $iLv\_x_{max}, iLv\_y_{max}, iLv\_z_{max}$ ▷ Max integerised coordinate of $Con_{cur}$ on $\mathbb{R}_s$
3: **integer** $iLv\_x_{cur}, iLv\_y_{cur}, iLv\_z_{cur}$ ▷ Current integerised coordinates
4: **integer** $iLv\_Z$ ▷ Current PlaneZ
5: $Con_{cur}$ ▷ Pointer to contactor (finite element) current
6: $Nod_{cur} = \text{Null}$ ▷ Pointer to current node
7: $Nod_{lst} = \text{Null}$ ▷ Pointer to last node
8: $Nod_{cur} = BT \rightarrow FndSmaGreZ\,(i_{z_{min}} - 1)$ ▷ FindSmallestGreater_Z
9: $iLv\_i_{z_{cur}} = BT \rightarrow GetZ\,(Nod_{cur})$ ▷ GetZcoordiante of node current
10: **while** $(iLv\_i_{z_{cur}} < iLv\_i_{z_{max}})$ **do**
11:     $Nod_{cur} = BT \rightarrow FndSmaGreZY\,(iLv\_i_{z_{cur}}, (iLv\_i_{y_{min}} - 1))$
12:     $iLv\_i_{z_{cur}} = BT \rightarrow GetZ\,(Nod_{cur})$ ▷ GetZcoordiante of node current
13:     $iLv\_i_{y_{cur}} = BT \rightarrow GetY\,(Nod_{cur})$ ▷ GetYcoordiante of node current
14:     $iLv\_Z = iLv\_i_{z_{cur}}$ ▷ Save current plane Z
15:     **while** $((iLv\_y_{cur} < iLv\_y_{max}) \&\& (iLv\_z_{cur} == iLv\_Z) \&\& (iLv\_z_{cur} <= iLv\_z_{max}))$ **do**
16:         $Nod_{cur} = BT \rightarrow FndSmaGreZYX\,(iLv\_i_{z_{cur}}, iLv\_i_{y_{cur}}, (iLv\_i_{x_{min}} - 1))$
17:         $Nod_{lst} = BT \rightarrow FndSmaGreZYX\,(iLv\_i_{z_{cur}}, iLv\_i_{y_{cur}}, iLv\_i_{x_{max}})$
18:         **while** $(Nod_{cur}! = Nod_{lst})$ **do**
19:             $vS\_Interaction\,(Con_{cur}, Nod_{cur})$ ▷ If there is contact add to data base
20:             $Nod_{cur} = BT \rightarrow GetNxt\,(Nod_{cur})$ ▷ GetNextNodeOfCurrentNode
21:         **end while**
22:         $Nod_{cur} = BT \rightarrow FndSmaGreZY\,(iLv\_z_{cur}, iLv\_y_{cur})$
23:         $iLv\_z_{cur} = BT \rightarrow GetZ\,(Nod_{cur})$
24:         $iLv\_y_{cur} = BT \rightarrow GetY\,(Nod_{cur})$
25:     **end while**
26:     $Nod_{cur} = BT \rightarrow FndSmaGreZ\,(iLv\_Z)$
27:     $iLv\_z_{cur} = BT \rightarrow GetZ\,(Nod_{cur})$
28: **end while**
```

MS-search. The MS contact search between contactor and target is explained in 2D in the rest of the section and is shown in algorithm 6.4. While the 3D algorithm is presented at the end of the section.

Only one BT function "search smallest greater than" shown in algorithm 6.2 is employed. The advantage of this is that all searches are on existing nodes and not on specific cells. If there are no nodes in a particular row or column, no operation is performed on them.

For each contactor, the first step in algorithm 6.4 is to find the first non-empty row y that is between the limits of $\mathbb{R}_{s\_CD}$. In the generic example shown in Figure 6.9, this means finding the first node of any row bigger than 0 (red square in Figure 6.9a). Then the first node in that particular row that belongs to the subdomain $\mathbb{R}_{s\_CD}$ is found and skipped as shown in Figure 6.10.

This procedure is repeated as shown in Figure 6.11 until there are no more nodes in $\mathbb{R}_{s\_CD}$. All contact couples are stored using an MS-tree[4] to be retrieved later when contact interaction is calculated.

As all contact nodes in the MS-tree are in order forming, a 1D list, only the first and the last nodes are binary searched. Following this, a linear walk on the 1D list is performed.

The 3D algorithm is shown in Algorithm 6.5. It follows the same principles as for the 2D algorithm performing "one" more search for the z component. For each non-empty z plane a 2D search is performed.

6.5 Test

This new algorithm is tested in sequential (one core) and in parallel (two and four cores) on a Multicore PC DELL Precision T5400 with one processor of four cores and 32 GB of RAM. The parallel domain decomposition into four processes is shown in Figure 6.12.

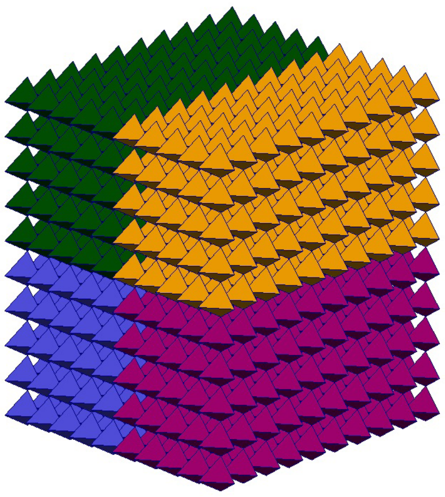


Figure 6.12: Raster 1000 DEs domain decomposition into four processes.

The test consists of a raster of simple discrete elements (DEs) made of one finite element (FE), each with elastic modulus E=4 MPa, Poisson's ratio $\nu = 0.45$ and penalty

[4]Also a simple connected list could be used

$\sigma_p = 400$ MPa. Each FE has one interaction point centred on each of its faces. Random velocities are assigned at a time equal to zero from 0 m/s to 137 m/s. A spherical boundary with penalty $\sigma_p = 400$ MPa is imposed to keep all FEs interacting with each other. The development of the simulation for the raster of 1000 elements is shown in Figure 6.14.

The total time CD-MS (clean data base, MS-load, MS-search, add contact couples to database, clean MS-tree) is shown in Figure 6.13 for the sequential and parallel numerical test. The excellent qualities of the CD-MS algorithm were tested up to 1.331 million of DEs and 5.324 million of interaction points.

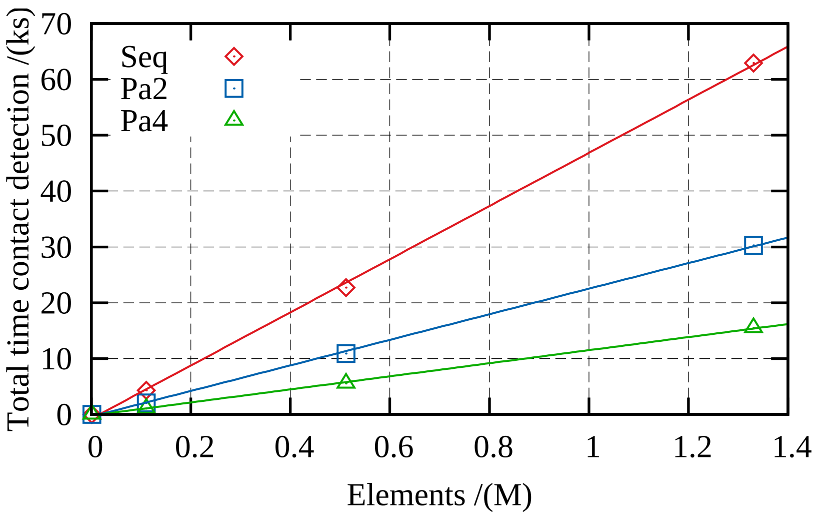


Figure 6.13: Total CD-MS time. The continuous lines are linear regressions.

CD-MS is not a linear algorithm, but still presents quasi-linear behaviour in the tested range. The non-linearity can also be appreciated as well in table 6.1 where values are less than the theoretical expected values of 45% for two processors and 25% for four processors.

Number of elements (M)	2 processors	4 processors
0.001	46.3%	23.4%
0.111	48.0%	24.3%
0.512	47.9%	24.4%
1.331	48.1%	24.6%

Table 6.1: Average total CD-MS time percentage of sequential time

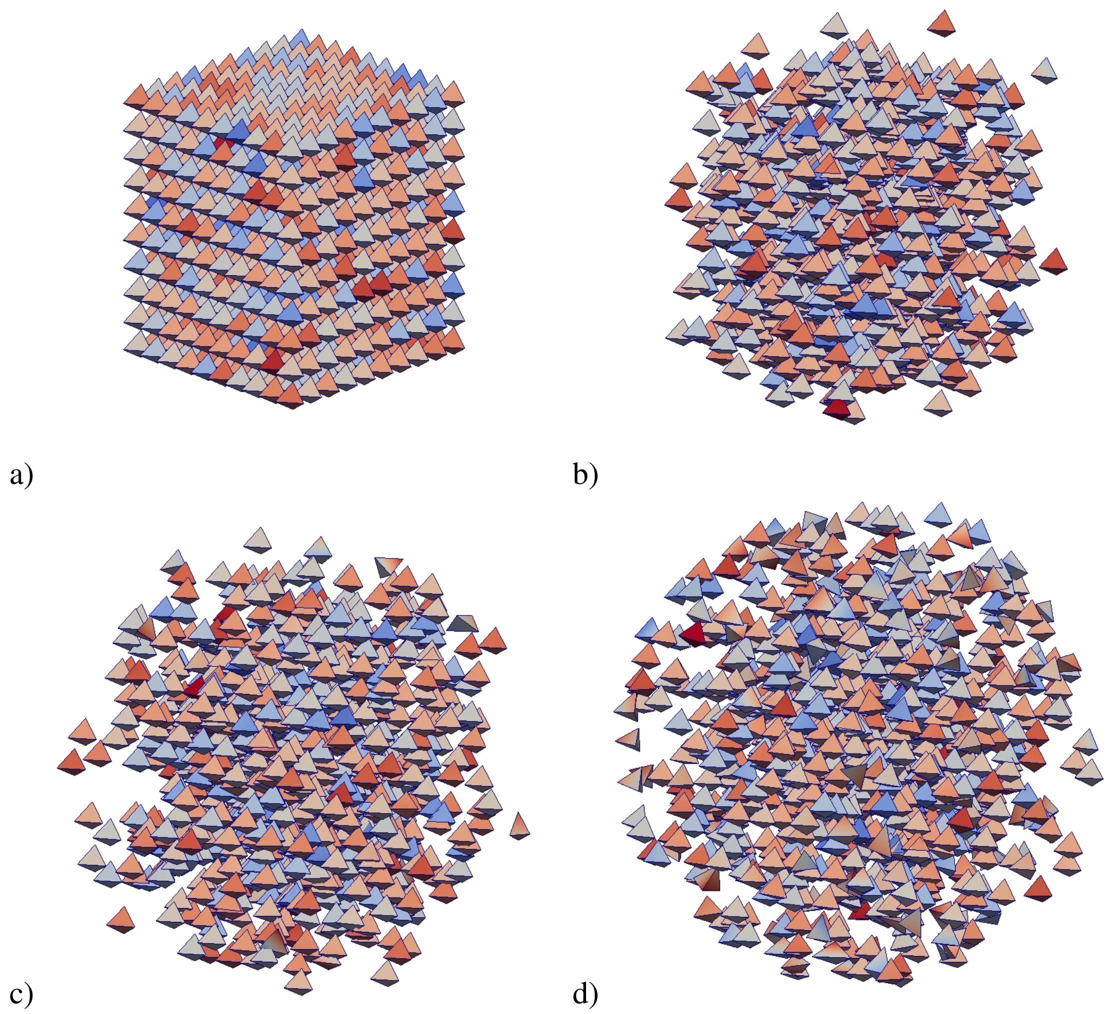


Figure 6.14: Raster 1000 DEs (parallel four processors). The colours represent the magnitude of the velocities. From 0m/s to 137m/s a)Time 0.0ms. b)Time 0.32ms. c)Time 0.60ms. d)Time 1.01ms.

6.6 Conclusions

A novel algorithm for contact detection was developed and tested for body-point interactions using algorithms suitable for a modern object oriented environment.

Further improvements to the algorithm can be made by reducing the area (in 2D) of the subdomain $\mathbb{R}_{s\_CD}$ from a rectangle containing the finite element (FE) as shown in Figure 6.15a into a series of smaller rectangles as shown in Figure6.15b, thereby avoiding unnecessary search operations. This requires the modification of algorithms 6.4 and 6.5.

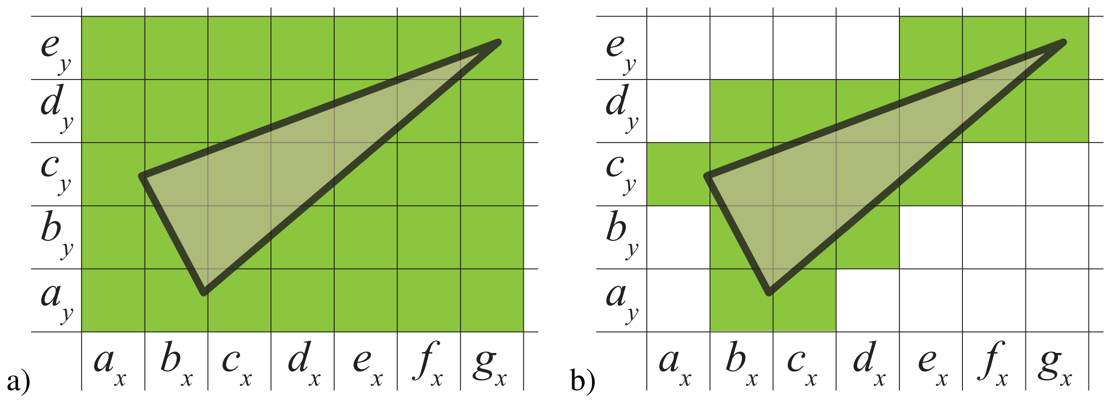

a)
e_y
d_y
c_y
b_y
a_y
a_x b_x c_x d_x e_x f_x g_x
b)
e_y
d_y
c_y
b_y
a_y
a_x b_x c_x d_x e_x f_x g_x

Figure 6.15: Schematic contact detection subdomain $\mathbb{R}_{s\_CD}$. a)Implemented in MS. b)Proposed improvement, not implemented in this work.

Chapter 7

DETAIL COMPARISONS OF MR, MR-S, NBS AND BBTS ALGORITHMS

7.1 Introduction

For developers working in DEM, FDEM, MD, etc., it is important to be able to decide which search algorithm to use in a particular situation. None of the searches in the literature are perfect and they may underperform in some situations. Thus, in this analysis, an extensive comparative study of different search algorithms MR(MunjizaRougier), MR-S(MunjizaRougier-Schiava), NBS (No Binary Search) and BBTS (Binary Balanced Tree Schiava) for bodies of similar size has been conducted.

7.2 Numerical experiments, comparison between MR, MR-S, NBS and BBTS

As mentioned before, the BBTS is a logarithmic CD algorithm, i.e. the time needed to find a matching item is given by

$$T \propto \log(K) \tag{7.1}$$

where K are the number of elementes loaded on the BBTS and the time needed to find N items is given by

$$T \propto N \log(K) \tag{7.2}$$

In the case of the MR, MR-S and NBS linear CD algorithms the time needed to find all the contacting couples increases linearly with the number of DEs. Thus, the total contact detection time is given by

$$T \propto N \tag{7.3}$$

The performance of each one of these algorithms is compared under the same conditions by six numerical experiments. These numerical experiments are explained in detail and are easy to implement and repeat. Experiments I and II are performed on a regular cluster of DEs. These are based on tests performed separately on the NBS[136, 133] and MR[148] algorithms. Experiments III, V and VI are performed on irregular clusters in three, two and one dimension. Experiment IV consist in one single cell where all the discrete elements are placed, this correspond to the extreme case where the largest DE is various order of magnitude bigger than the smallest DE.

All the discrete elements employed are spherical of diameter d to simplify the calculus of contact detection. All the experiments were carried on a DELL 4700 (Intel 2.8 GHz / 4G RAM) running on Fedora Core 4.0. To save RAM memory pointers O to objects in BBTS, MR, MR-S and NBS are replaced by an integer. The search tests are performed on ghost discrete elements with their integerised position as the only data store. There is no calculus of interaction and no calculus of deformation.

At the beginning of the experiments de positions of each DE are defined on different package shapes and package ordering. As there is no interaction new positions of the discrete elements are calculated using a random number generator. In order to make the experiments close as possible to real FDEM simulations the cumulative contact detection times are measured after 10 cycles of :

1. Perform contact detection
2. Move DEs

Example I. The discrete elements are placed at equidistant distance d forming a cuboid shape cluster as shown in Figure 7.1.

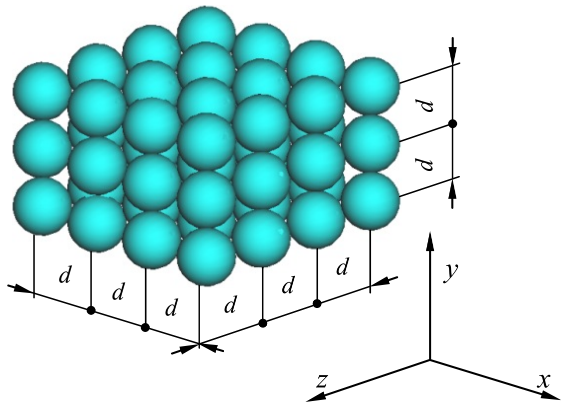


Figure 7.1: Example I cluster. Figure modified from Munjiza and Rougier[148]

The order into which the elements of the uniform raster are introduced is given by

$$\begin{array}{ccc} (i_{xmin}, i_{ymin}, i_{zmin}) & (i_{xmin}, i_{ymin}, i_{zmin}+1) & (i_{xmin}, i_{ymin}, i_{zmin}+2) \\ \ldots & (i_{xmin}, i_{ymin}, i_{zmax}) & \ldots \\ (i_{xmin}, i_{ymax}, i_{zmax}) & \ldots & (i_{xmax}, i_{ymax}, i_{zmax}) \end{array} \tag{7.4}$$

where i_{xmin}, i_{ymin}, i_{zmin} are the minimum initial x, y, z integerised coordinates and i_{xmax}, i_{ymax}, i_{zmax} are the maximum initial x, y, z integerised coordinates.

At each time step, each particle (DE) is moved randomly along the three axes. The randomness is obtained by using a random number generator routine called *Generator* based on a subroutine called rand2.[199] The generator produces a random number between 0.0 and 1.0, exclusive of the endpoint values.

$$\begin{aligned} r_x &= -i_x + iniRan \\ r_y &= -i_y + iniRan \\ r_z &= -i_z + iniRan \end{aligned} \tag{7.5}$$

where i_x, i_y, i_z are the integerised coordinates and

$$\begin{array}{lll} iniRand_t & = -1 & t = 0 \\ iniRand_t & = iniRand_{t-1} - 10 & t = 1, 2, 3, ..., 9 \end{array} \tag{7.6}$$

and t is the time step. Each random number is obtained as

$$\begin{aligned} \alpha_x &= Generator(r_x) \\ \alpha_y &= Generator(r_y) \\ \alpha_z &= Generator(r_z) \end{aligned} \tag{7.7}$$

and each new coordinate is calculated by

$$\begin{aligned} i_x &= i_x\,(int)\,(\alpha_x + \alpha_x) \\ i_y &= i_y\,(int)\,(\alpha_y + \alpha_y) \\ i_z &= i_z\,(int)\,(\alpha_z + \alpha_z) \end{aligned} \tag{7.8}$$

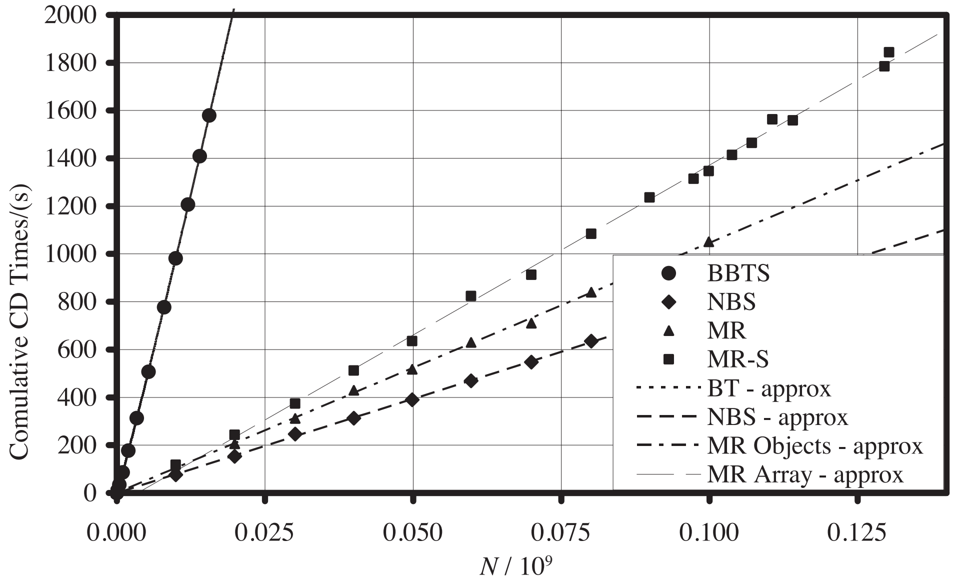


Figure 7.2: Example I: cumulative contact detection times.

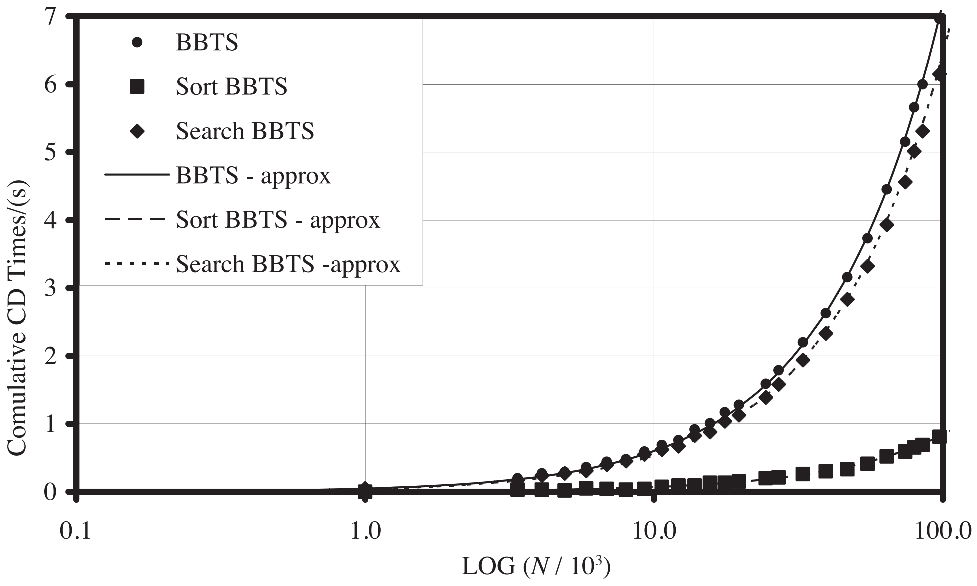


Figure 7.3: Example I: cumulative times for BBTS.

The cumulative contact detection times for the different algorithms are shown in Figure 7.2. The cumulative times of contact detection, sort and search times for the BBTS are shown in Figure 7.3.

The approximation curves were obtained by applying the least square method to the function

$$t = \alpha N \log(N) \tag{7.9}$$

Example II. A total of 1,000,000 discrete elements are placed at equidistant distance s as shown in Figure 7.4.

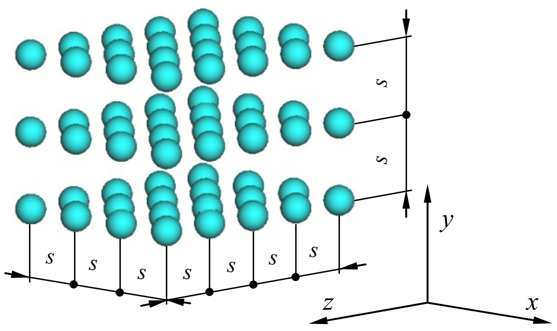


Figure 7.4: Example II cluster. Figure modified from Munjiza and Rougier[148]

The order into which the elements are introduced is given by

$$\begin{array}{cc}
(i_{xmin}, i_{ymin}, i_{zmin}) & (i_{xmin}, i_{ymin}, i_{zmin}+s) \\
(i_{xmin}, i_{ymin}, i_{zmin}+2s) & \ldots \\
(i_{xmin}, i_{ymin}, i_{zmin}+js) & \ldots \\
(i_{xmin}, i_{ymin}+js, i_{zmin}+js) & \ldots \\
\ldots & (i_{xmin}+js, i_{ymin}+js, i_{zmin}+js)
\end{array} \tag{7.10}$$

where i_{xmin}, i_{ymin}, i_{zmin} are the minimum initial x, y, z integerised coordinates and j is the number of particles per axis. This order is called xyz.

The “density changes with spacing s as”[148]

$$\rho \propto \frac{1}{s^3} \tag{7.11}$$

At each time step, each particle is moved randomly along the three axes as shown in Equation 7.8. As different values of s are chossen the packing density varies as[148]

$$1 \leq \rho \leq \frac{1}{8000000} \tag{7.12}$$

The cumulative contact detection times are shown in Figure 7.4.

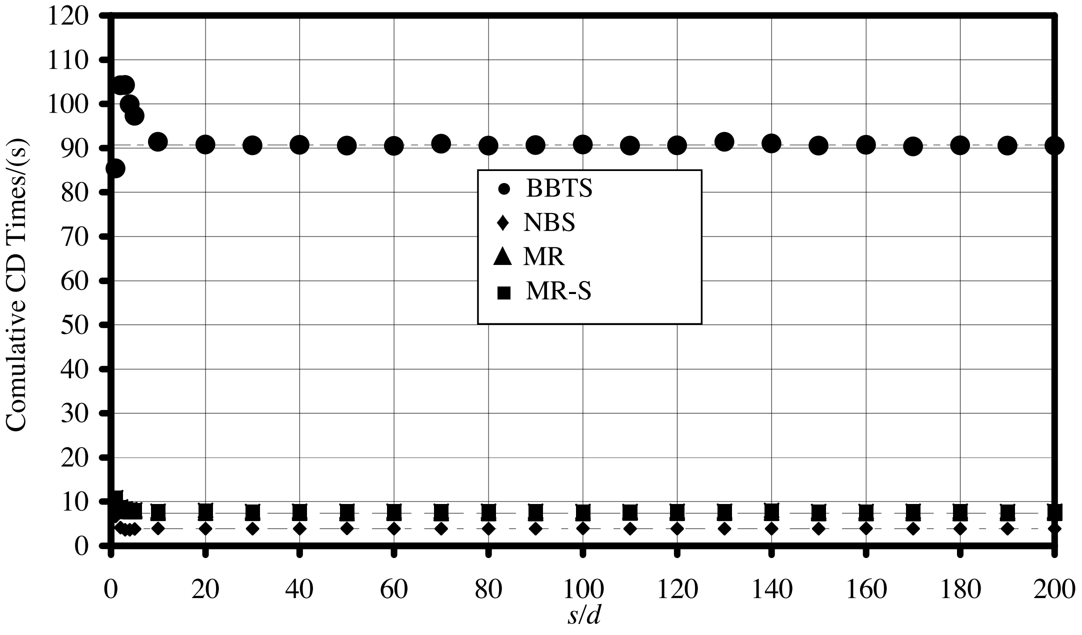


Figure 7.5: Example II: cumulative contact detection times.

A detail of the total cumulative contact detection times for the NBS, MR and MR-S algorithms are shown in Figure 7.6.

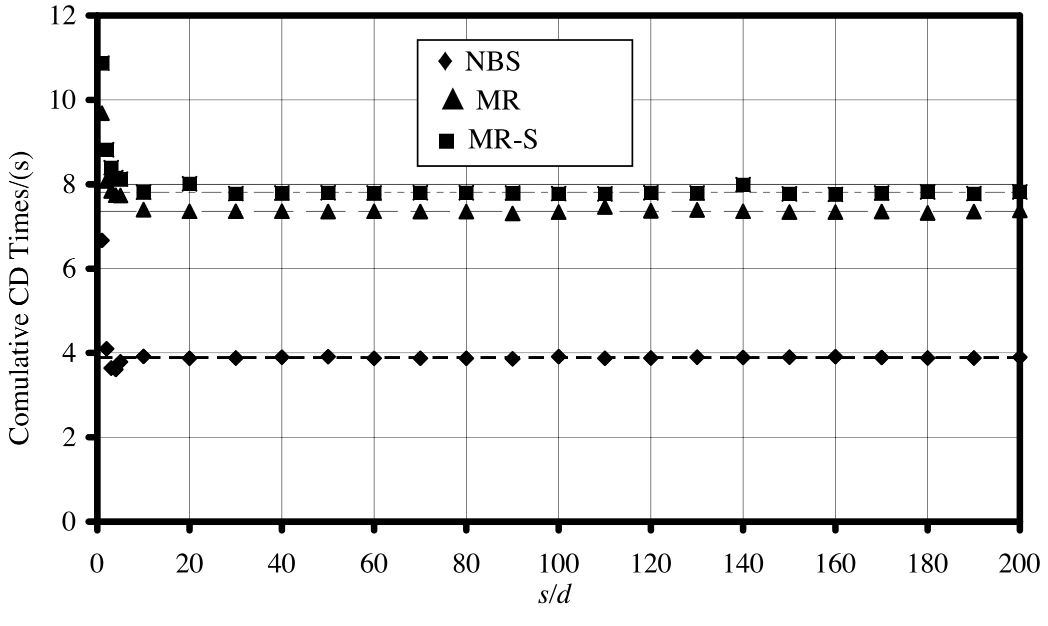


Figure 7.6: Example II: MR and NBS cumulative contact detection times.

Example III. The discrete elements are placed randomly in x, y and z directions inside a cube as shown in Figure 7.7.

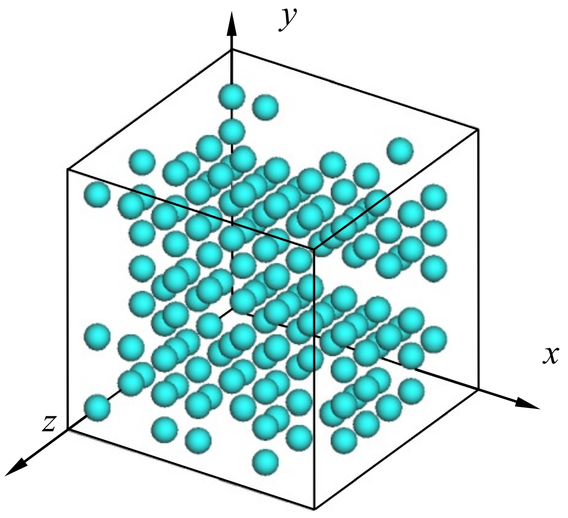


Figure 7.7: Example III cluster

The order into which the elements are introduced is also random. The randomness is obtained by using the *Generator* routine. The constants to initialize the random generator are

$$\begin{aligned} r_x &= -1 \\ r_y &= -2000 \\ r_z &= -500000 \end{aligned} \tag{7.13}$$

each coordinate is calculated as

$$\begin{aligned} i_x &= (\text{int})\,(\delta Generator\,(r_x)) + i_{xmin} \\ i_y &= (\text{int})\,(\delta Generator\,(r_y)) + i_{ymin} \\ i_z &= (\text{int})\,(\delta Generator\,(r_z)) + i_{zmin} \end{aligned} \tag{7.14}$$

where

$$\delta = js \tag{7.15}$$

and j is equal to the number of elements per axis, in the case of the uniformly packed examples. The variable s controls the packing density of the system and in this case is set to

$$s = 3 \tag{7.16}$$

The total number[133] of DEs is given by

$$N = j^3 \tag{7.17}$$

The process of calculating the random integerised coordinates for each one of the elements is illustrated in the Algorithm 7.1.

Algorithm 7.1 3D: random integerised coordinates.

```
1:  $i = 0$
2:  for $(i_1 = 0; i_1 < j; ++i_1)$ do
3:      for $(i_2 = 0; i_2 < j; ++i_2)$ do
4:          for $(i_3 = 0; i_3 < j; ++i_3)$ do
5:              $x[i] = (\text{int})(\delta Generator(r_x)) + i_{xmin}$
6:              $y[i] = (\text{int})(\delta Generator(r_y)) + i_{ymin}$
7:              $z[i] = (\text{int})(\delta Generator(r_z)) + i_{zmin}$
8:          end for
9:      end for
10: end for
```

At each time step, each particle is moved randomly along the three axes using the Equation 7.8. The cumulatives contact detection times are shown in Figure 7.8.

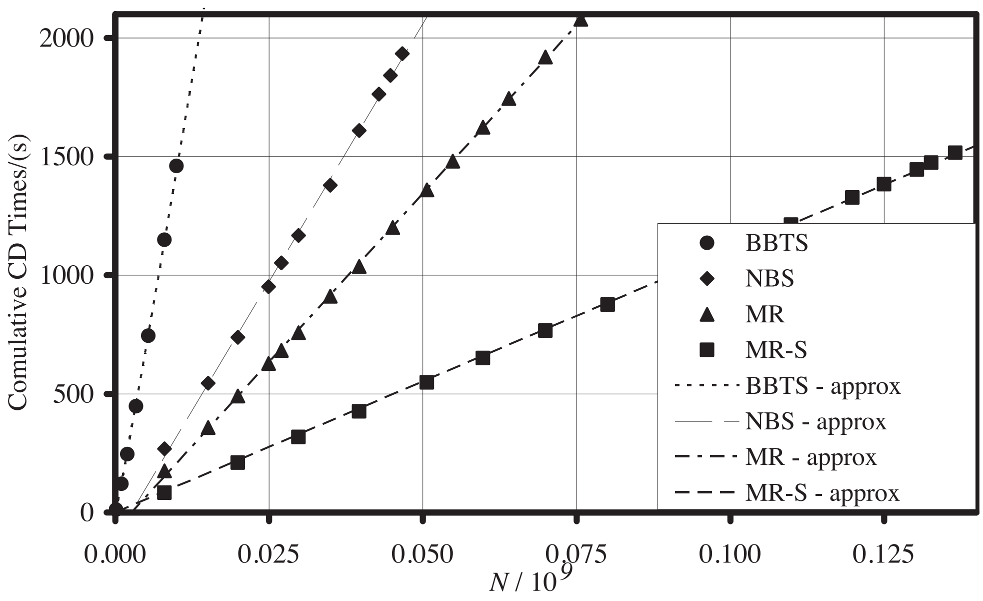


Figure 7.8: Example III: cumulative contact detection times.

In the case of the BBTS, a detail of the total cumulative times for sort, search and contact detection are shown in Figure 7.9.

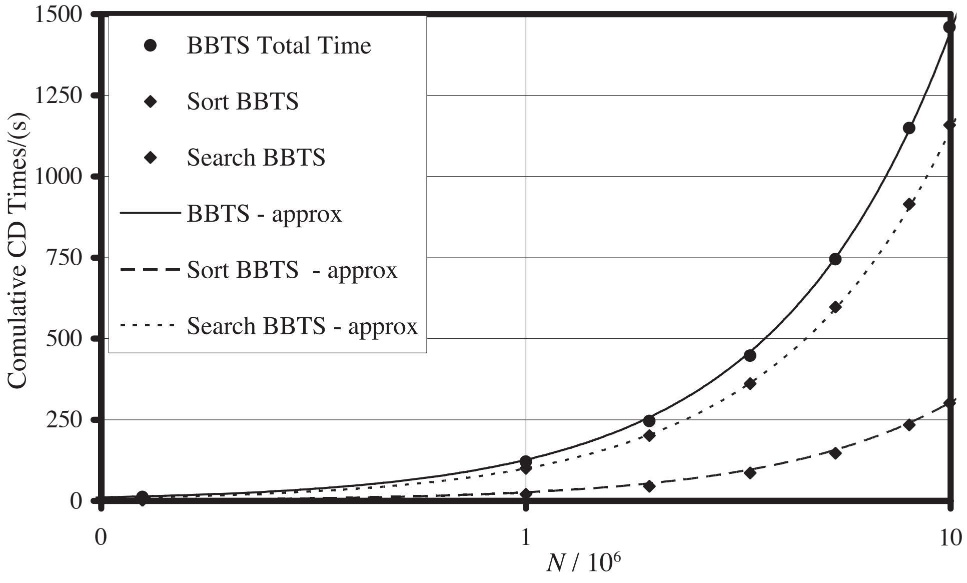


Figure 7.9: Example III: cumulative times for BBTS.

Example IV. All discrete elements are placed in the same cell. During the whole simulation the elements cannot move away from the box. Basically each algorithm solves a quadratic search.

The total cumulative times are shown in Figure 7.10. The approximation curves shown in Figure 7.10 are obtained by fitting the experimental data to the following function

$$f(N) = aN^2 + bN + c \tag{7.18}$$

The value of the coefficients a, b and c are determined through the least square method.

The behaviour for all the algorithms in terms of cumulative contact detection time is quadratic. This shows the disadvantage in contact detection time, when the size of the cell is excessively big, relative to the size of the objects. These algorithms are not designed to work with objects with large variances in their sizes.

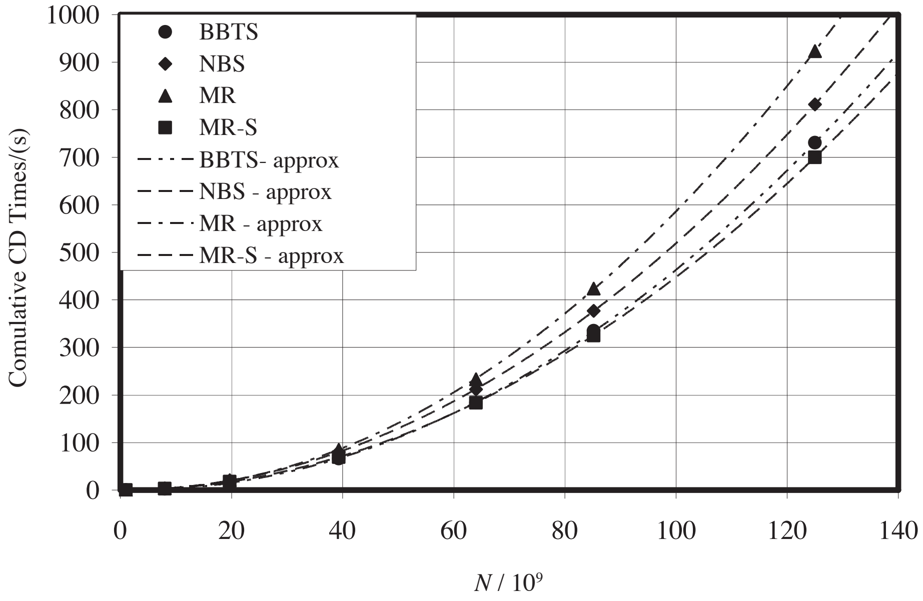


Figure 7.10: Example IV same cell: cumulative contact detection times.

Example V. All discrete elements are placed randomly in x direction and keeping y and z constants, i.e.

$$\begin{aligned} i_y &= i_{ymin} \\ i_z &= i_{zmin} \end{aligned} \tag{7.19}$$

as shown in Figure 7.11.

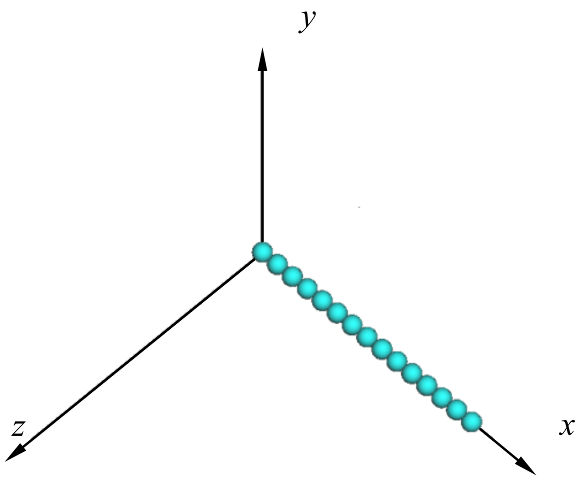


Figure 7.11: Example V cluster.

The order into which the elements are introduced is also random. The density for each value of N is constant and equal to 4.

$$\rho = \frac{N}{L} = 4 \tag{7.20}$$

The constant used initialize the random generator is

$$r_x = -1 \tag{7.21}$$

The process of calculating the random integerised coordinates for each one of the elements is illustrated in Algorithm 7.2.

Algorithm 7.2 1D: Random integerised coordinates.

```
1:  $i = 0$
2:  for $(i_1 = 0; i_1 < j; ++i_1)$ do
3:      for $(i_2 = 0; i_2 < j; ++i_2)$ do
4:          for $(i_3 = 0; i_3 < j; ++i_3)$ do
5:                      $x[i] = (\text{int})(\delta Generator(r_x)) + i_{xmin}$
6:                      $y[i] = i_{ymin}$
7:                      $z[i] = i_{zmin}$
8:              $i = i + 1$
9:          end for
10:     end for
11: end for
```

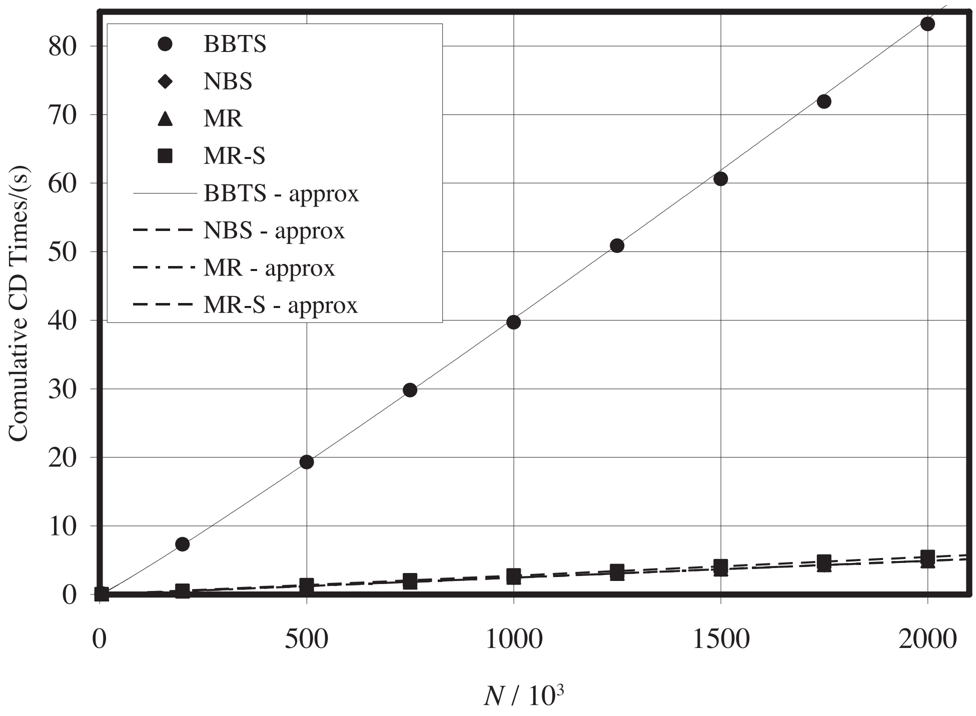


Figure 7.12: Example V (1D): cumulative contact detection times.

At each time step, each particle is only moved along the x axis. This movement is done randomly. The cumulative contact detection times are shown in Figure 7.12 and in Figure 7.13.

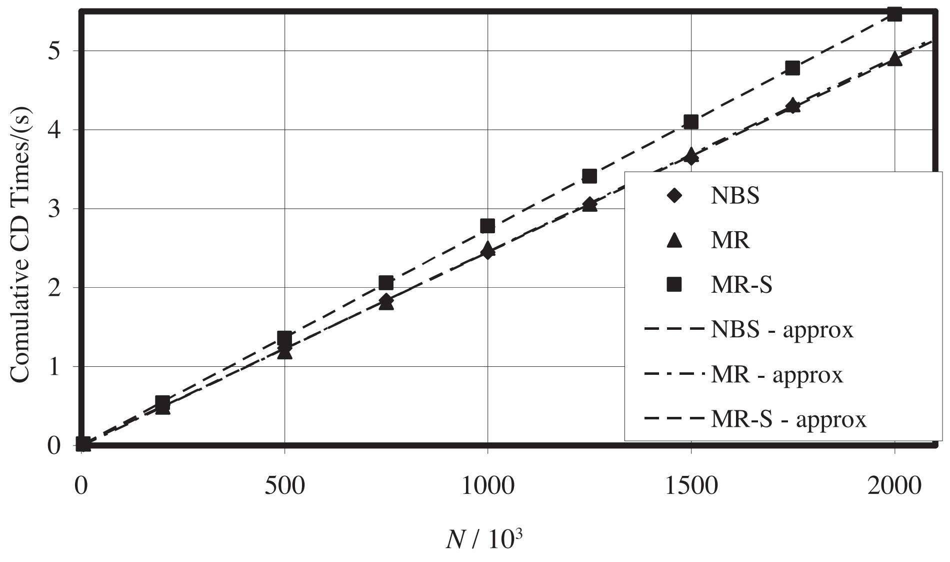


Figure 7.13: Example V (1D): cumulative contact detection times.

Example VI. All discrete elemntes are placed at randomly in x and y and $z = z_{min}$ as shown in Figure 7.14.

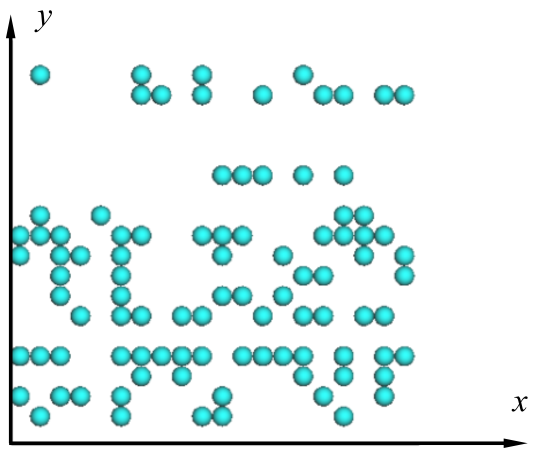


Figure 7.14: Example VI cluster.

The order into which the elements are introduced is also random. The density for each value of N is constant and equal to 0.33333.

$$\rho = \frac{N}{A} = 0.33333 \tag{7.22}$$

The constants used to initialize the random generator are

$$\begin{aligned} r_x &= -1 \\ r_y &= -1 \end{aligned} \tag{7.23}$$

The process of calculating the random integerised coordinates for each one of the elements is illustrated in Algorithm 7.3.

Algorithm 7.3 2D: Random integerised coordinates.

```
1:  $i = 0$
2:  for $(i_1 = 0; i_1 < j; ++i_1)$ do
3:      for $(i_2 = 0; i_2 < j; ++i_2)$ do
4:          for $(i_3 = 0; i_3 < j; ++i_3)$ do
5:              $x[i] = (\text{int})(\delta Generator(r_x)) + i_{xmin}$
6:              $y[i] = (\text{int})(\delta Generator(r_y)) + i_{ymin}$
7:              $z[i] = i_{zmin}$
8:              $i = i + 1$
9:          end for
10:     end for
11: end for
```

At each time step, each particle is moved randomly along the *x* and *y* axes only, i.e. during the simulation all the particles remain in the plane defined by

$$z = z_{min} \tag{7.24}$$

The cumulative contact detection times are shown in Figure 7.15 and Figure 7.16.

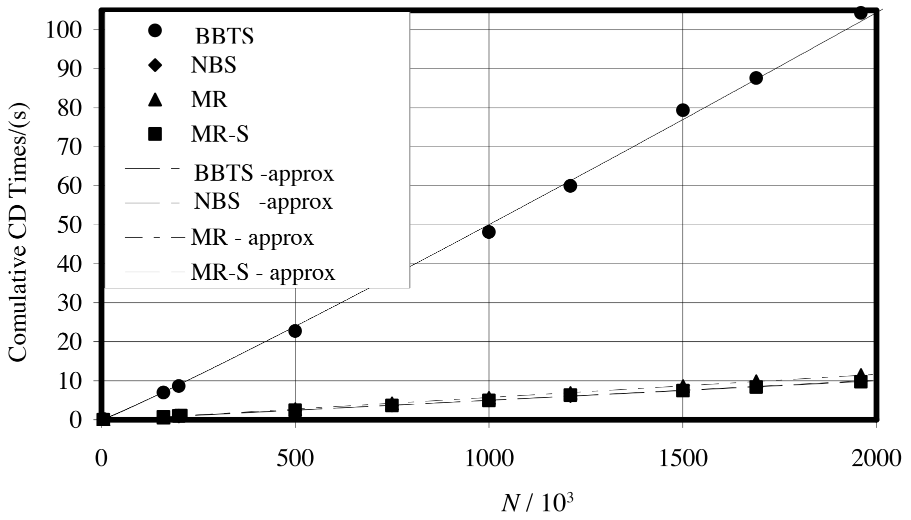


Figure 7.15: Example VI (2D): cumulative contact detection times.

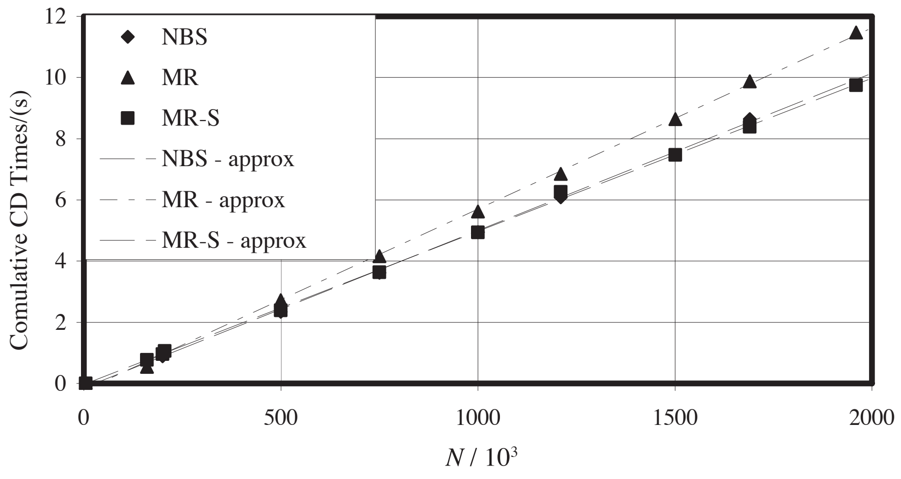


Figure 7.16: Example VI (2D): cumulative contact detection times.

Memory speed travelled test. In Example I (Figure 7.2) and in the Example II (Figure 7.5) the NBS algorithm outperforms any other algorithm. However in example III (Figure 7.8) the MR outperforms the other algorithms as expected. This difference in the speed between NBS and MR, occurs when the elements are introduced in a different order. A simple test is performed to demonstrate that the differences in time may be due to the internal use of memory. This test *cannot be taken as a general behaviour* of an x86 computer, and may be particular to the OS/Machine employed in this work.

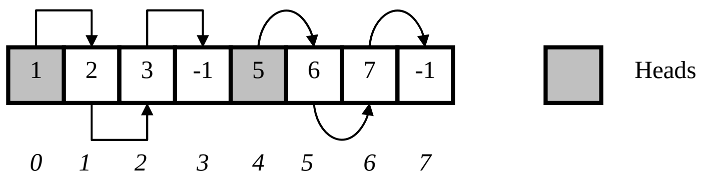


Figure 7.17: Case A.

Position Array	First Path Pointers	Second Path Pointers
0	1	
1	2	
2	3	
3	-1	
4		5
5		6
6		7
7		-1

Table 7.1: Case A.

The velocity to traverse a 1D vector with two heads is tested. In Case A the vector is ordered in such a way that each of its values is equal to the number of the next position, as shown in Figure 7.17. and in Table 7.1.

In the second test (Case B), the vector is ordered in such a way that each value of the vector is equal to the number of the next position plus 1 as shown in Figure 7.18 and in Table 7.2.

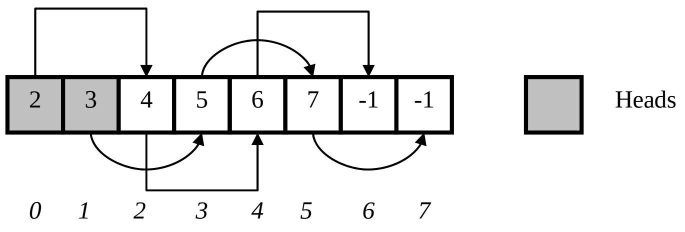


Figure 7.18: Case A.

In theory there should be no difference in the time expended to “travel” through this array. However, as is shown in Figure 7.19 and Figure 7.20, there is a notable difference in the time of around 10%.

Position Array	First Path Pointers	Second Path Pointers
0	2	
1		3
2	4	
3		5
4	6	
5		7
6	-1	
7		-1

Table 7.2: Case B.

7.3 Conclusions

Different tests were performed comprising up to 0.12 billon particles. The linear behaviour of the NBS, MR as well as the new MR-S algorithms were shown. This exhaustive experiment can be used as a tool to choose the best algorithm to be implemented in various cases.

The BBTS was the slowest algorithm. However the main objective of these simulations was to compare different algorithms in different real situations. On the other hand, for researchers who are already working with BT or who need to implement a BT for other reasons (i.e. for a data base) a simple modification to the structure of the

BT will allow them to use it as a CD-search algorithm without the difficulty associated with complex discretization of the space.

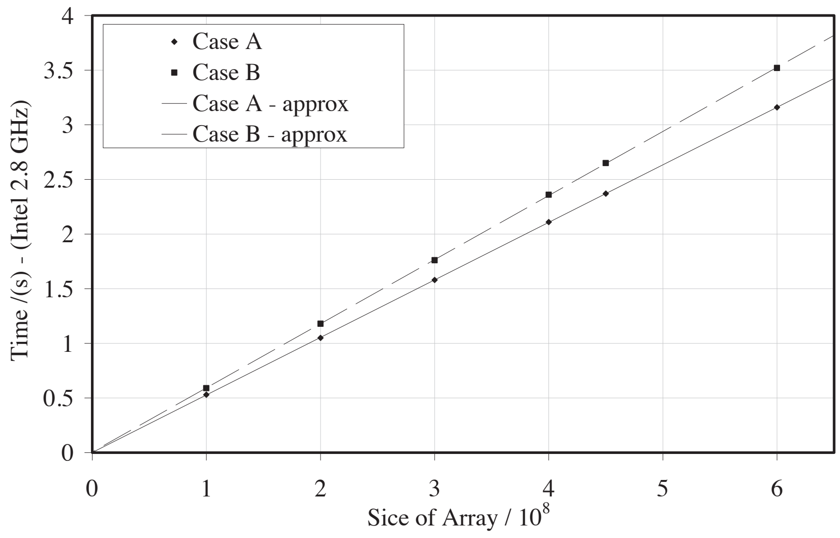


Figure 7.19: CPU time as a function of the size of the arrays.

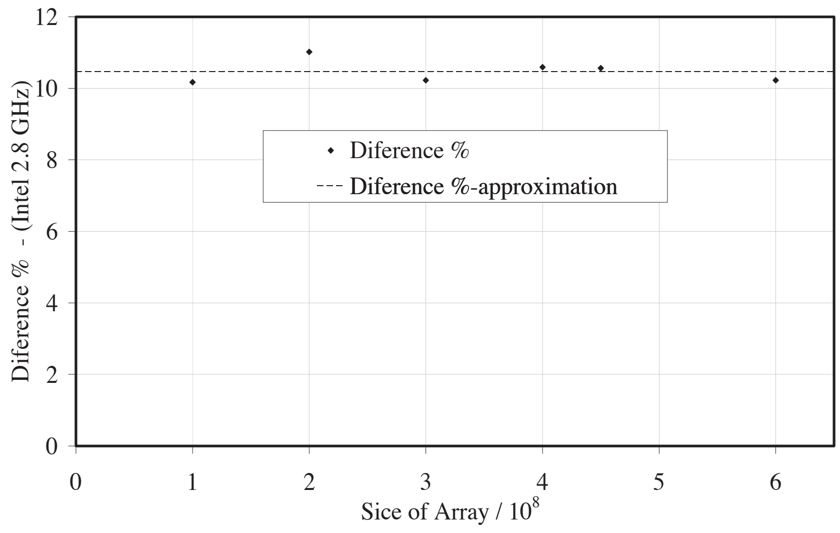


Figure 7.20: Percentage difference as a function of the size of the arrays.

It was also shown that, for vector algorithms, the way in which the elements are introduced can affect their velocity (Example I and Example III).

Example IV shows the problem encountered when a relatively large cell size is used. There is practically no difference in the search time, as all algorithms have quadratic behaviour.

In Example V and Example VI, the BBTS algorithm underperforms and the variation in CPU time for the MR, MR-S and NBS, is negligible.

In most of the Discrete Elements Simulations the particles are randomly allocated to the space (Example III). For these situations the MR and MR-S algorithms were the fastest.

If the code is written in an object oriented environment the original MR should be implemented. On the other hand if the code is written in a non-object oriented environment MR-S should be implemented.

Chapter 8

NON ELASTIC NORMAL INTERACTIONS

8.1 Non elastic interaction in 3D

The normal contact interaction between rocks, concrete blocks, etc. presents a component that is not elastic[1]. As cracks are produced, part of the energy is dissipated.[206] This dissipation during contact is different to other dissipative mechanisms such as those produced during deformation by the damping.[146] As the interaction affects the final trajectory of a discrete element, not only is the final energy is affected, but also the spatial distribution of the different entities on the domain.

For applications like coastal defence where hundreds of concrete blocks are used, with typical dimensions to an order of magnitude in metres, the use of such small discretisation to capture micro cracks is impractical. A similar situation arises in simulations of rock landslides.[9] A more suitable option is to modify the interacting algorithm making it able to produce non elastic interactions, where part of the energy is dissipated.

In this work, a model previously developed by An and Tannant[9] for the 2D DEM code PFC2D[2], and later imported into 2D-Ycode by Lisjak,[105,119] is further implemented into 3D. A modification in the behaviour of the uploading/downloading curves is implemented to avoid abrupt energy changes due to jumps in the interaction force.

The algorithm is divided into two stages depending on the current overlapping distance d_{cur} and the previous overlapping distance d_{prv}

[1]Do not confuse with frictional tangential interaction

[2]Commercial DEM software from Itasca

- Uploading curve for $d_{cur} > d_{prv}$
- Downloading curve $d_{cur} < d_{prv}$

Depending on the interacting algorithm d could be a real penetration distance d_{real} or a distance calculated using a contact potential ϕ where $d_\phi \neq d_{real}$. Even so, the model proposed in this chapter should work.

8.1.1 Uploading curve

The upload curve follows the traditional spring-like behaviour of any stress below the ceiling/top stress σ_T as shown in Figure 8.1a. When the maximum stress σ_T is reached, a further increase in the distance d does not produce changes in the contact stress σ as shown in Figure 8.1b. The equation for the stress on the upload curve is

$$\begin{cases} \sigma = \sigma_p \, d_{cur} & if \, \sigma < \sigma_T \\ \sigma = \sigma_T & if \, \sigma \geq \sigma_T \end{cases} \tag{8.1}$$

where σ_p is the penalty stress and d_{cur} is the current distance overlap. The top stress σ_T affects the amount of energy removed from the system.

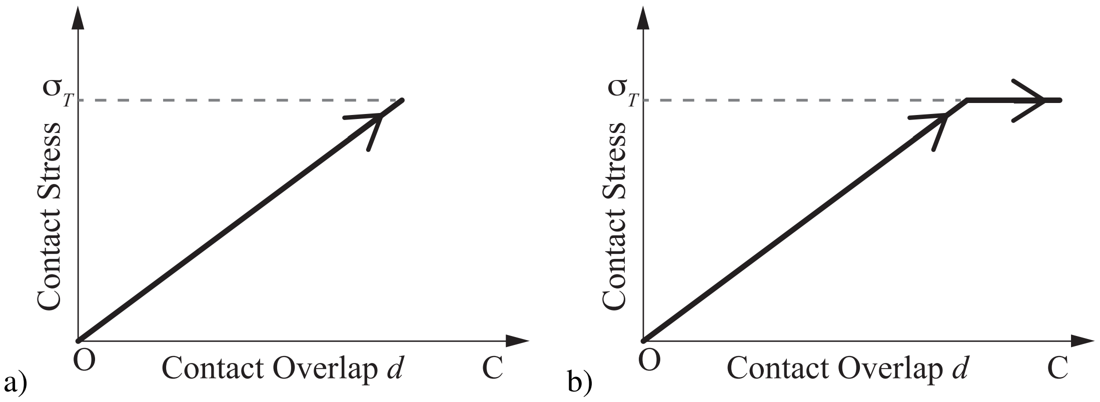


Figure 8.1: Schematic plot of contact stress. a)Stress below σ_T.b)Stress greater than σ_T. Figure modified from An et al.[9]

8.1.2 Downloading curve

An and Tannant[9] chose an exponential function to be used for the downloading curve[3], equal to

[3]Their original implementation is with forces not with stresses

$$\sigma = a\,(d_{cur})^{b} \tag{8.2}$$

where σ_p is the penalty, a is a parameter to be determined automatically by the algorithm to fit the curve, d_{cur} is the distance, and b is the exponent that regulates with σ_T the energy to be lost during interaction.

The parameter a fits the curve from the point of maximum overlapping to zero and is calculated as

$$a = \frac{\sigma_{max}}{(d_{max})^{b}} \tag{8.3}$$

where σ_{max} is the value of the stress at the maximum overlap distance d_{max}. If the exponent $b = 1$ and the stress is lower than σ_T there will be no loss in energy as all the kinetic energy Ke is returned as shown in Figure 8.2a where the loading and downloading path coincide. On the contrary if the limit of σ_T is passed the path does not coincide and there will be a loss of energy as shown in Figure 8.2b. The loss in the amount of kinetic energy is proportional to the area highlighted in grey and the kinetic energy returned is proportional to the area highlighted in blue.

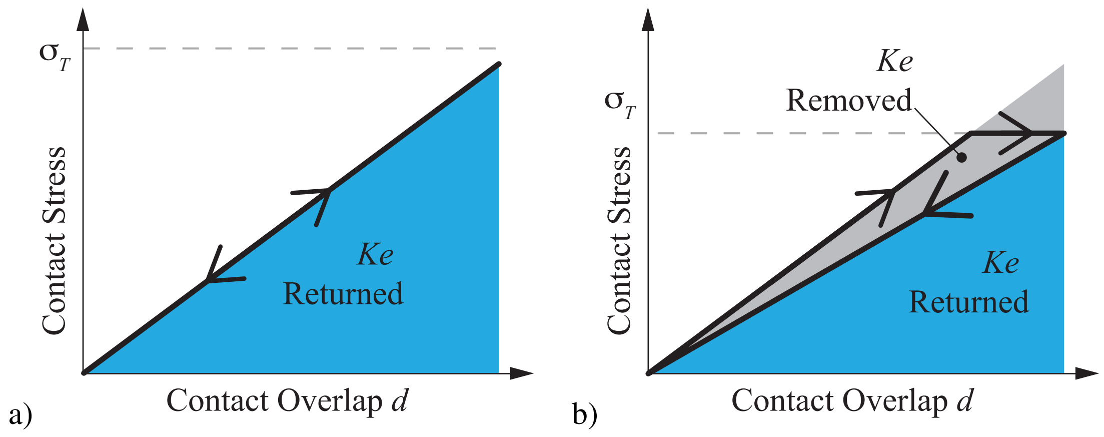


Figure 8.2: Schematic plot of contact stress. a)Elastic contact $b = 1.0$ and $\sigma < \sigma_T$. b)Inelastic contact $b = 1$ and σ_T is reached. Figure modified from An et al.[9]

Any value of $b > 1$ will produce an inelastic contact regardless of the stress σ_T as shown in Figure 8.3a and Figure 8.3b

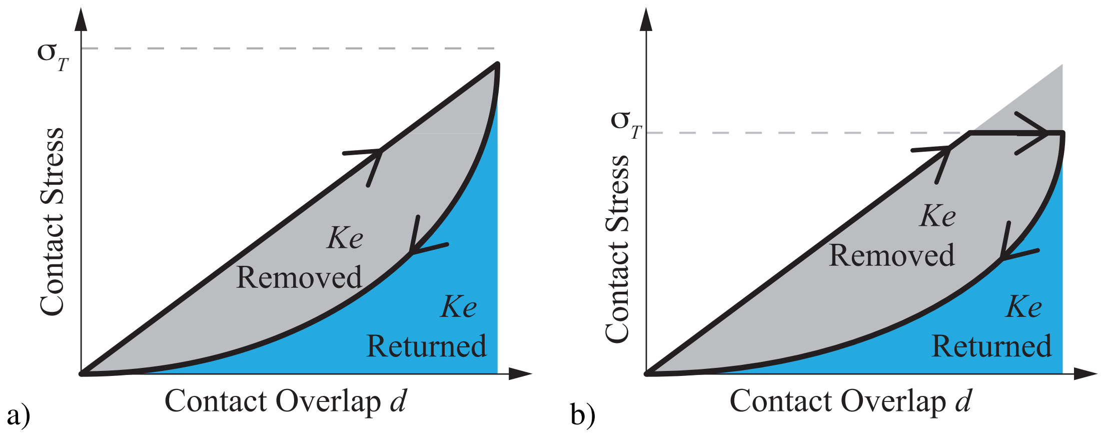


Figure 8.3: Schematic plot of contact stress. Inelastic contact. a)$\sigma_T > \sigma_D$ and $b > 1.0$. b)$\sigma_T < \sigma_D$ and $b > 1.0$. Figure modified from An et al.[9]

8.2 Modification on the non-elastic model

In simulations involving more than 2 discrete elements it is possible to have complex contact interactions, in which the contact distance during a rebound does not decrease but increases. Neither An and Tannant[9] nor Lisjak[105, 119] explicitly explain the behaviour of the model for complex interactions. They mention only two curves loading and downloading.

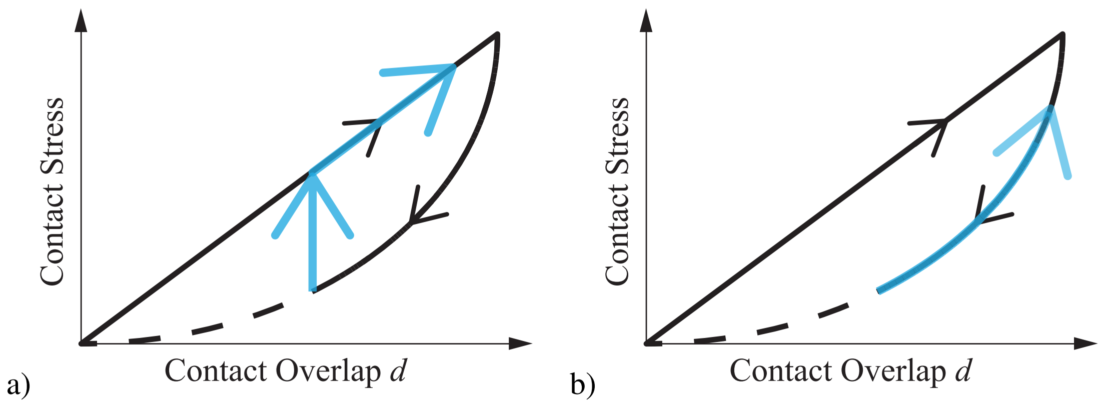


Figure 8.4: Schematic plot of contact stress in case of overlapping distance increase after decrease. a)Increasing contact stress on the uploading curve. b)Increasing contact stress on the downloading curve.

There are two possibilities

1. Go back to the uploading curve as shown in Figure 8.4a.

2. Go back on the downloading curve shown in Figure 8.4b.

If the first option is applied it will produce an increase in kinetic energy as there is an abrupt change in the interaction force. In this work the second path is chosen to avoid that increase.

8.3 Test

The inelastic normal interaction used in this work is tested in three different cases. For the first two cases the position of all the discrete elements is imposed to reproduce the three types of contact described in this chapter:

- Case A: Elastic.
- Case B: Inelastic stresses below σ_T.
- Case C: Inelastic stresses reaching σ_T.

while in the last test a concrete coastal defence block is dropped with initial velocity of 0.5 $^{\mathrm{m}}/_{\mathrm{s}}$.

8.3.1 Test A

This test consists of two discrete elements shown in Figure 8.5. The left tetrahedra is fixed, while the position of the moving right tetrahedra function of time is shown in Figure 8.6 reproducing a simple interaction. The $A = 173.2\text{mm}^2$ with penalty $\sigma_p = 90$ MPa.

The interaction forces are shown in Figure 8.7 , where the curves reproduce the theoretical behaviour.

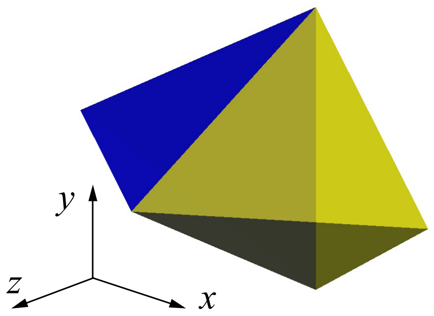


Figure 8.5: Simple discrete elements: the left tetrahedra is fixed, while the red tetrahedra moves.

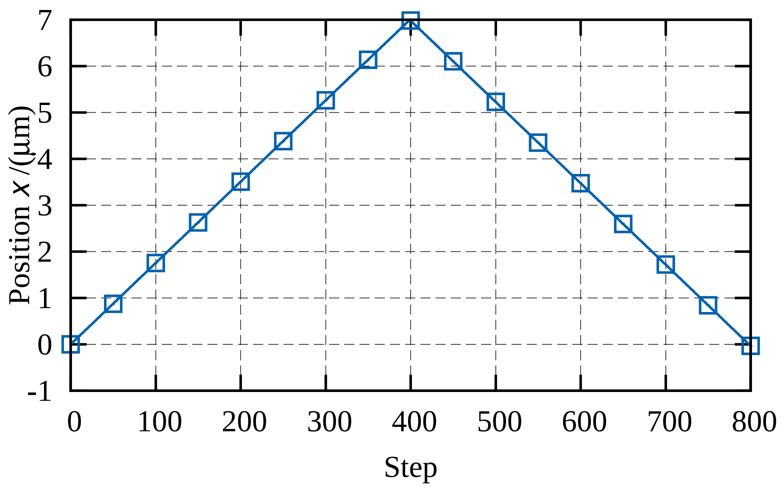


Figure 8.6: Imposed position on right tetrahedra.

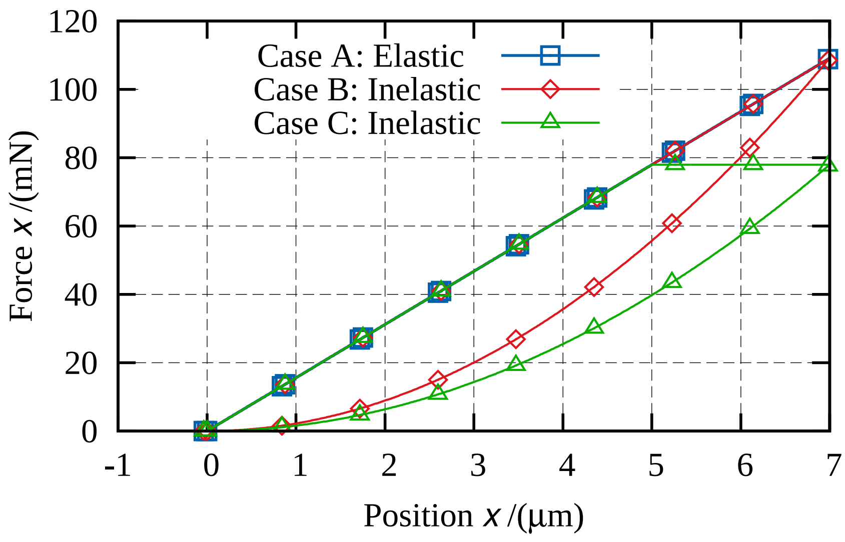


Figure 8.7: Interaction between two discrete elements. Case A: σ_T=60MPa, $b = 1$. Case B: σ_T=60MPa, b =2. Case C: $\sigma_T = 450$ Pa, b =2.

8.3.2 Test B

This test is similar to the previous test A. The only difference is the imposed position on the right tetrahedra shown in Figure 8.8. Contact forces are shown in Figure 8.9 with no fluctuations in the interaction stresses as expected after the modification previously discussed.

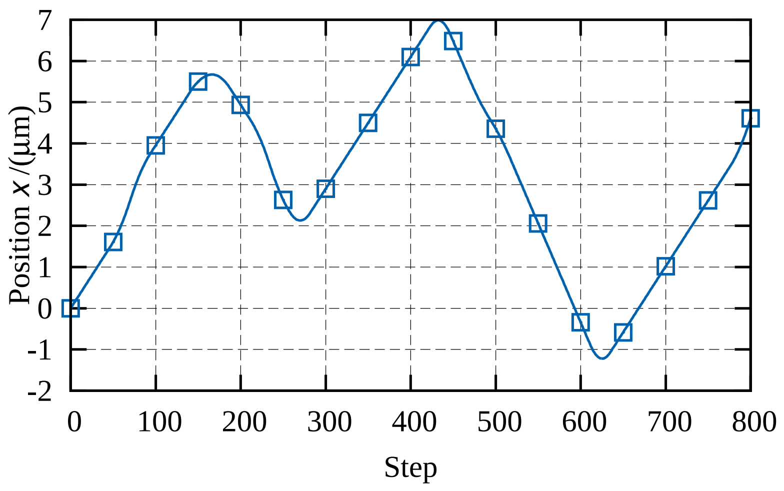


Figure 8.8: Imposed positions on right tetrahedra.

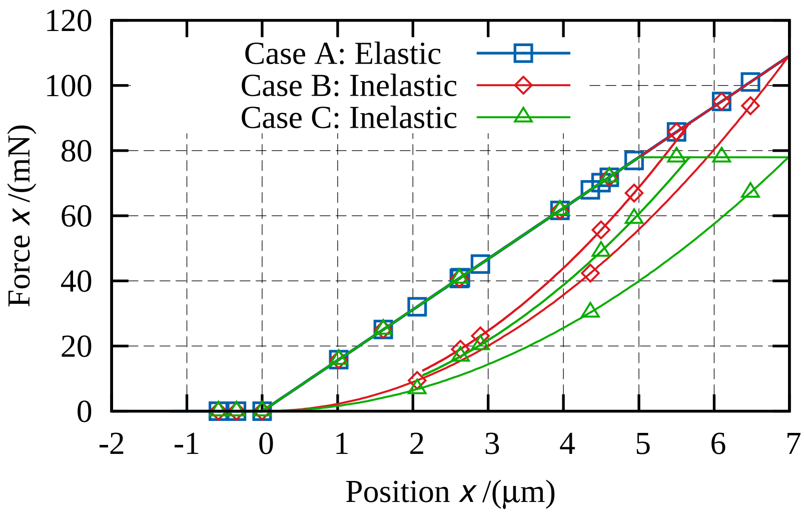


Figure 8.9: Interaction between two discrete elements. Case A: σ_T=60MPa, $b = 1$. Case B: σ_T=60MPa, b =2. Case C: $\sigma_T = 450$ Pa, b =2.

8.3.3 Test C

This test simulates the drop of a coastal defence block, different values of σ_T and b are tested. The dimensions[206] of the coastal block are shown in Figure 8.10a.

The material of the block is concrete with elastic modulus E=26.6 MPa, Poisson's ratio ν =0.205, density ρ =2340$^{\text{kg}}/_{\text{m}^3}$, while the damping is set to zero. The only dissipation on the system is the one produced during contact interaction. The discrete elements used to simulate the floor have an imposed velocity equal to zero during the whole simulation, while the defence block has an initial velocity equal to 0.5 $^{\text{m}}/_{\text{s}}$.

The kinetic energy of the block function of time is shown in Figure 8.11 where for σ_T=100 MPa and b =1 there is no loss of kinetic energy. For all the other cases the amount of kinetic energy removed increases as σ_T decreases and b increases.

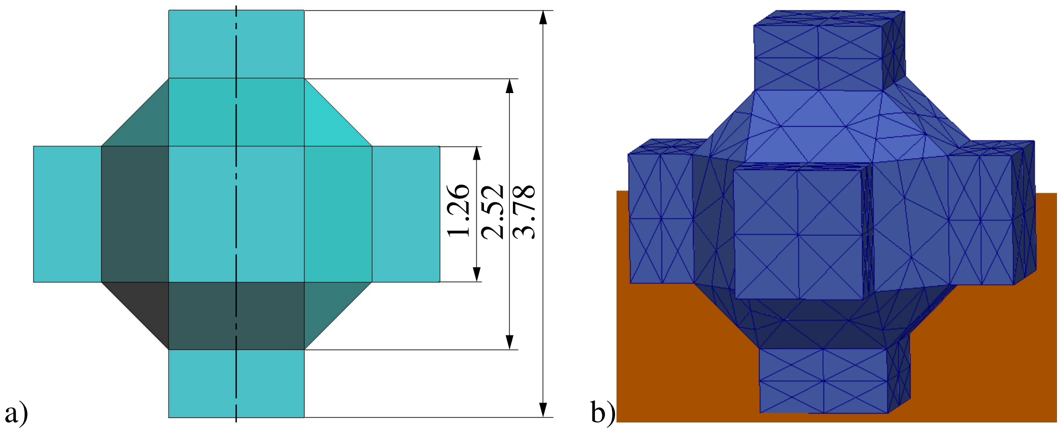


Figure 8.10: Concrete coastal defence. a)Dimensions (m). b)Mesh.

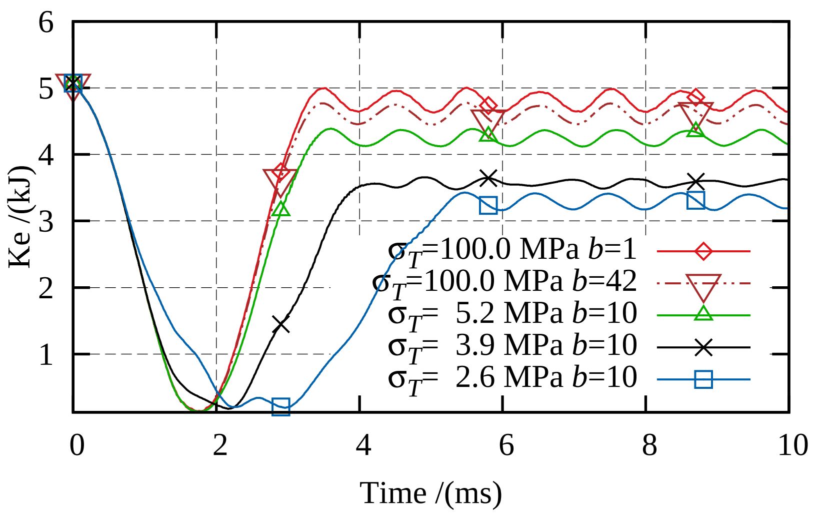


Figure 8.11: Kinetic energy of the coastal block.

Chapter 9

NOVEL PARALLEL SOLUTIONS FOR THE COMBINED FINITE-DISCRETE ELEMENT METHOD

In this chapter a novel parallel solution for the FDEM is presented. This solution is built upon previous developments in this work: binary tree (Chapters 4 and 6), contact detection (Chapter 6) and non-elastic interaction (Chapter 8).

In Chapter 3 an extensive description of different hardware and library/software were presented. There is no simple answer to the question of which is the best combination of hardware/library/programming language.
Ideally, it should be possible to develop a new sequential solution and leave the hard parallelization work to the compiler. Automatic parallel[79] compilers which take a sequential code and transform it into a parallel code are not yet developed enough to be used on CPU demanding applications such as those created via FDEM.

Many libraries are available for usage in parallelization of code, including OpenMP, CUDA, OpenCL, MPI, etc. The Message Passing Interface (MPI) is quite wide spread and supported by many companies and universities. It must be noted that MPI is "not a language, it is a specification of a library of routines that can be called from programs".[42]

It is possible to use MPI from Python, Fortran, C, C++, C#, among other languages. It can be used on multicore and cluster computers. In this work, an explicit parallelization using MPI and the C++ language is the chosen path.

A code that relies heavily on a particular library (such as MPI) and uses most, if not all, of its available resources has the advantage of some of the "tailor made" speed up functions. Libraries change with time, and their development is sometimes discontinued. In either case, changes in external libraries incur great programming efforts to adapt to a new library that fulfils the characteristics of the old one. Therefore in this work, the functions used from the MPI package are kept at a minimum.

The chapter is divided into 3 sections. Section 9.1 introduces important concepts and operations used in MPI. Section 9.2 explains the algorithm developed in this work and finally the section 9.3 presents different test cases.

9.1 Message Passing Interface

In general, it is possible to divide computer memory into two main groups[160]

- Shared Memory: nowadays common multicore computers. Each processor can access all the memory available.
- Distributed Memory: HPC (High Performance Computing) each processor can only access their own local memory.

OpenMP[25, 168] provides coding tools for shared memory systems. The advantage of OpenMP is the speed up in the communication as there is no need to "move" memory from one processor to another. All processors can "see" all the memory. Message Passing[61] on the other hand, produces an exchange of messages between processors, moving data across the network. The message is copied from the sender to a buffer on the receiver.[42]
When MPI is used on a multicore computer, there is no memory access between processors, thus isolating[1] the operations in each processor. The advantage in this case lies in the *firewall* imposed on the memory,[61] facilitating the debugging process. At the same time, there is a price to pay in CPU time, since memory has to be passed around.

It is important to note that some concepts of MPI can be easily misinterpreted. MPI operates on a per-process basis and, for a given implementation, more than one process may reside in the same processor[61] i.e. it is possible to run two processes on a single core PC. This means that

$$n_{prcocessors} \geq n_{processes} \tag{9.1}$$

[1]MPI-2 allows the access of memory from processor to processor explicitly[61] thus copying from or acceding the memory of the other processor. Still this could be seen as a message delivered straight to/from the other processor memory. Is not sharing memory as is the case of OpenMP.

where $n_{processors}$ is the number of processors and $n_{processes}$ is the number of processes, is a condition that is not always satisfied. Unless a parallel program has a strong nonlinear[2] dependency with the size of the problem, in terms of CPU time, having more processes than processors will always reduce the parallel efficiency.

The complete description of MPI and its usage is out of the scope of this thesis. The books by Pacheco,[159,160] Gropp et, al[61] and by the Message Passing Interface Forum,[50] provide an in-depth description of MPI. In the rest of the section some key aspects of MPI are explained.

9.1.1 Topology

There are two classes of topology in parallel computing. One is built with processors and switches as shown in Figure 9.1a and Figure 9.1b. This is known as hardware topology.

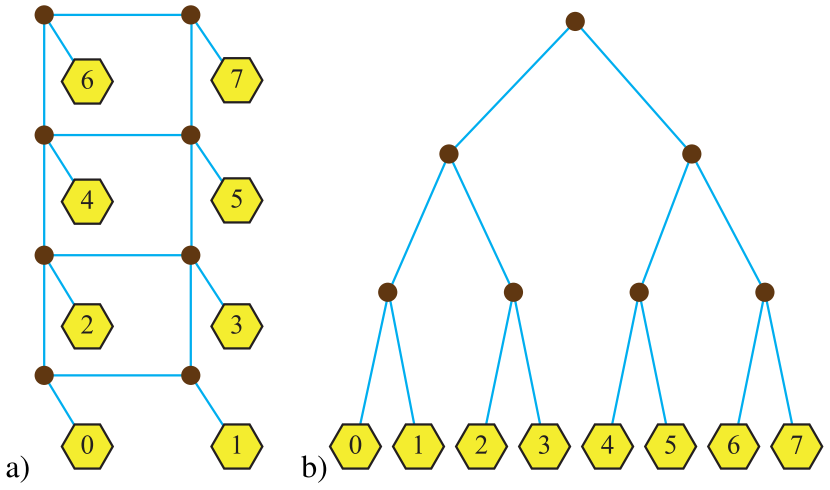


Figure 9.1: Hardware topology (network) processors are highlighted in yellow and switches are highlighted in brown. a)Simple 2D network b)Binary tree network. Figure modified from Quinn and Michael[168]

The second type is process topology, which is shown in Figure 9.2 where arrows signal communications between processes. Process topology is independent of the topology of the processors (hardware), hence, for instance, MPI is normally unaware of nearby processors in terms of communication velocity. Thus, the same code running the same problem on different HPC clusters will normally perform differently.

Process topology is built between processes sender-to-receiver, each one addressed by a unique *id* called "rank", assigned automatically by the implementation of MPI.

[2]i.e. Quadratic search.

Depending on the problem to be solved, different process topologies can be arranged. MPI gives total freedom to link processes in any shape (square, cubical, etc) regardless of the hardware topology. The process topology in general is kept fixed during the simulation, however there is no restriction from MPI on this and it could be changed (within the same MPI_COMM_WORLD) during running time.

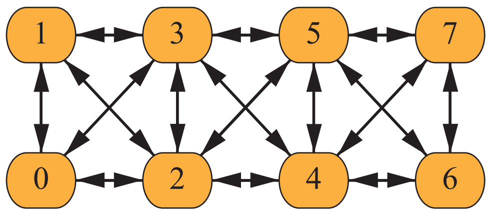


Figure 9.2: MPI 2D processes topology. Integer number correspond to the rank of each process.

9.1.2 Moving data

Buffer use in send/receive. One important concept is that of the buffer. MPI sends and receives a buffer of data during a communication between different processes. Buffers are made of different kinds of data such as integer, double, char, amongst other types, and the simplest buffer is made of a continuous agglomeration of data in the RAM memory, divided into n_{buf} objects of size $sizeType$. An example is shown in Figure 9.3.

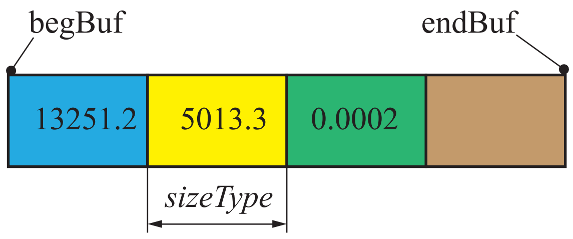


Figure 9.3: Buffer with n_{buf} =4, total size =4$sizeType$. Only the first 3 elements of the buffer have data.

There is no imposition in MPI to send a complete buffer. The process that receives the message must have its own buffer (where the data is going to be copied) big enough to accommodate the biggest possible message.

To send these simple buffers only the memory address at the beginning of the buffer and the amount of data to send are needed. For the example as shown in Figure 9.3 the address=&$begBuf$[3] and the amount of data to send =3.

[3]**&** in C/C++ gets the memory address of an entity.

Objects in C++ are “made” of different kinds of data, such as double numbers, integers, string of chars, etc. When an object has to migrate from one process to another, or when it needs to copy itself as a proxy to a particular process, it may be necessary to send all or part of its data. An example of this is shown in Figure 9.4 where 3 different data types are on non-continuous positions on the memory. Pacheco[159] presents two options to deal with this issue:

- Send a message for each data type.
- Implement one of the MPI derived type constructors.

In any case MPI needs a way to know the address of each data type (char, double, etc.) to be sent, their type, and their quantity.

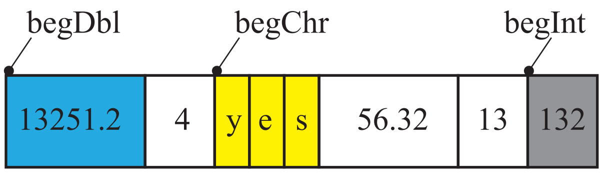


Figure 9.4: Different data types to be sent are highlighted in blue, yellow and grey. Figure modified from Pacheco.[159]

MPI_Bcast. Broadcast is an operation that sends a message from the process rank as a root to all other processes. The root process is defined by the user and can be any integer number from 0 to $n_{processes}$-1. A common practice is to define process zero as root. Usually processor zero reads the input file and distributes data to all remaining processes.

MPI_Send, MPI_Recv, MPI_Sendrecv. These are operations between two different processes. For a process i to send a buffer to a process j both processes have to reach their functions send/receive at the same time. Otherwise one of them will wait until the other is available to establish the communication. Not only is information exchanged at this point but the program is synchronised between processes.

Usually, the sender process i needs to obtain information from the receiver process j. It makes sense then to send and receive at the same time, only establishing the communication between the involved processes once.[112] In MPI this is done using MPI_Sendrecv.

To synchronize processes there is no need to send or receive a buffer. MPI has a communicator function called MPI_Barrier that pauses the execution of the program in a process and waits for others to reach the same point.

Asynchronous messages. Moving data from one processor to another is computationally and relatively expensive. The delay produced is called latency and is hardware dependent. It is possible to send a message and, while the message is being sent, allow for the continuation of other calculations without waiting for the receiver to receive it. This is called “asynchronous communications”.[41]

Operation on data been moved. MPI gives the flexibility of operating on data been copy between processes, however the order in which the data is operated on is not specified. Only operations that are order independent can be realized i.e. sum, max, min, etc. using function MPI_Reduce.[61,118] An example is shown in Figure 9.5, where the values of different processes are added and the result is collected on process zero.

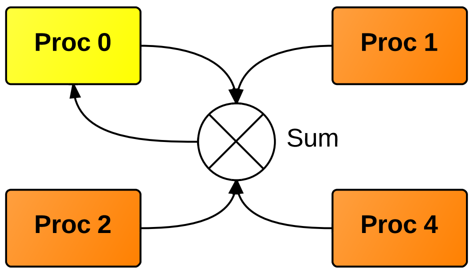


Figure 9.5: MPI_REDUCE add. Figure modified from Madron and Remigton[118]

9.2 Parallel algorithm

The FDEM method presents a higher level of complexity than other DEM methods. Contact forces have to be distributed between all finite elements (FEs) sharing a node.[133] Joint forces and failures (fractures) extending between neighbouring processes impose extra communications when compared with simple DEM parallelization.

On the other hand, the equations to be solved are localised. There is no global matrix to be shared across processes as is the case for a typical FEM parallelization.

After point, surface and volume forces have been exchanged the equation of motion Equation 3.3 (reproduced here for clarity reasons) is solved for each FE independently as

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{C}\dot{\mathbf{u}} + \mathbf{F}_{int} - \mathbf{F}_{ext} - \mathbf{F}_{con} - \mathbf{F}_{jnt} = \mathbf{0} \tag{9.2}$$

where $\ddot{\mathbf{u}}$ is the acceleration, $\dot{\mathbf{u}}$ is the velocity, $\mathbf{M}$ is the mass matrix, $\mathbf{C}$ is the damping matrix, $\mathbf{F}_{int}$ is the internal force vector, $\mathbf{F}_{ext}$ is the external force vector, $\mathbf{F}_{con}$ is the contact (interaction) force vector and $\mathbf{F}_{jnt}$ is the joint force vector.

In this section the solution methodology employed in this work is explained, starting from domain decomposition and topology, followed by the communication function for send/receive messaging. Finally the algorithm for parallel simulations for FDEM is presented.

9.2.1 Domain decomposition and topology

Domain decomposition is based on the spatial distribution of entities on the geometric-physical domain.[112] The domain is divided into n volumes, where each is assigned a processor. In this work, it should be noted that only one process is assigned to each processor.

In a sequential code there is only one process with one domain and all entities are able to interact between each other as shown in Figure 9.6a. If the same problem is solved in parallel using two processes then the domain is divided as shown in Figure 9.6b where the discrete element D lies in process zero, C lies in process one, and the discrete elements A and B are in the buffer zone between the processes. Each process can only act on entities that reside on it, meaning that some entities may need to reside in more than one process at the same time, as is the case for entities A and B shown in Figure 9.6c and Figure 9.6d.

Entities reside on a process in one of these two categories:

- as originals: all the data of the entity is stored here.
- as proxies: copies of the original, with all or part of the original's data.

An entity can have none, one or more than one proxy, but it can only have one original. Originals and proxies need to exchange data (positions, forces, flags) during the simulation. Originals act as a master, while the proxies act as slaves, meaning that any original entity needs to know where all its proxies reside i.e. the rank of the processes. Furthermore, each proxy needs to know the rank of the original process. This dependency between originals and proxies defines the processes topology.

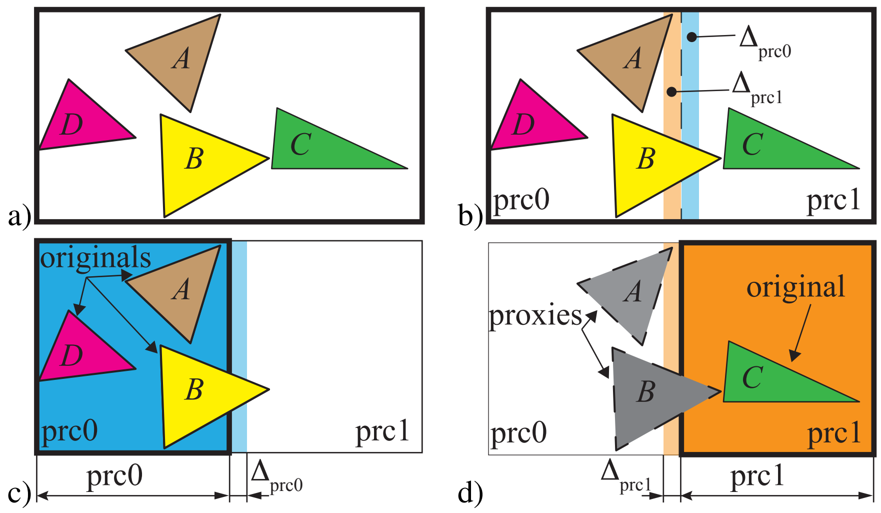


Figure 9.6: Original and proxies a)Sequential simulation, all discrete elements are original. b)Buffer zones of processes zero and one. c)Process zero domain (blue) and its buffer zone (light blue). All DEs are original d)Process one domain (orange) and its buffer zone (light orange). Discrete element *C* is original while DEs *A* and *B* are proxies (copies).

To simplify the calculus of originals and proxies and to minimise the amount of communications between processes, the shape of the volumes is limited to rectangular cuboids as shown in Figure 9.7a, while their size is related to the size of the biggest FE.

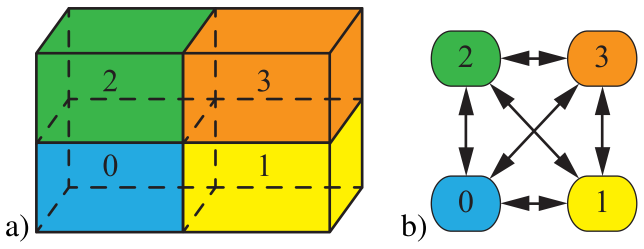
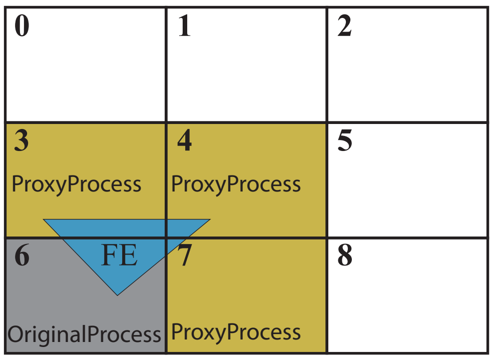


Figure 9.7: Domain decomposition. a)Volumes. b)Processes topology.

If the minimum edge of the smallest domain is greater than the maximum edge of the biggest FE then each domain needs only to communicate with its closest neighbours, as shown in Figure 9.8a. It must be considered that no FE cannot have proxies in any other process except the neighbors' processes where the original FE resides. Otherwise each process will need to communicate with an indeterminate amount of processes as shown in Figure 9.8b, where there is a proxy in a non-boundary process.

a)


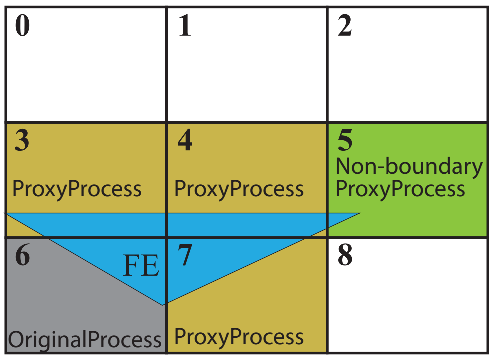
b)


Figure 9.8: Original and proxies processes for the blue highlighted FE. a)The original rank process is 6, while the proxies processes are 3, 4 and 7. b)The original rank process is 6, and the proxies processes are 3, 4, 7 and 5.

Algorithm 9.1 Contact detection between processes. The function *MkComCouples* is based on the communication engine developed by Munjiza et al.[141]

```
1:  for $(iLv\_i = 0, iLv\_i < n_{processes}, ++iLv\_i)$ do
2:      $oPrc_i = GetProcessor(iLv\_i)$
3:      $oPrc_i \rightarrow SetComArrayEmpty()$        ▷ Set all components to -1
4:  end for
5:  for $(iLv\_i = 0, iLv\_i < n_{processes}, ++iLv\_i)$ do
6:      $oPrc_i = GetProcessor(iLv\_i)$
7:      $iLv\_jIni = iLv\_i + 1$
8:      for $(iLv\_j = iLv\_jIni, iLv\_j < n_{proceses}, ++iLv\_j)$ do
9:          $oPrc_j = GetProcessor(iLv\_j)$
10:         if $(oPrc_i \rightarrow IsInContac(Pro_j))$ then
11:             $MkComCouples(oPrc_i, oPrc_j)$
12:         end if
13:     end for
14: end for
```

In general, each process has to communicate with more than one process as shown in Figure 9.7b. This could lead to a communication bottle neck as only one communication can be established at any one time. This would force the other processes to wait idly until they are allowed to communicate. The communication has to be organised into communication couples, where any process is allowed to communicate either with just one process at a time, or with none at all.

The communication couples are set automatically by performing a simple contact detection between all *oPrc* objects as shown in the Algorithm 9.1. The object *oPrc* stores the geometric properties and the communication couples of each domain.

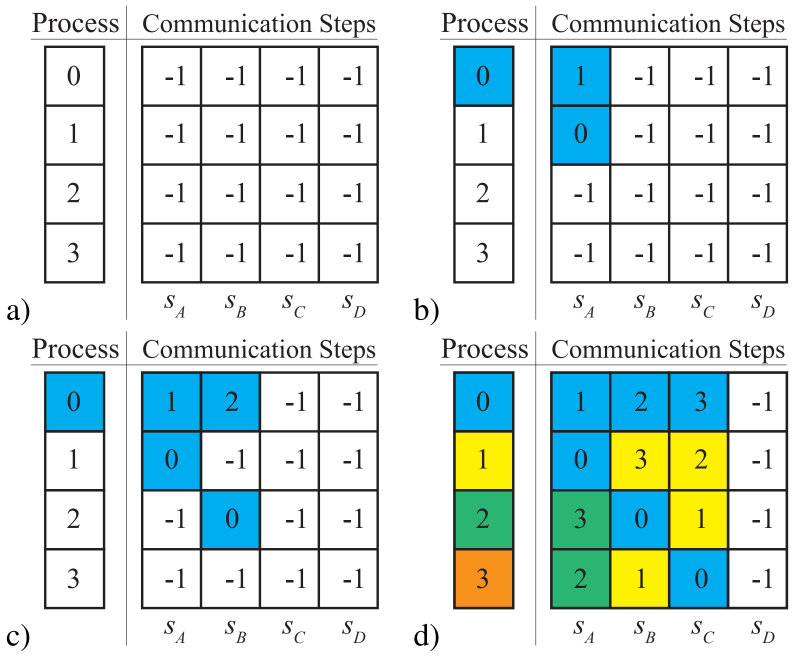


Figure 9.9: Create communications couples. a)All communications couples are empty indicated by -1. b)Set communication between process zero and one at step s_A b)Set communication between process zero and process two at step s_B. c)Final distribution of communications

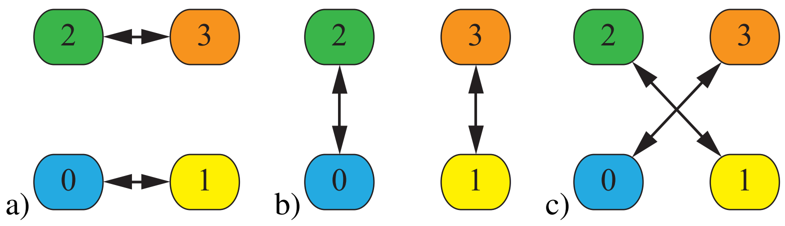


Figure 9.10: Communications between processes, split into three steps. a)Step s_A. b)Step s_B. c)Step s_C.

The above mentioned algorithm is better explained using the domain decomposition shown in Figure9.7a where all processes are in contact with each other. At the beginning of the algorithm all communications are set to null (-1) as shown in Figure 9.9a. As the “interaction” between processes is detected by performing a quadratic contact detection search, the communication couple between a process i and a process j will be created on the first communication step where both processes are available for communication i.e. where both have their communication flags equal to -1.

The first interaction detected is between process zero and process one. The communication couple is set on step s_A where both are free to communicate as shown in Figure 9.9b. Then, once the interaction between process zero and process two is detected, the algorithm will try to find a communication step where both processes have their communication flags equal to -1. Since process zero is busy at step s_A, the communication couple is set at step s_B as shown in Figure 9.9c. This procedure is repeated until there are no more processes to check. The final configuration of the communication couples is split into three steps as shown in Figure 9.9d, Figure 9.10a, Figure 9.10b and Figure 9.10c.

Read and write. Input files are read by process zero and broadcast to all processes where each of the processes only creates its own objects. While reading is centralised and only performed at the first step, the writing of data is not.

It does not make sense to spend time collecting data (output buffer) between processes to then only write it on a single process (usually process zero). It is far simpler to allow each process to write onto the hard-drive independently (especially on HPC) and to analyse the data from the simulation after it is finished. In this work the open source software ParaView is used to combine different output files into one visualisation.

9.2.2 Send receive send/recv

Objects in C++ are complex entities, containing different kinds of data. Sending an object or part of it in an MPI message is not an easy task. One alternative possibility is to build custom made data types on MPI.[61] This would mean that for any new object/class that has to be sent/received a new data type has to be built.

In this work only three fundamental data types are employed

- MPI_LONG
- MPI_DOUBLE
- MPI_CHAR

Each message to be sent/received has these three types. Dividing the data in this manner means that a message has to be sent/received for each data type to be sent. This small price (more messages instead of one in the case of MPI data types) in CPU time, is compensated by the simplicity of the MPI implementation.

The operation of mapping an object to the buffer is shown schematically in Figure 9.11, where different "engines" are employed in the transformation. For example, if only contact forces are exchanged, there is no need to map deform forces.

To speed up the transfer operation, send and receive is carried out at the same communication time using MPI_Sendrecv.

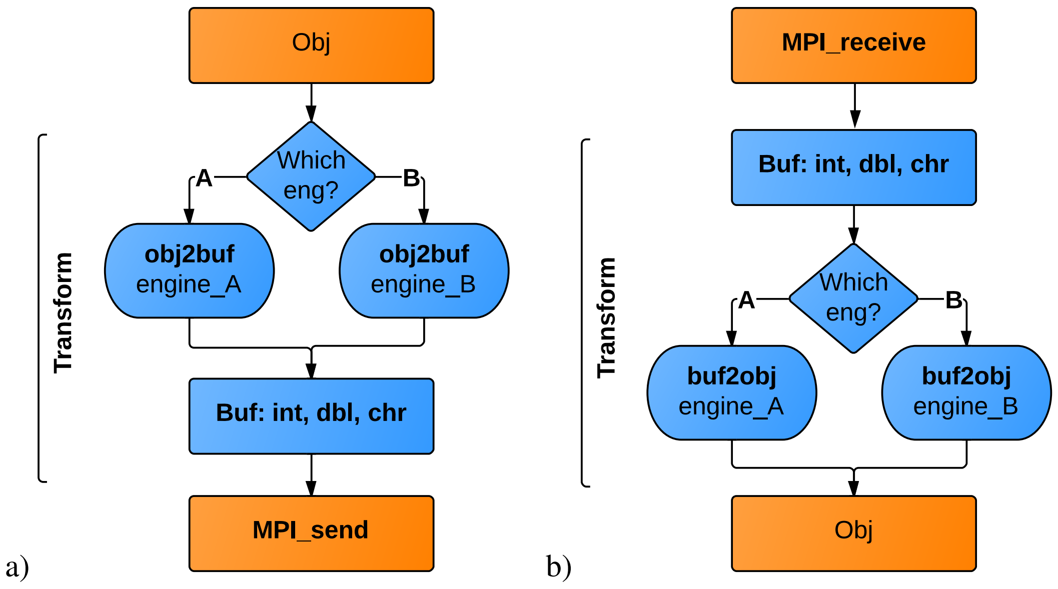


Figure 9.11: Mapping of an object to buffer. a)Send. b)Receive

9.2.3 Set originals and proxies

Objects are divided in this work into two main categories, depending on the type of data required to perform the task assigned to them:

- Independent: Does not need any object other than itself to perform its task.
- Manipulator: Points to one or more *independent* objects and/or one or more *manipulators*, from which it extracts data and/or modifies to perform its task.

In the FDEM each DE is made of a series of FEs (i.e. a triangle, tetrahedra) glued together using joints, as it was explained in Chapter 3. Each finite element is categorised as *independent* and needs only to "know" of coordinates, current velocities and current forces, without any details regarding materials properties or joints. Following along this line of thinking, for each FE there is one object called *deform* of the type *manipulator* pointing to it. The object *deform* has all the material properties and laws of deformation, which allows the implementation of different materials, only changing one object and not all of the code. Similarly, joints are categorised as *manipulators* as they need to get information from two FEs to calculate joint forces.

The setting of originals and proxies has to be organised hierarchically. Firstly, *independent* objects have their processor rank set, then *manipulator* objects that only depend on one object have their processor rank set, and subsequently *manipulator* objects pointing to two objects have their processor rank set and so on.

Set originals. Objects that require setting their original process rank can be divided into two groups:

- Set original by the spatial position of the object.
- Set original by other criteria, i.e. by function. Object A operating on an object B.

The first group is made of FEs, interaction points, where the original position is calculated using the Algorithm 9.2 and where the function *IsOriPntInsDom* is shown in Algorithm 9.3. This is the classic criteria for setting original rank.

The second group consists of objects that do not necessarily have a spatial position. The criteria to set the original rank depends on each particular object. For example, the object *deform* has its original rank defined by the FE that it manipulates, for these kinds of objects the original rank is set using the Algorithm 9.4.

Algorithm 9.2 Set original by spatial position.

```
1: $obj2setOri$    ▷ Object to set original
2: $ListBouPro$    ▷ List boundary processes. Single connected list element
3: $ListBouPro = GetAllBouPro\,(obj2setOri \to GetProOriRank\,())$ ▷ The first one is OriPro, and the rest of the element in the list are all the processes neighbours to it
4: $obj2setOri \to SetProOriRank\,(-1)$
5: while $((obj2setOri \to GetProOriRank\,() == -1)\,\&\&\,(ListBouPro != \text{Null}))$ do
6:     if $ListBouPro->IsOriPntInsDom\,(obj2setOri \to GetCentre\,())$ then
7:         $obj2setOri \to SetProOriRank\,(ListBouPro \to GetRank\,())$
8:     end if
9:     $ListBouPro = ListBouPro \to GetNext\,()$
10: end while
11: if $(obj2setOri \to GetProOriRank\,() == -1)$ then
12:     $WriteErrorObjWithOutOriginalProcess\,()$
13: end if
```

Set proxies. Proxies can be defined in a similar way as "set original" by their position, or by other criteria. When a proxy object is defined by its position, then algorithm 9.3 is modified to take into account $\triangle_{\text{prc}}$ shown in Figure 9.10b and Figure 9.10c, where $\triangle_{\text{prc}}$ is given by

$$\triangle_{\text{prc}} = n_{mov\_ori}\,2\,v_{max}\triangle t \tag{9.3}$$

where n_{mov_ori} is the number of steps (frequency) to move originals and proxies, v_{max} maximum velocity of any object and $\triangle t$ delta time. The modified algorithm is shown in Algorithm 9.5 for a point proxy. The increase in the dimensions of the domain by $\triangle_{\mathrm{prc}}$ is to account for possible displacements of entities (in other border processes) that may be in contact with objects inside the domain between calls to move originals and proxies.

Algorithm 9.3 Check if original point is inside domain. Where x_{min}, y_{min}, z_{min} are the minimum coordinates of the domain and x_{max} ,y_{max}, z_{max} are the maximum coordinates of the domain. To account for numerical rounding errors all domain dimensions are increased by ε.

```
1:  function IsOriPntInsDom(point)                ▷ point object with 3 coordinates.
2:      if $((point \rightarrow x < (x_{min} - \varepsilon)) \,||\, (point \rightarrow x > (x_{max} + \varepsilon)))$ then
3:          return no
4:      end if
5:      if $((point \rightarrow y < (y_{min} - \varepsilon)) \,||\, (point \rightarrow y > (y_{max} + \varepsilon)))$ then
6:          return no
7:      end if
8:      if $((point \rightarrow z < (z_{min} - \varepsilon)) \,||\, (point \rightarrow z > (z_{max} + \varepsilon)))$ then
9:          return no
10:     end if
11:     return yes
12: end function
```

Algorithm 9.4 Set original by *manipulated* object.

```
1: $objA$                                          ▷ In example a finite element
2: $objB$                                          ▷ In example a deform operator
3: $objB \rightarrow SetProOriRank(objA \rightarrow GetProOriRank())$
```

Move originals and proxies. Developing parallel solutions using C/C++ presents a particular problem with data types called "pointers". Pointers are used to speed up access to different data/objects. If an object is using a pointer, it is moved from one process to another will "expect" to have its pointer set in the new process. If a pointer is not initialised (pointing to a random address of the memory) it will usually cause the program to crash.

It is not possible to send a pointer as such (address to a block in the RAM memory) in MPI. What is needed is a way to map a pointer to a different kind of data. The most intuitive data type to use is an integer. If all objects that are going to be moved are assigned a unique *id*[4] then it is possible to refer to a pointer not by its memory address, but by the unique *id* of the object being pointed to.

Algorithm 9.5 Check if proxy point is inside domain. Where x_{min}, y_{min}, z_{min} are the minimum coordinates of the domain and x_{max} ,y_{max}, z_{max} are the maximum coordinates of the domain.

```
function IsProPntInsDom(point)                ▷ point object with 3 coordinates.
    if $((point \to x < (x_{min} + \triangle_{\text{prc}})) \,||\, (point \to x > (x_{max} + \triangle_{\text{prc}})))$ then
        return no
    end if
    if $((point \to y < (y_{min} + \triangle_{\text{prc}})) \,||\, (point \to y > (y_{max} + \triangle_{\text{prc}})))$ then
        return no
    end if
    if $((point \to z < (z_{min} + \triangle_{\text{prc}})) \,||\, (point \to z > (z_{max} + \triangle_{\text{prc}})))$ then
        return no
    end if
    return yes
end function
```

Different strategies can be applied to deal with pointers in MPI:

- Keep strict order of arrival to each process. Therefore, the first objects that do not depend on other objects arrive first, and so on. Pointers are set as soon as the object is created.
- Set pointers to NULL and have rules to deal with NULL pointers.
- Do not use pointers on objects that are going to be moved between processes.
- Set pointers after all objects have been moved.
- A combination of the previous rules.

To set pointers on objects that have been received, it is necessary to have a database where one can find an object by its *id*. It is possible to use a binary tree (Chapter 6) to build such a database of objects. This database can be used to set pointers as objects that are arriving, or to set pointers after all objects have been moved. There is no simple answer or a silver bullet to solve this problem. In this work a mix of these strategies has been implemented.

[4]It could be a vector *id* i.e. made of two integers (processRank,objectNum)

9.2.4 Algorithm

The golden rule in any parallel code is to avoid unnecessary communications. It is less expensive in CPU terms to repeat the same operation in all processes than to perform it in one and communicate the results to the rest of processes. A good example of this philosophy is the *zero* operation in which FEs have their forces set to zero. Finally the complete procedure is shown in Algorithm 9.6.

Algorithm 9.6 Parallel algorithm for FDEM."$\rightleftharpoons$" means sendReceive operations (communication between processes), "ori" means original and "prx"proxy.

```
 1: IniMPI                      ▷ Ini MPI and set processes communication topology
 2: $\rightharpoonup$ ReadInput             ▷ Process zero reads input, broadcast to all processes
 3: while ($iLv\_i < iMv\_numStp$) do
 4:     FeZero       ▷ FE(ori) calc contact pos. FE(ori) and FE(prx) set forces to zero.
 5:     $\rightleftharpoons$ o2p_pvCon               ▷ FE ori2prx send position and velocity contact
 6:     if ($doMov\_ori$) then
 7:         $\rightleftharpoons$ MovOriPrx                 ▷ All objects, calc and move ori and prx.
 8:     end if
 9:     JointForces                          ▷ Joint calculate forces on ori joints
10:     if ($fracturePar$) then
11:         $\rightleftharpoons$ FracturePar                                        ▷ Fracture
12:     else if ($fractureSeq$) then
13:         FractureSeq                                          ▷ Fracture
14:     end if
15:     PntZero  ▷ Interaction point (ori and prx) calculate position, set forces to zero
16:     FeDef                                  ▷ FE(ori) calculate forces deform
17:     FeConFor                                  ▷ FE(ori) apply force conditions
18:     if (($doCD$) || ($doMov\_ori$) || ($fractureAny$)) then
19:         DeleteDataBaseInteraction
20:         CD_search              ▷ CD_search, build interaction database. Chapter 6
21:     end if
22:     Interaction                    ▷ Perform interaction on interaction database
23:     PntPassFor2Fe                     ▷ Interaction point (ori) pass force to FE
24:     $\rightleftharpoons$ p2o_fConDef                 ▷ FEs prx2ori send force contact and deform
25:     $\rightleftharpoons$ o2p_fCon                              ▷ FE ori2prx send force contact
26:     FeAvConFor                             ▷ FE(ori) average contact force
27:     FemConVel                            ▷ FE(ori) apply velocity conditions
28:     WriteOutput                                           ▷ Write output
29:     FeMov        ▷ FE(ori) move. Calculate new position using central difference
30:     $\rightleftharpoons$ o2p_pvDef                   ▷ FE ori2prx send position velocity deform
31: end while
32: EndMPI
```

9.3 Test cases

Three different cases are employed to test the performance of this parallel algorithm. In Chapter 6 simple discrete elements (DEs) and up to 1.331 million finite elements (FEs) were tested. Here, more complex DEs are tested. All the simulations were performed on a Multicore PC DELL Precision T5400 with one processor of four cores and 32 G of RAM. Each of the tests were performed sequentially (one core) and in parallel with two cores and with four cores.

Generally, in dynamic simulations there is always some type of unbalance between different processes. In this work, to obtain the time to perform of a particular operation, i.e. contact detection, the total time is averaged between all processes namely

$$t_{par} = \frac{\sum_{i=0}^{n_{prc}} t_i}{n_{prc}} \tag{9.4}$$

where t_{par} is the parallel time, t_i is the time of process i and n_{prc} is the number of processes.

As the amount of simulated objects increase, the communication time relative to the total simulation time decreases. The decrease is related to the ratio between the total number of objects N given by

$$N \propto V_{dom} \tag{9.5}$$

where V_{dom} is the volume of the domain, and the total the total number of objects to communicate N_{com} given by

$$N_{com} \propto A_{bou} \tag{9.6}$$

where A_{bou} is the boundary area between processes. N_{com} does not normally increase as fast as N. The efficiency of a parallel solution is related to the relationship between these two numbers.

The speed up was explained in Chapter 3, and is reproduced here for clarification reasons. The relationship between the sequential and parallel runs is given by

$$S = \frac{t_{seq}}{t_{par}} \tag{9.7}$$

where t_{seq} is the sequential time and t_{par} is the parallel time. While the standard ratio $R_{\%}$ is given by

$$R_{\%} = \frac{t_{par}}{t_{seq}}100 \tag{9.8}$$

Times are measured without taking into account the time spent writing output files to the hard-drive. Usually, in multicore systems a program will attempt to write into the same hard-drive regardless of the particular process (0,1,..,$n_{processes}$). As more than one process tries to access the hard-drive, the writing times increase. This *idle* time is not taken into account.

9.3.1 MPI test A

The test consists of a raster of DEs (boxes) made of 24 FEs each. Random velocities are assigned to each box at time equal to zero from 0.0 m/s to 137 m/s. The mechanical properties are: elastic modulus E=4 MPa Poisson's ratio $\nu = 0.45$ and penalty $\sigma_p =$ 400 MPa. DEs are not allowed to break.

Each FE has one interaction point centre on each one of its faces. A spherical boundary with penalty $\sigma_p = 400$ MPa is imposed to keep all DEs interacting with each other.

Initial and final configurations for the raster of 4066 boxes are shown in Figure 9.12 for two and four processes.

FEs	DE (Boxes)	2 prs: S	4 prs: S	2 prs: $R_{\%}$	4 prs: $R_{\%}$
1536	64	1.87	2.92	53.46	34.29
12288	512	1.93	3.67	51.83	27.23
98304	4096	1.98	3.87	50.63	25.82
526848	21952	1.99	3.93	50.33	25.42

Table 9.1: Total simulation time S and $R_{\%}$.

FEs	DEs (Boxes)	2 prs: S	4 prs: S	2 prs: $R_{\%}$	4 prs: $R_{\%}$
1536	64	2.01	4.12	49.71	24.26
12288	512	2.03	4.12	49.36	24.26
98304	4096	2.05	4.18	48.86	23.93
526848	21952	2.05	4.19	48.86	23.87

Table 9.2: Total CD-MR time S and $R_{\%}$

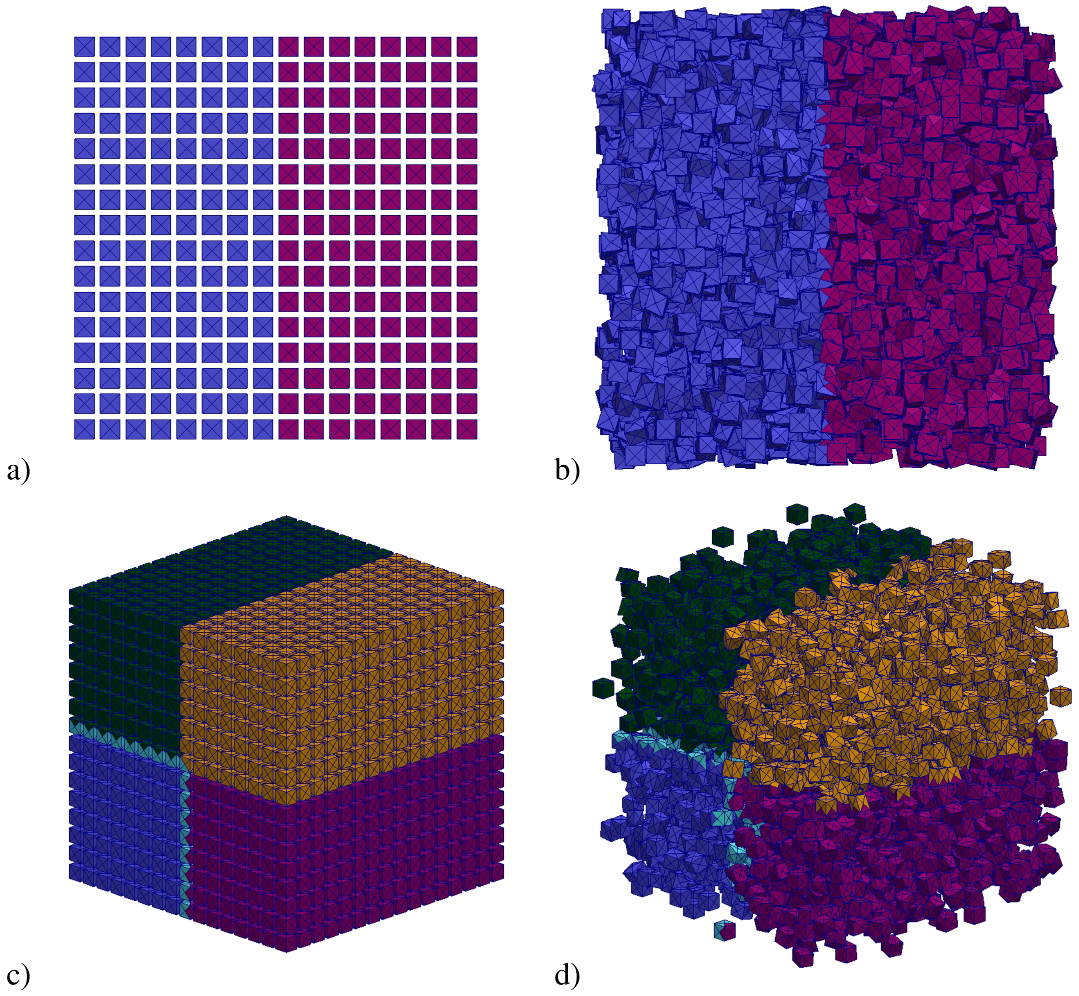


Figure 9.12: Geometric domains two and four processes. The colours scheme represents different processes. a)Time 0.0ms. b)Time 0.36ms. c)Time 0.0ms. d)Time 0.36ms, in light blue the proxies of process zero (blue) are shown.

The total simulation times S and $R_{\%}$ are shown in table 9.1, where $R_{\%}$ values gradually convert to the ideal values of 50% for two processes and 25% for four processes. And S converges to the theoretical values of 2.0 for two processes and 4.0 for four processes.

CD-MS (clean database, MS-load, MS-search, add to database of contact couples, clean MS-tree) times are shown in table 9.2, where S values exceed the maximum theoretical value of 2.0 for two processes and 4.0 for four processes. The small discrepancies from the theoretical values are due to the non-linearity of the CD-MS algorithm.

The total time is shown in Figure 9.13 while the total time for CD-MS is shown in Figure 9.14.

Finally the kinetic energy for each of the rasters is shown in Figure 9.15, where there is an excellent agreement between sequential and parallel solutions.

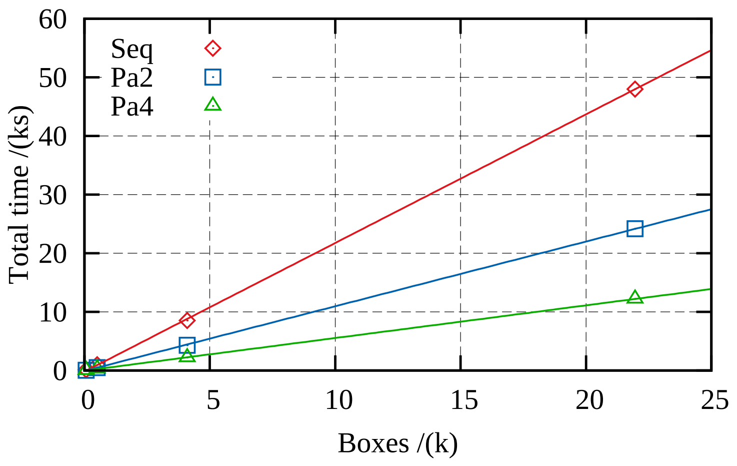


Figure 9.13: Total simulation time. The continuous lines are linear regressions.

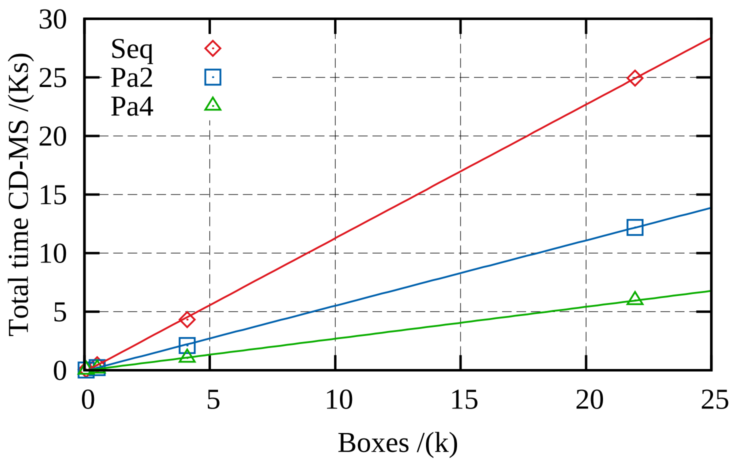


Figure 9.14: Total CD-MR time. The continuous lines are linear regressions.

9.3.2 MPI test B

All conditions are identical to the previous test, with the difference that DEs objects are allowed to fracture. The development of the simulation for the raster of 4066 boxes is shown in Figure 9.16.

Fracture procedure changes the geometry of a DE object. FEs that were deeply embedded inside a DE can become boundary FEs. If a fracture is produced then CD has to be performed.

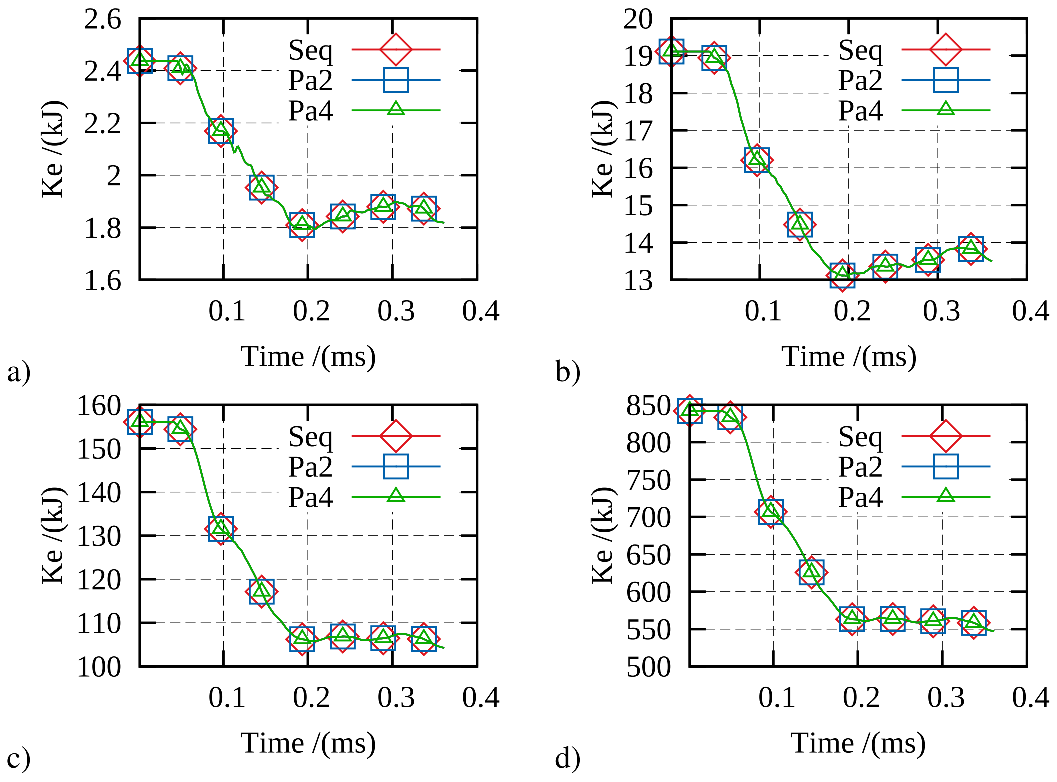


Figure 9.15: Kinetic energy. a)64 boxes. b)512 boxes. c)4096 boxes. d)21952 boxes.

In this test there is a fracture in almost every step of the simulation. This is reflected on the CD time shown in Figure 9.18. Not only is CD performed more often but contacts that may theoretically occur in $n_{\triangle_{CD}}$ are calculated and added to the database of contact couples, just to be deleted and recalculated in the next step.

The CD-MS and the parallel algorithm perform as expected (minus the increase in the CD time) with the total time shown in Figure 9.17.

S and $R_{\%}$ are shown in table 9.3 for the total time and in table 9.4 for the contact detection time. Finally, the kinetic energy for each of the rasters is shown in Figure 9.19, where there is a good agreement between sequential and parallel simulations.

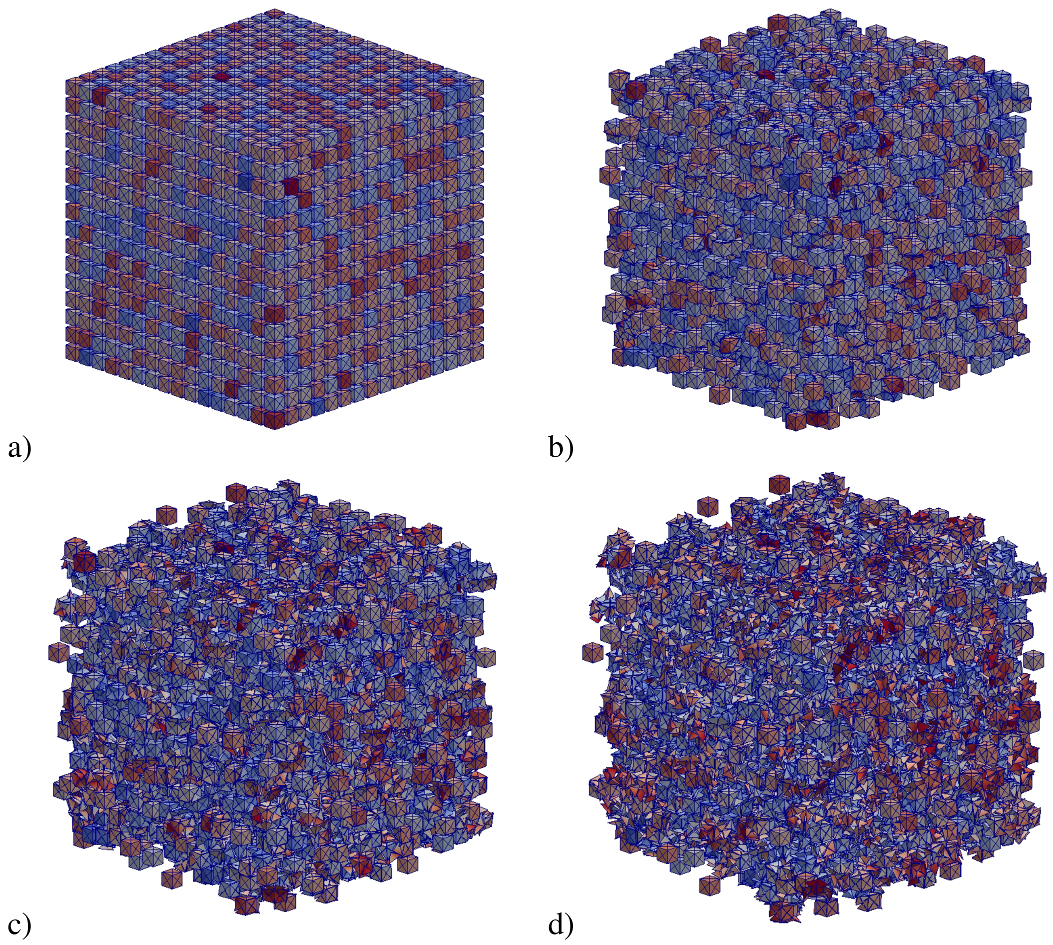


Figure 9.16: Simulation four processes with 4096 boxes. The colours represent the magnitude of the velocities. a)Time 0.0ms. b)Time 0.12ms. c)Time 0.24ms. d)Time 0.36ms.

FEs	DEs (Boxes)	2 prs: S	4 prs: S	2 prs: $S_{\%}$	4 prs: $S_{\%}$
1536	64	1.85	2.92	53.91	34.24
12288	512	1.98	3.80	50.61	26.31
98304	4096	2.01	3.94	49.75	25.37
526848	21952	2.02	4.03	49.60	24.84

Table 9.3: Total simulation time S and $R_{\%}$.

FEs	DEs (Boxes)	2 prs: S	4 prs: S	2 prs: $R_{\%}$	4 prs: $R_{\%}$
1536	64	2.11	4.27	47.47	23.41
12288	512	2.03	4.07	49.16	24.56
98304	4096	2.04	4.08	48.99	24.49
526848	21952	2.04	4.13	49.05	24.22

Table 9.4: Total CD-MR time S and $R_{\%}$

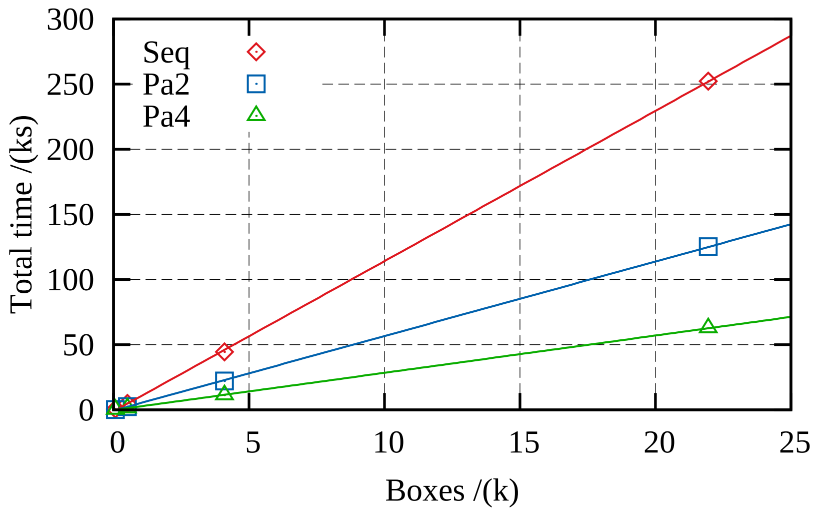


Figure 9.17: Total simulation time. The continuous lines are linear regressions.

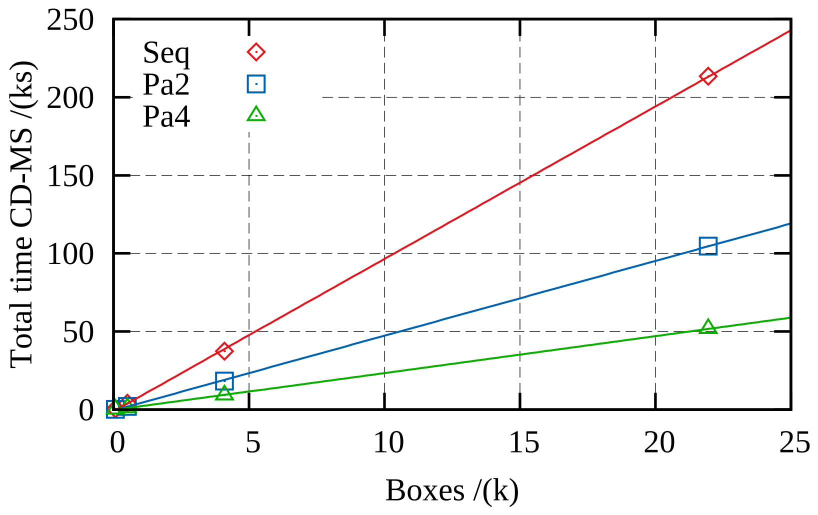


Figure 9.18: Total CD-MR time. The continuous lines are linear regressions.

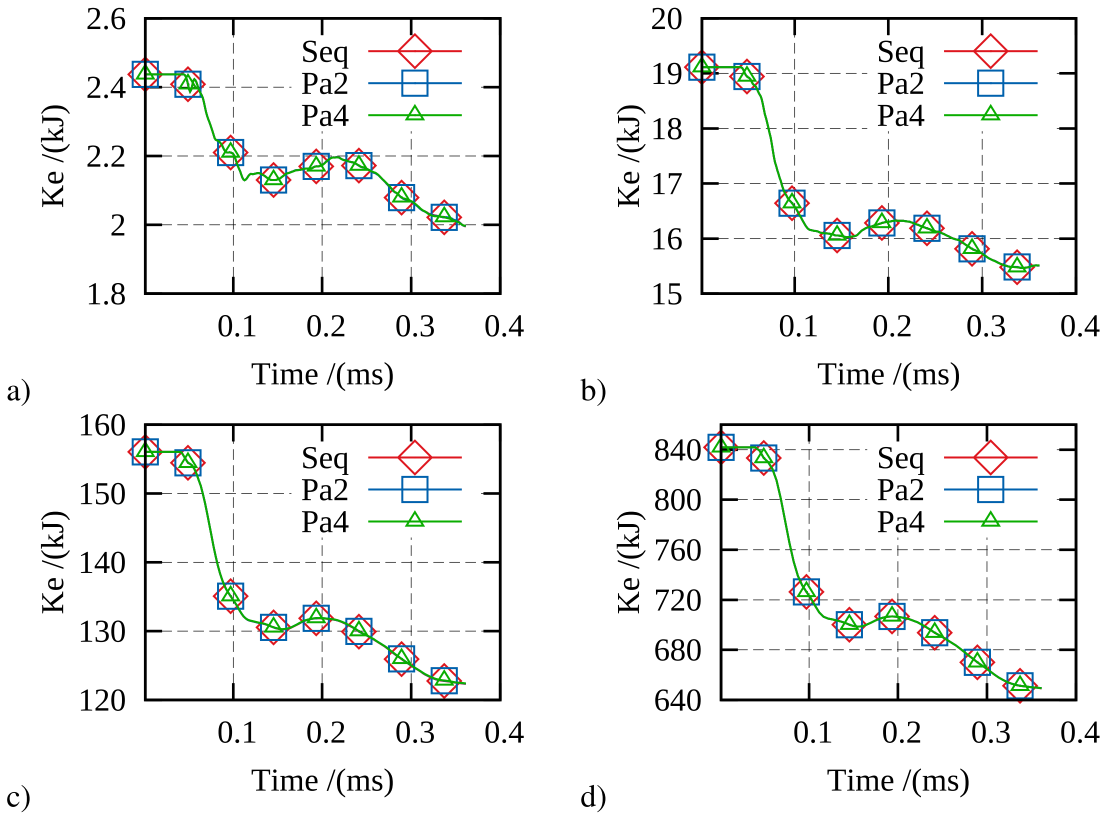


Figure 9.19: Kinetic energy. a)64 boxes. b)512 boxes. c)4096 boxes. d)21952 boxes.

9.3.3 MPI test C

The test[140] consists in a rectangular discrete element, with a discontinuity into which pressure is applied as shown in Figure 9.20a. The domain decomposition topology for the mesh with 442368 FEs is shown in Figure 9.20b for two processes and in Figure 9.20c for four processes.

The pressure is gradually increased via the function as

$$\sigma_t = 20t\,[\text{GPa/s}] \tag{9.9}$$

where t is time, the elastic modulus $E = 26.6$ GPa and Poisson's ratio is 0.205. Interaction points are only created on the discontinuity. To test the performance under static conditions only the first 2000 steps are run.

The CD-MS time as expected is almost null ≈ 0.14 ks (for 442 kFEs) when compared with the total time of ≈ 58 ks. Contact detection times are not taken into account the analysis of this test. The speed up S and the ratio $R_{\%}$ for the total time are shown in table 9.5.

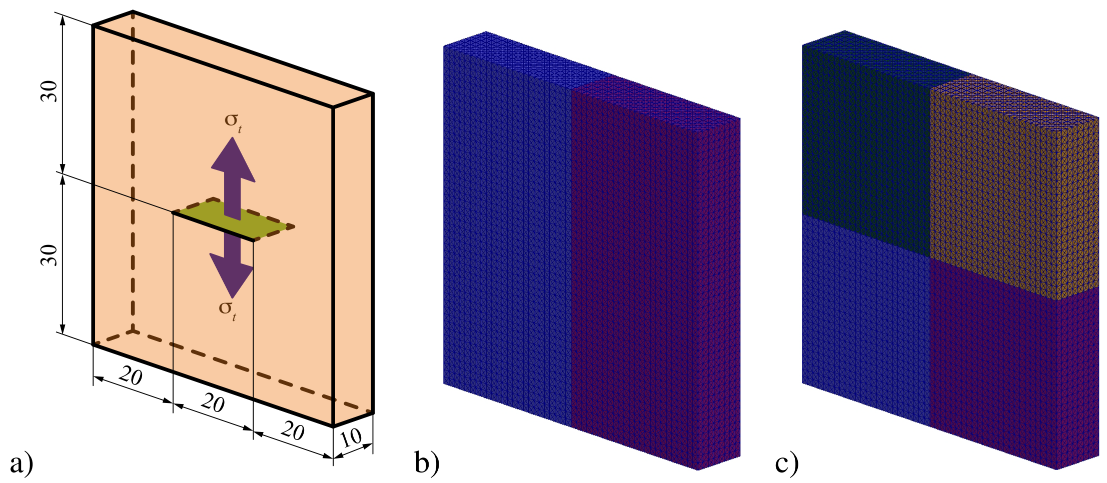


Figure 9.20: Geometry definitions. a)Dimensions mm b)Domain topology two processes. c)Domain topology four processes.

The speed up S for two and four processes shown in table 9.5 are below the ideal theoretical values of 2.0 and 4.0 respectively as expected. Since the problem is semi-planar, the ratio between the total amount of objects N (equation 9.5) and the amount of objects to communicate N_{com} (equation 9.6) does not change as fast as for the previous two tests.

On the other hand as the CD-MS has a minor influence, these values can be used to address the performance of the algorithm without CD.

Elements	2 prs: S	4 prs: S	2 prs: $R_{\%}$	4 prs: $R_{\%}$
264	1.85	3.29	54.11	30.43
6912	1.94	3.65	51.51	27.39
55296	1.94	3.67	51.45	27.28
442368	1.97	3.76	50.67	26.56

Table 9.5: Total simulation time S and $R_{\%}$.

The total time is shown in Figure 9.21 where the non-linearities attributed to the CD-MS are not present. The stresses $\sum\left|\sigma_{yy}\right|$ are shown in Figure 9.22, where it is possible to appreciate the similarity between sequential and parallel tests.

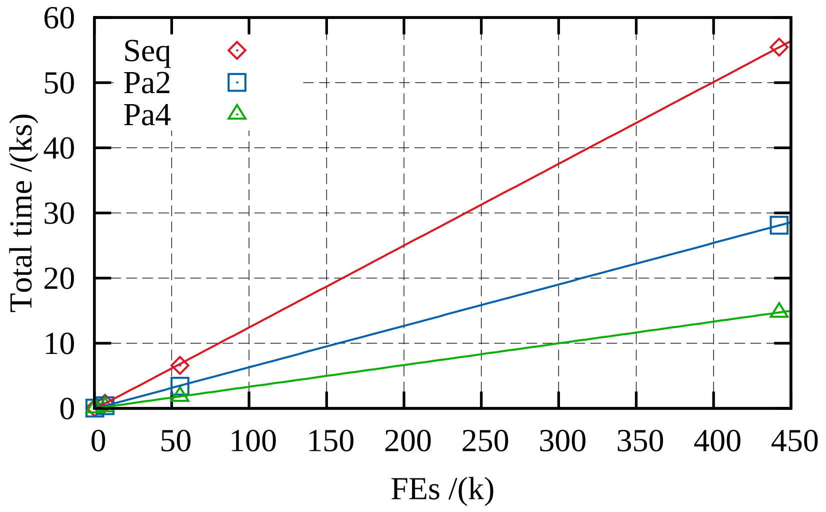


Figure 9.21: Total simulation time. The continuous lines are linear regressions.

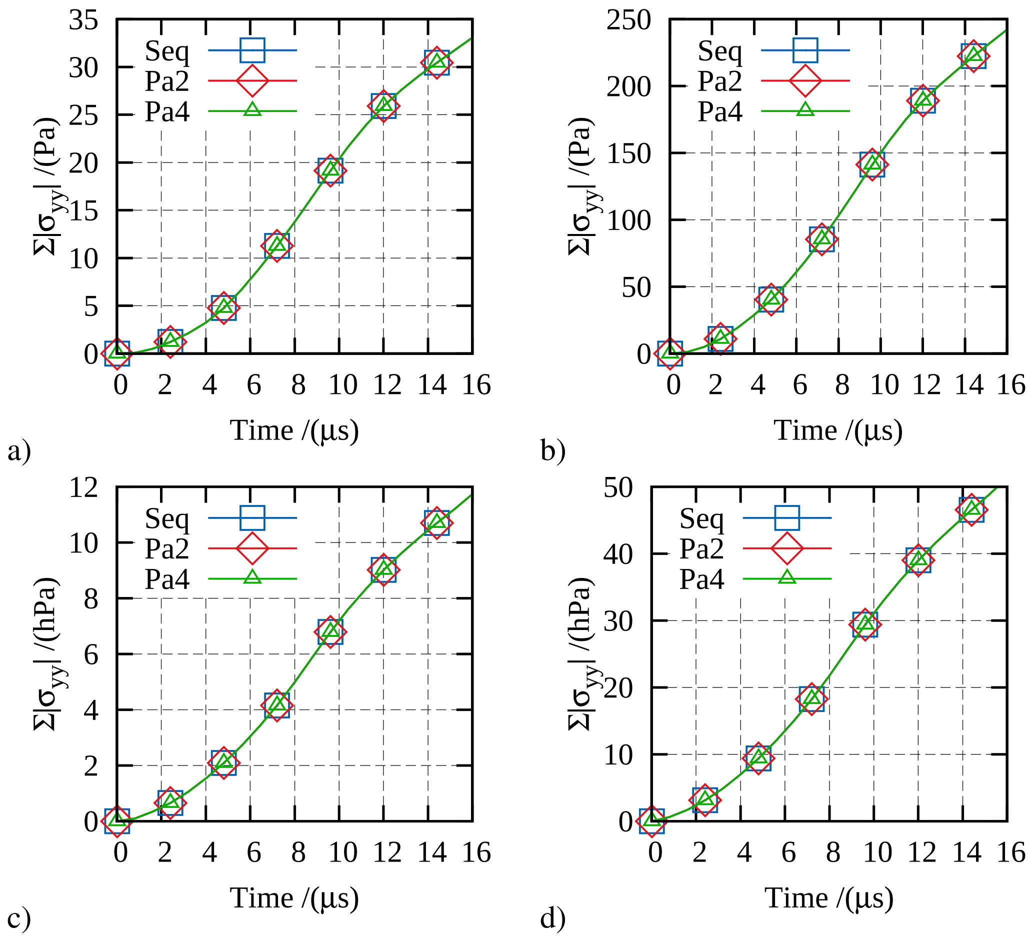


Figure 9.22: $\sum \left|\sigma_{yy}\right|$. a)264 FEs. b)6912 FEs. c)55296 FEs. d)442368 FEs.

9.4 Conclusion

The performance of this novel MPI algorithm when compared with a sequential algorithm, is good. Physical quantities such as Ke and stress are preserved during the simulations.

Non-linear behaviours are observed in the total simulation times for test cases A and B. These can be attributed to non-linearities of the CD-MS algorithm. If more processes are used the parallel efficiency is expected to decrease, as each process has to communicate with more neighbours increasing the total communication time.

Chapter 10

APPLICATION AND EXAMPLES

Any newly developed codes have to be tested against theoretical, empirical and computational solutions.[211,164,117,195] To test this work a set of cases involving tension, compression and fracture are chosen to determine the behaviour in different situations. Structured and unstructured meshes are employed.

FDEM have to behave and reproduce results expected by the FEM method in the absence of fracture. The first example is of a cantilever beam with a distributed load test. The second example is a true FDEM where results in 3D fracture are compared with a previously published 2D fracture numerical test. The third test involves the interaction of a projectile against a rectangular glass; the obtained fracture patterns are compared with published data. The final test simulates the collapse of a hyperboloid cooling tower structure with two different initial conditions, obtaining different fracture patterns.

10.1 Cantilever beam under distributed load

This example is of the classic cantilever beam bending under its own weight. The material is simulated as linear elastic with the following properties: elastic modulus $E = 2$ MPa, Poisson's ratio $\nu = 0.25$ and density $\rho = 2340$ $^{kg}/_{\mathrm{m}^3}$. The dimensions and boundary conditions are shown in Figure 10.1a. The gravity is increased gradually from zero to the maximum value of 9.81 $^{m}/_{s^2}$ at time equal to 2.4 s as shown in Figure 10.1b.

The maximum theoretical deflection of a cantilever beam under a distributed load along the full length of the beam[17] is given by the Equation 10.1

$$y_{max} = \frac{wL^4}{8EI} = \frac{\rho AgL^4}{8EI} = 5.756\,\text{mm} \qquad (10.1)$$

where , w is the load per unit length, L is the length of the beam, E is the elastic modulus, I is the second moment of area, ρ is the density, g is the gravity acceleration and A is the cross section area. The test is performed on two processes using three different meshes, as shown in Figure 10.3.

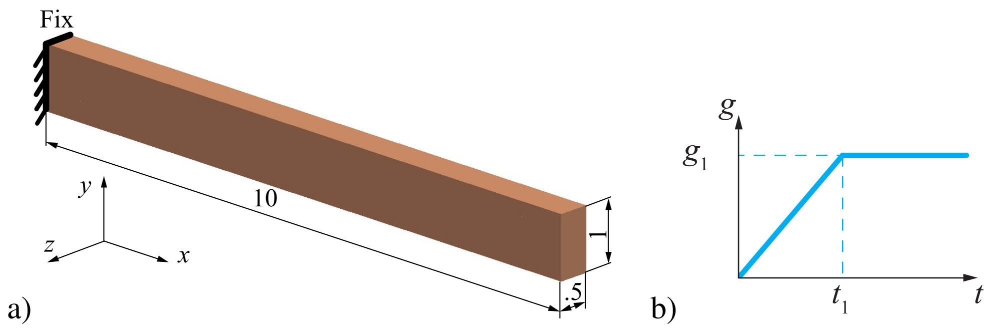


Figure 10.1: a)Dimensions (m) and boundary conditions. b)Gravity function of time where g_1=9.81 m/s^2 and $t_1 = 2.4$ s.

The maximum deflections are shown in Figure 10.2 and the final deflections in table 10.1. As expected as the number of FEs increase, the maximum deflection approaches the theoretical value of 5.756 mm.

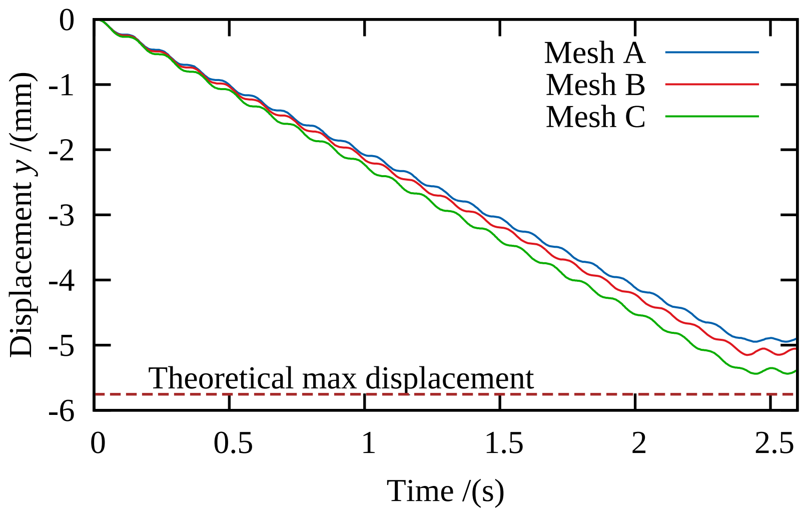


Figure 10.2: Maximum deflections.

Mesh	Number of elements	Maximum deflection mm
A	960	4.9
B	1920	5.1
C	6325	5.4

Table 10.1: Maximum deflections.

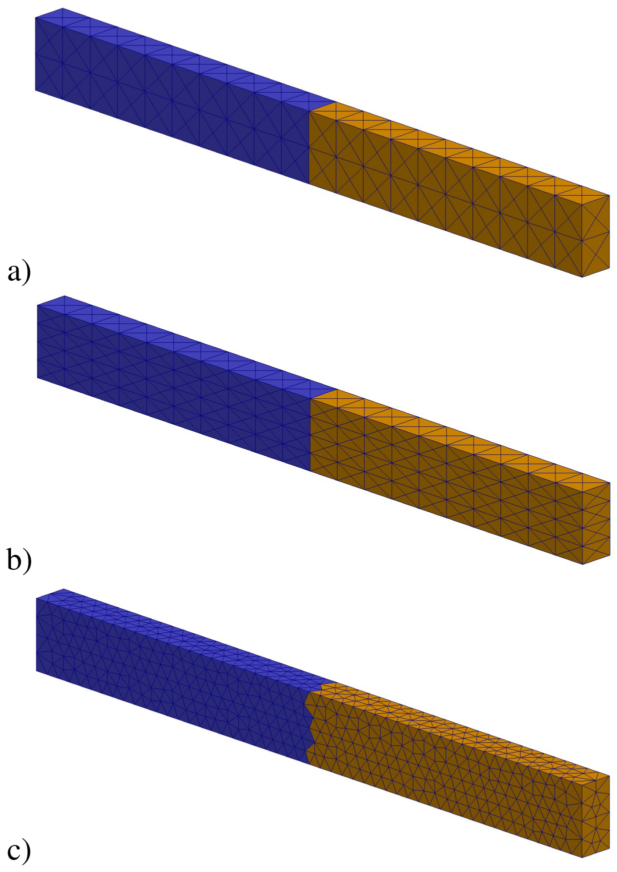


Figure 10.3: Spatial distribution of elements between processors. The colour scheme represents different processes. a)Mesh A, 960 FEs . b)Mesh B, 1920 FEs. c)Mesh C, 6325 FEs.

10.2 Fracture sensitivity

To capture the plastic zone in a fracture process it is necessary that the size of the mesh (mesh discretization) should be smaller than that of the plastic zone. Where the lower limit of the plastic zone for a short Mode I fracture, for an “infinite body under plane stress conditions”[140] is given by

$$\triangle_{low} = \frac{E}{4f_t}\triangle_t \tag{10.2}$$

where E is the elastic modulus, $f_t$is the tensile strength and $\triangle_t$ is the separation at which the stress in the fracture is equal to zero. The lower limit of the plastic zone for a long crack in Mode I is given by

$$\triangle_{long} = \frac{\pi E}{32 f_t}\triangle_t \tag{10.3}$$

The tensile separation $\triangle_t$ for the smeared crack model is given by the equation 3.17, reproduced here for clarity reasons

$$\triangle_t = \frac{G_f}{A_{sof} f_t} \tag{10.4}$$

where G_f is the fracture energy and A_{sof} is the area of the softening branch of the stress-displacement curve. The equations 10.2 and 10.3 can be used as reference for the biggest size h_{max} of an FE that can capture the plastic zone of the fracture as

$$h_{max} < Min\left(\triangle_{low}, \triangle_{long}\right) \tag{10.5}$$

For sizes $h > h_{max}$ the stress around the crack opening are not captured and the resulting fracture is similar to the one obtained under “uniform stress distributions”[140]. On the other hand, for finer meshes the stress is better captured and the plastic zone is spread on more FEs.

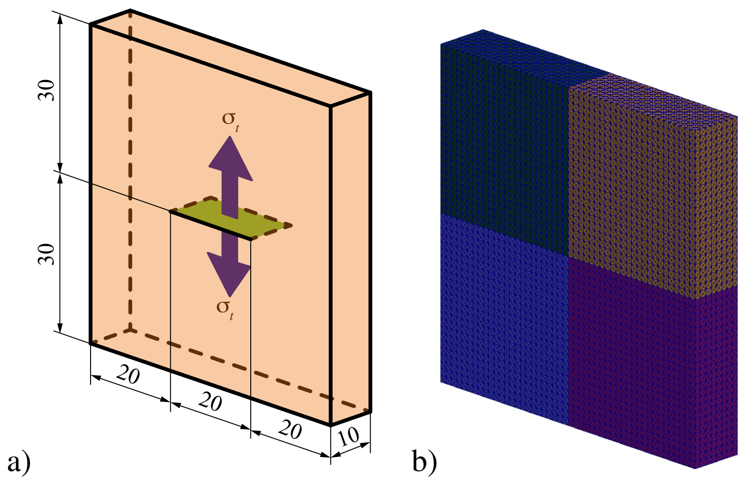


Figure 10.4: Geometry. a)Dimensions mm. b)Domain divided into 4 processes.

Munjiza originally developed the fracture sensitivity test[140] originally in 2D, but here the test is extended to 3D, using the parallel algorithm developed in Chapter 10. It is the same test presented in the previous chapter, but in this case the test is only run on four processes until a fracture is produced. The test consists of a discrete rectangular element with a discontinuity in the middle, as shown in Figure 10.4a.

A uniform pressure is gradually increased on this discontinuity as

$$\sigma_t = 20t\,[\mathrm{GPa/s}] \tag{10.6}$$

where t is time. The material properties are: elastic modulus $E = 26.6$ GPa, Poisson's ratio is 0.205, tensile strength $f_t = 5$ MPa, density $\rho = 2340$ kg/m^3 and fracture energy $G_f = 30$ N/m.

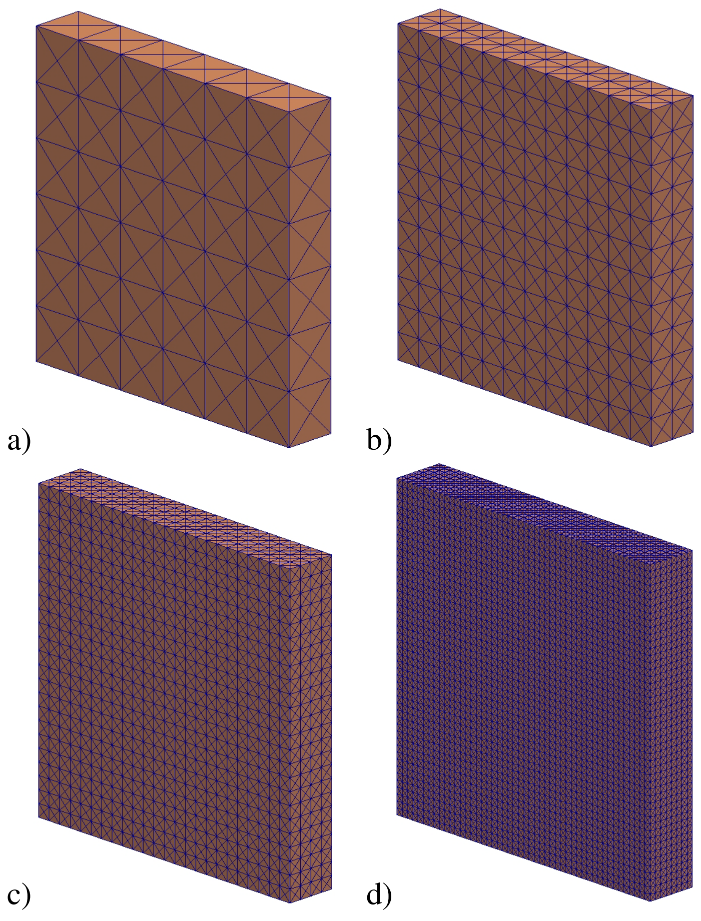


Figure 10.5: Mesh topology. a)Mesh A, 864 FEs b)Mesh B, 6912 FEs. c)Mesh C, 55296 FEs. d)Mesh D, 442368 FEs.

The parameters for the curve z (equation 3.14) are $a = 0.63$, $b = 1.8$ and c=6. Therefore with these values, h_{max} should be

$$\triangle_t = \frac{G_f}{A_{sof} f_t} = 15.47\,\mu\text{m} \tag{10.7}$$

$$\triangle_{low} = \frac{E}{4f_t}\triangle_t = 20.6\,\text{mm} \tag{10.8}$$

$$\triangle_{long} = \frac{\pi E}{32 f_t}\triangle_t = 8.1\text{mm} \tag{10.9}$$

$$h_{max} < 8\,\text{mm} \tag{10.10}$$

Four different meshes are tested with $h_A = 10$ mm, $h_B = 5$ mm, $h_c = 2.5$ mm, and h_D=1.25 mm as shown in Figure 10.5. The fracture initialisation and fracture growth are shown in Figure 10.7 for mesh D, where it is possible to appreciate the total separation between the top and bottom of the discrete element at the end of the simulation.

The load value of the stress σ_t at the beginning of the fracture growth (first fracture) is shown in Figure 10.6, presenting good agreements with the values obtained by Munjiza in 2D. Comparing the sizes of the mesh with the theoretical limit[140] given by the equation 10.10, meshes C and D should give relatively good results as the plastic zone is discretized on more than 3 FEs.

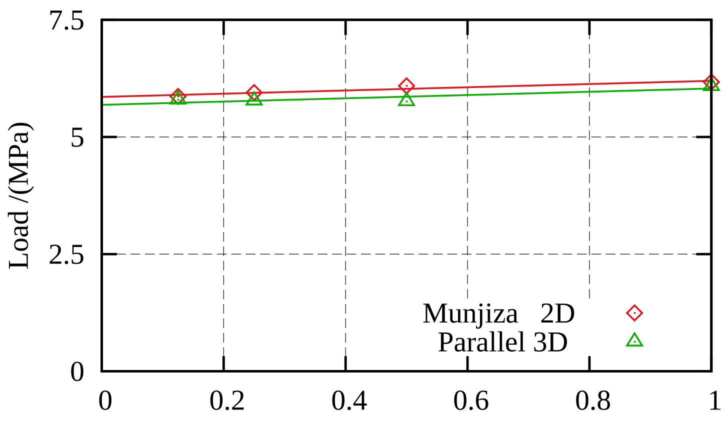


Figure 10.6: Fracture. The continuous lines are linear regressions.

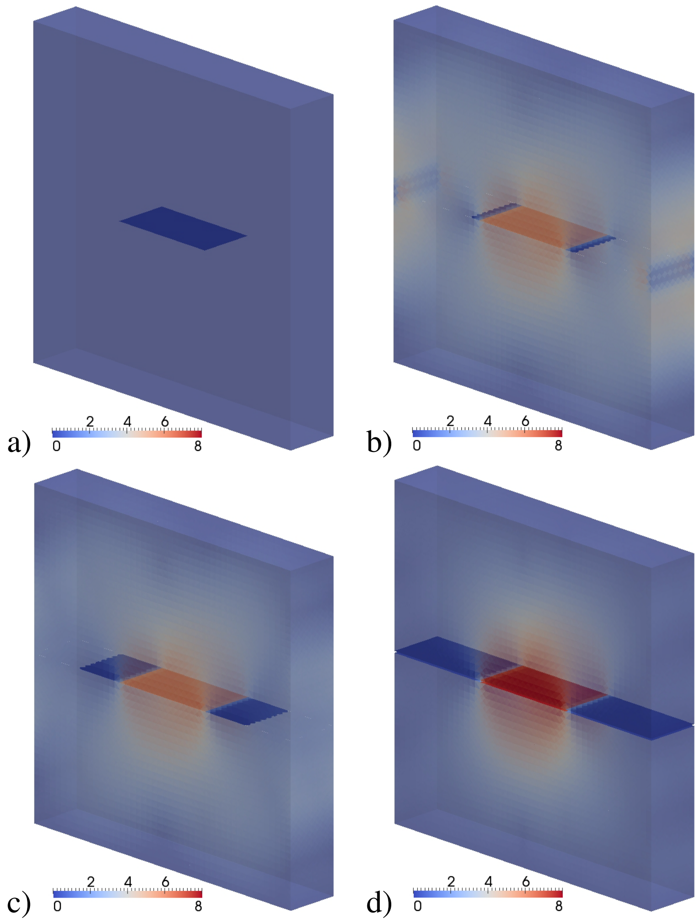


Figure 10.7: Fracture growth. The colour scheme is the modulus of σ_{yy}. a)Time 0.000 ms. b)0.297 ms. c)0.308 ms. d)0.396 ms

10.3 Glass projectile impact

Dynamic impacts present challenging algorithm conditions (contact detection and fracture propagation, among others) to the code's solver, nevertheless have great importance in safety and security engineering problems. In this test, a rectangular piece of glass[150], held by four rectangular supports, as shown in Figure 10.8, is impacted by a projectile with mass equal to 50 g traveling at -3 $^{m}/_{s}$ on y direction. The four supports have their velocities fixed v_x=v_y=v_z=0 for the duration of the simulation.

The projectile is not allowed to brake and is modelled as an elastic material with E=750 GPa and Poisson's ratio ν=0.2. The glass is also modelled as an elastic material with E=75.0 GPa, Poisson's ratio ν=0.2, density ρ=2456 $^{kg}/_{m^3}$, energy release rate G_f =10 $^{N}/_{m}$ and tensile strength f_t =10 MPa. The contact interactions between glass, supports and the projectile are simulated as elastic interactions with penalty $\sigma_p = 7500$

GPa.

Two domain topologies (two and four processes) and three different meshes (coarse, semi-coarse and fine) are used in the test. The coarse mesh consists of 67597 elements in two layers as shown in Figure 10.9a, the semi-coarse mesh with 105120 elements in three layers, as shown in Figure 10.9b and finally the fine mesh consists of 212860 elements in four layers as shown in Figure 10.9c.

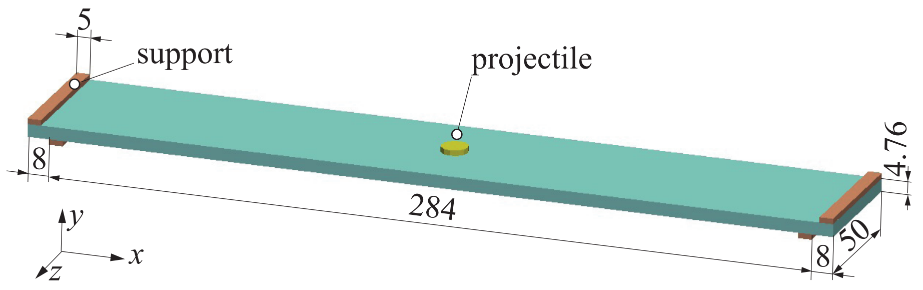


Figure 10.8: Dimensions (mm). Figure adapted from Munjiza et al.[150]

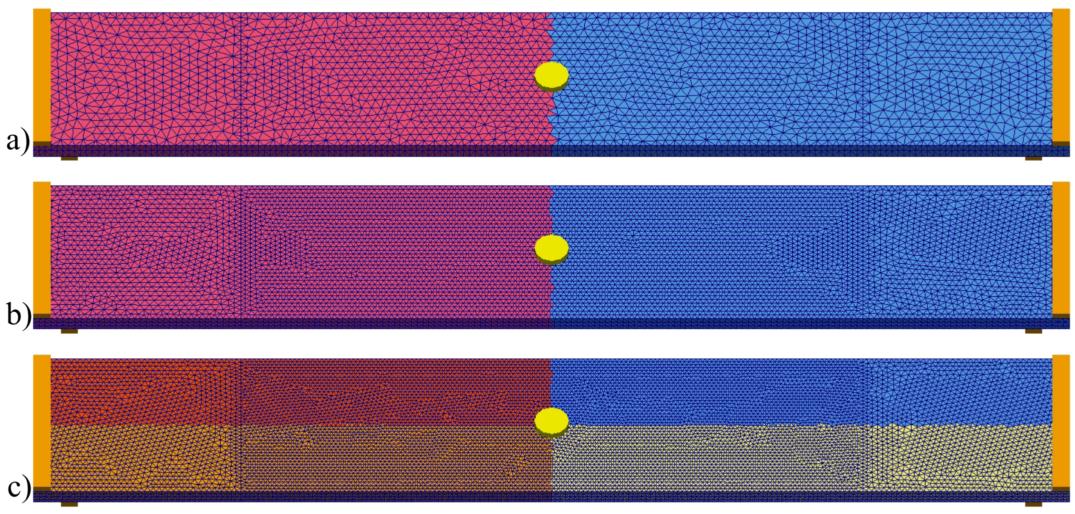


Figure 10.9: Mesh discretisation and domain topology. a)Coarse mesh 67597 FEs. b)Semi-coarse mesh 105120 FEs. c)Fine mesh 212860 FEs.

The total kinetic energy is shown in Figure 10.10 where there is a slow convergence as the number of elements increase. As the kinetic energy from the projectile is transmitted to the glass through contact interaction, the glass starts to fail and the fracture pattern begins to emerge, as shown in Figure 10.11 for $t = 10\,\mu$s. As more energy is passed from the projectile to the glass the fracture grows, as shown in Figure 10.12 for $t = 50\,\mu$s and in Figure 10.13 for $t = 90\,\mu$s . Through the different images it is possible to appreciate the convergence of the fracture patterns among the different combinations of meshes and processes.

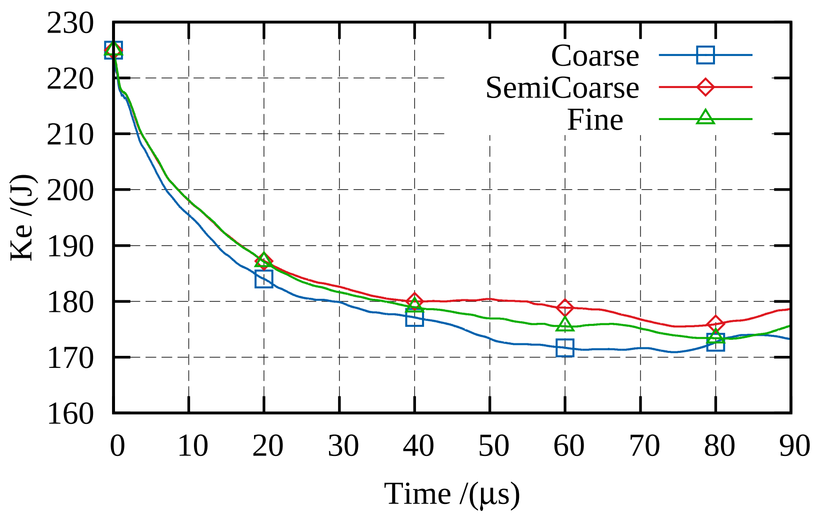


Figure 10.10: Total Ke

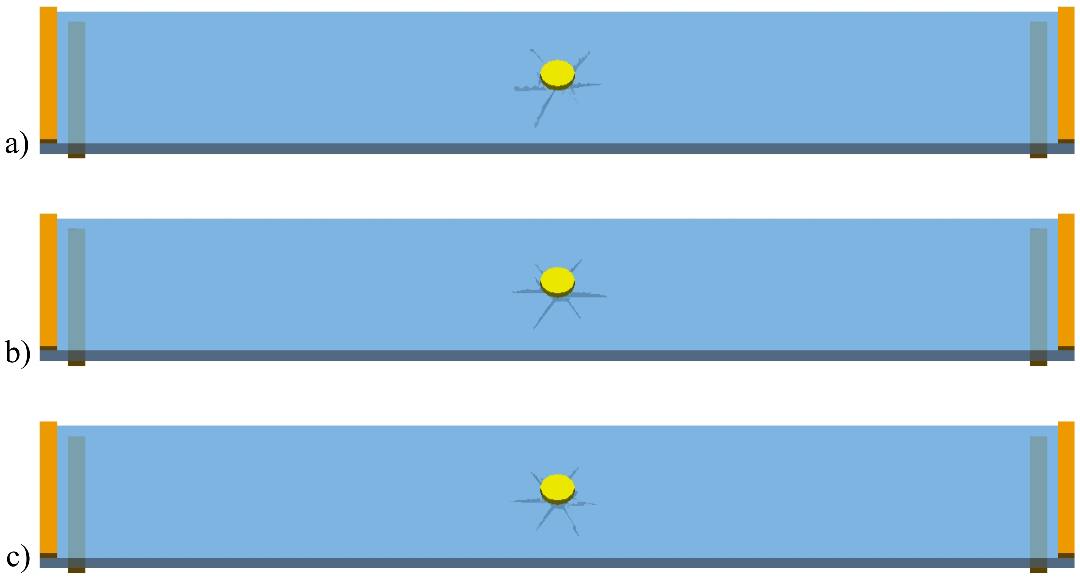


Figure 10.11: Fracture pattern at 10 μs. a)Coarse mesh. b)Semi-coarse mesh. c)Fine mesh.

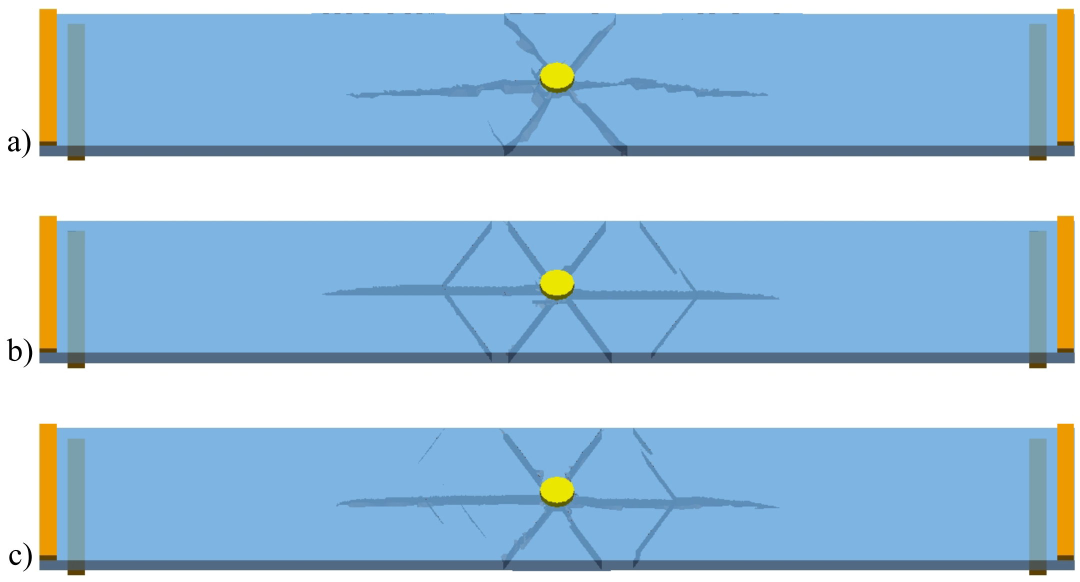


Figure 10.12: Fracture pattern at 50 μs. a)Coarse mesh. b)Semi-coarse mesh. c)Fine mesh.

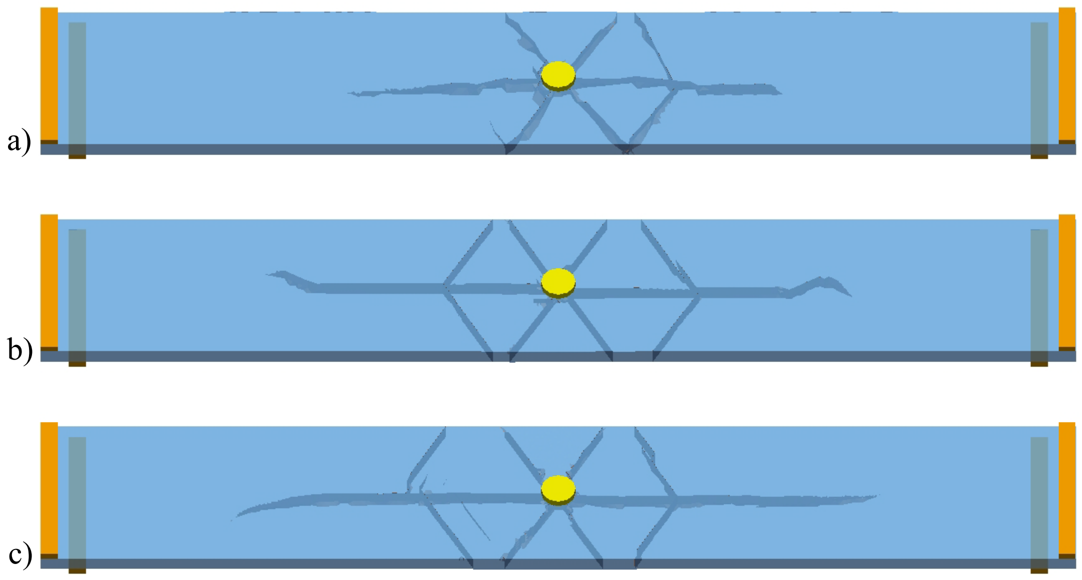


Figure 10.13: Fracture pattern at 90 μs. a)Coarse mesh. b)Semi-coarse mesh. c)Fine mesh.

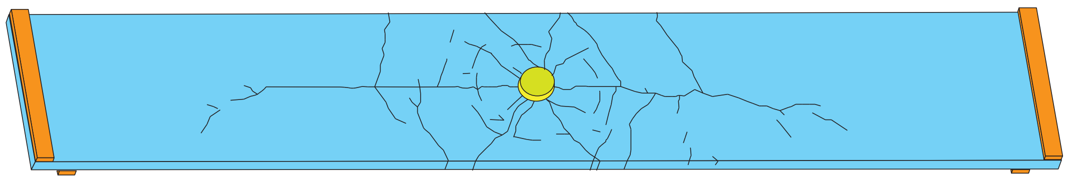


Figure 10.14: Fracture pattern at 90 μs. Figure adapted from Munjiza et al.[150]

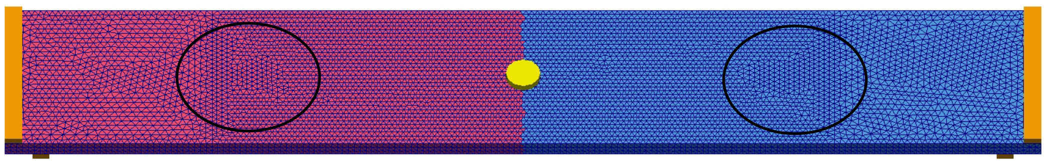


Figure 10.15: Semi-coarse mesh: the abrupt changes in the mesh topology are highlighted with two ellipses.

When the fractures shown in Figure 10.13 are compared to the 2.5D shell fracture pattern obtained by Munjiza et al.[150] shown in Figure 10.14, the main features of the fracture are captured. Nevertheless, the simple model of joints described in Chapter 3 used in the present work is not able to fully capture the radial fracture patterns.

The fracture can only happen on the joints acting between FEs, this means that for coarse meshes the topology of the mesh will greatly influence the fracture pattern. There are chiefly two reasons for this to happen. Firstly, as was shown in the previous example, in coarse meshes the plastic zone on the tip of the fracture is not well captured. The second reason is purely geometric as the topology *guides* the fracture. This is shown in Figure 10.13b where, for the semi-coarse mesh, it is possible to observe that a *hook* like end of the fracture emerged purely because of the mesh discretization as highlighted in Figure 10.15.

10.4 Hyperboloid cooling tower collapse

Even the most efficient fossil-fuel power station wastes 55% of energy in the form of heat that needs to be transferred to the environment.[72] If this excess of heat is transferred into rivers, lakes, etc., the surrounding area may experience a fluctuation in the temperature, in turn affecting the natural life around the power station. On the other hand, a hyperboloid cooling tower is an environmentally energy efficient way to dissipate heat energy as it takes advantage of the use of natural draft to move air from the base to the top. Thus, it decreases the heat foot-print of the area around the power station. Hyperboloid cooling towers are an important part of carbon, natural gas and nuclear energy stations.

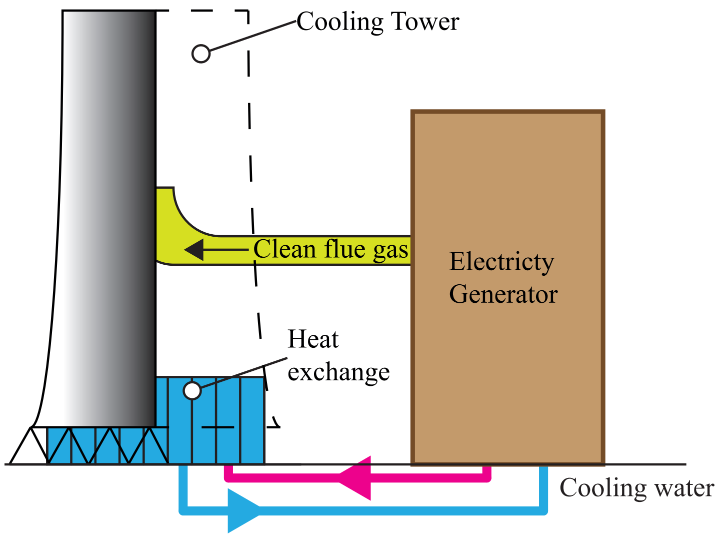


Figure 10.16: Schematic view of a cooling tower with a discharge pipe. Figure adapted from Harte and Kratzing[72]

Environmental requirements in Germany have changed the previous symmetric shape of the hyperboloid towers, as they are used not only for cooling, but for the discharge of gases produced during the combustion of fossil fuels.[72] Fuels have to be filtered before they can be released into the atmosphere. In the process of cleaning there is a decrease in the energy of the gas (pressure drop), therefore making the use of standard chimneys impossible. A schematic view of a cooling tower with a discharge pipe is shown in Figure 10.16.

When a power station has reached the end of its life it has to be decommissioned and its cooling towers demolished. The collapse of the hyperboloid cooling tower is a complex process involving the failure of a composite material called reinforce concrete.[154]

The numerical experiment consists of a hyperboloid cooling tower with 0.5 m wide walls with external dimensions[73] shown in Figure 10.17a. The tower is discretized into 36295 FEs and the domain is broken down into four processes, as shown in Figure 10.17b. The tower lies on a floor with fixed velocities v_x=v_y=v_z=0 during the entire simulation.

The tower's material is simplified as elastic homogenous concrete, made up of the following properties: elastic modulus E=35 GPa,[20] strain energy release rate G_f=147.5 N/m,[176] tensile strength $f_t = 6.3$ MPa,[176] density ρ=2400 $^{kg}/_{m^3}$,[14] and Poisson's ratio ν=0.15.[14]

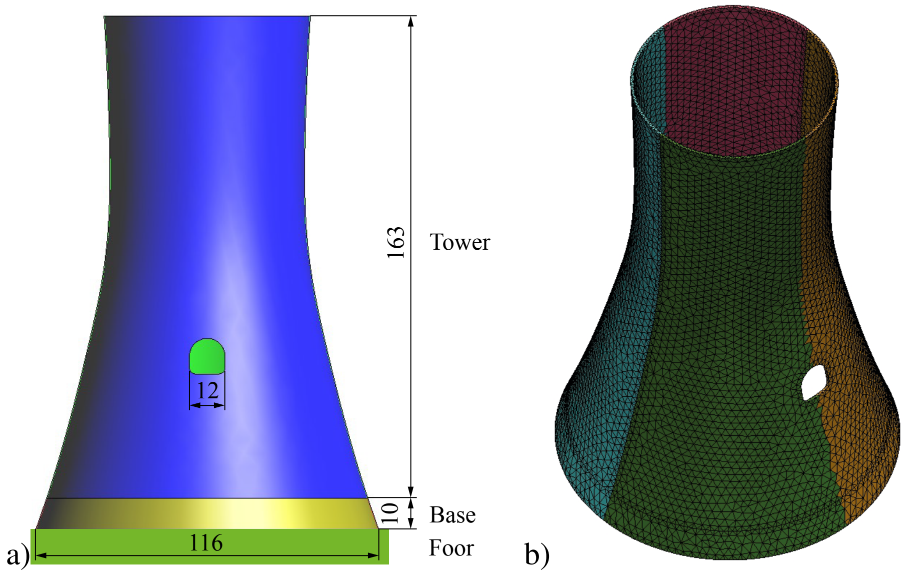


Figure 10.17: Tower geometric description. a)Dimensions (m). b)Mesh and process topology (the floor is not shown).

The interactions between discrete elements are calculated using non elastic normal interactions developed in Chapter 8. The values for the interaction between tower-tower DEs and tower-floor DEs are shown in Table 10.2.

DEs Interaction	Penalty σ_p TPa	Maximum stress σ_T MPa	Exponent b
Tower-Tower	1.8	500	6
Tower-Floor	1.8	5	4

Table 10.2: Non elastic normal interaction properties

The numerical experiment is split into two steps:

1. Gravity load: the tower is first allowed to deform and reach equilibrium with negligible values of kinetic energy under the effect of gravity equal to 9.81$^{m}/_{s^2}$, while the bottom of the front and back base are fixed (v_x=v_y=v_z=0), as shown in Figure 10.18a.

2. Tower collapse: taking the initial conditions from the previous step and maintaining the same value of gravity, two different tests are run:

    - Test A: the front of the base is fractured at t=0.0s and there are no restrictions on the base as shown in Figure 10.18b.
    - Test B: the front of the base is fractured at t=0.0s and only the bottom of the back base is fixed from t=0.0 s until t=0.5 s, as shown in Figure 10.18c. For t>0.5s there are no more restrictions on the base.

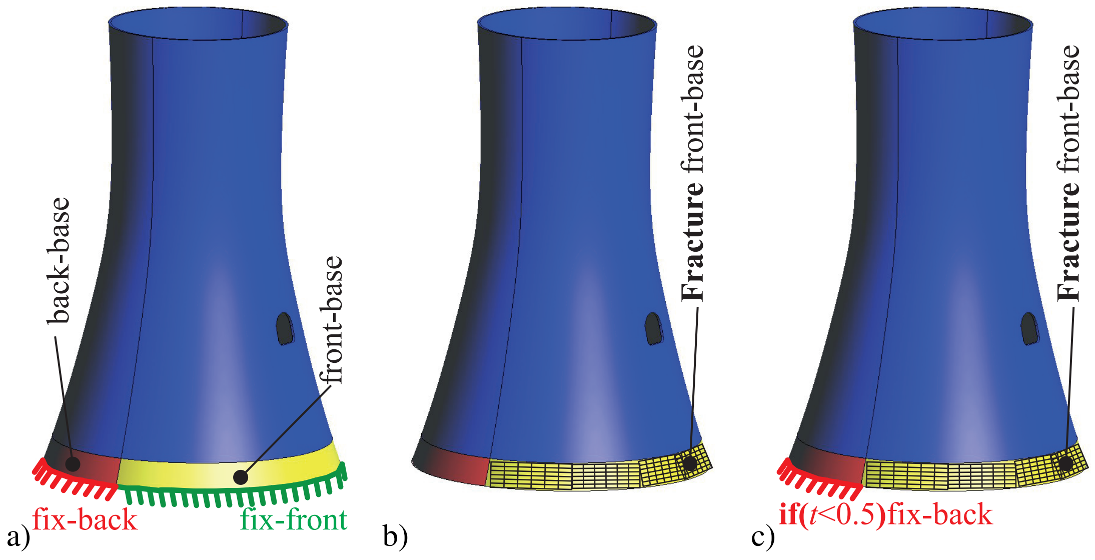


Figure 10.18: Boundary conditions. a)During gravity load. The base of the tower is fixed. b)Test A. The front base is fractured at $t = 0$. c)Test B. The front base is fractured at t=0, while he back base is fixed for $t < 0.5$.

The tower collapse sequence for Test A and Test B are shown in Figure 10.21, Figure 10.22, Figure 10.23 and Figure 10.24. Upon scrutiny, it is possible to appreciate the different dynamics between the tests and how complex fracture patterns started and grew crossing boundaries between different processes.

The kinetic energy for Test A and Test B is shown in Figure 10.19, where there is a shift in time in the kinetic energy produced by the boundary conditions on Test B. After the towers collapsed, as shown in Figure 10.24a for Test A and in Figure 10.24b for test B, there is a significant change in the slope of the kinetic energy in Figure 10.19, where the rate of dissipation of kinetic energy decreases. Mainly, this is due to the absence of new fractures as the only dissipation mechanism left is the damping in the material and normal interactions: tower-tower and tower-floor. Simultaneously, the decrease in vertical velocities implies that the relative importance of interactions tower-floor (which dissipates more energy see table 10.2) decrease as well.

The material properties employed in these tests corresponded to those of a good quality concrete. While this cannot be used to realistically simulate the collapse of a cooling tower it is a good initial step of comparable relevance, as it shows the capacity of the different algorithms developed in this work (contact detection, non elastic interactions and parallel solutions) to deal with complex, dynamic boundary changes and fracture patterns across different processes. However, further research must still be carried out to take into account the properties of reinforced concrete regarding the constitutive law of the material(Chapter 3) and the joints (Chapter 3).

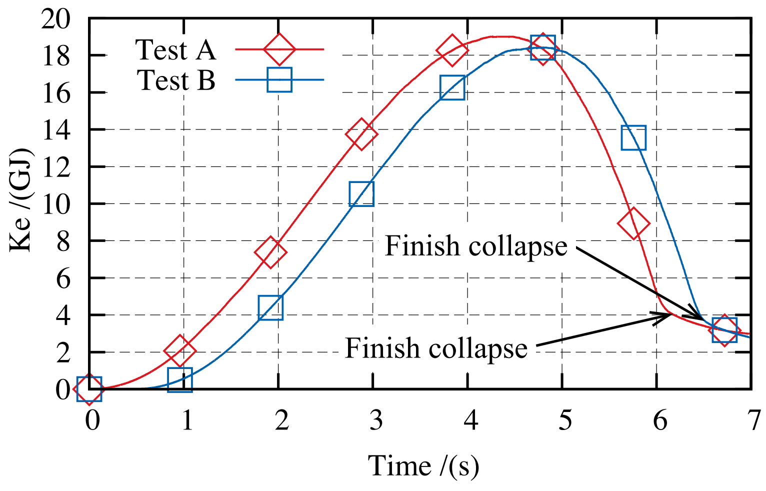


Figure 10.19: Tower Ke.

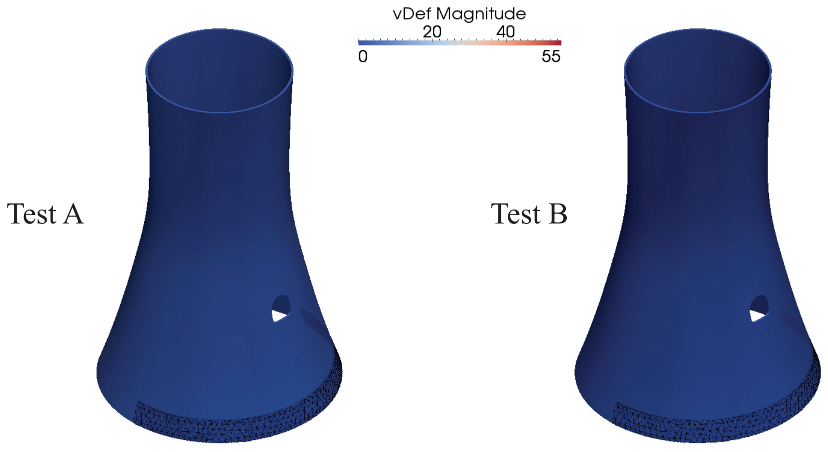
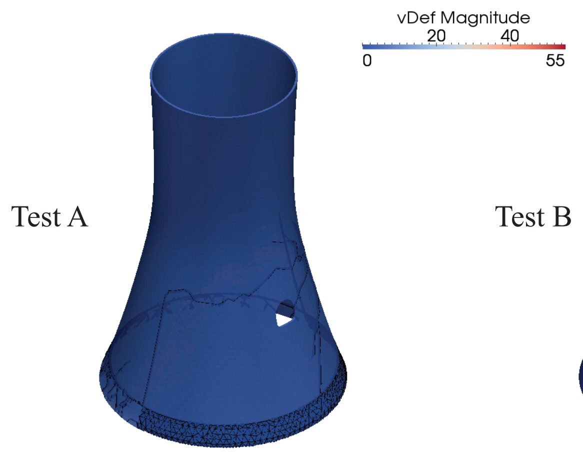
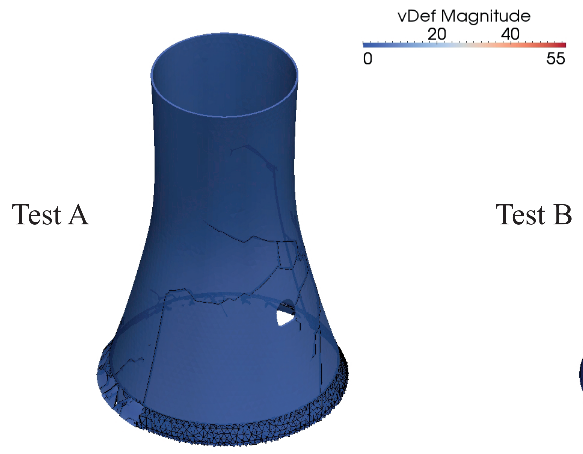


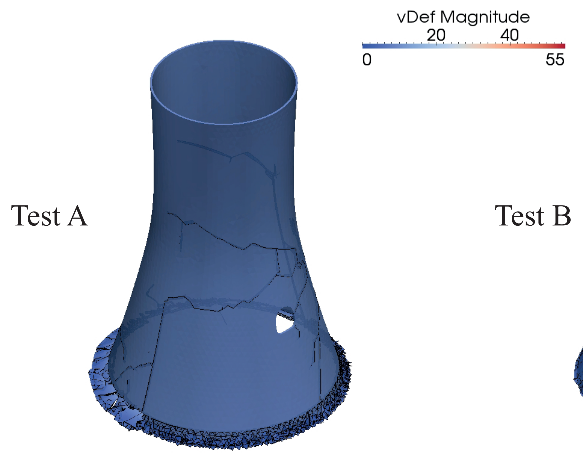
Figure 10.20: Tower collapse sequence $t = 0$ s.



a)

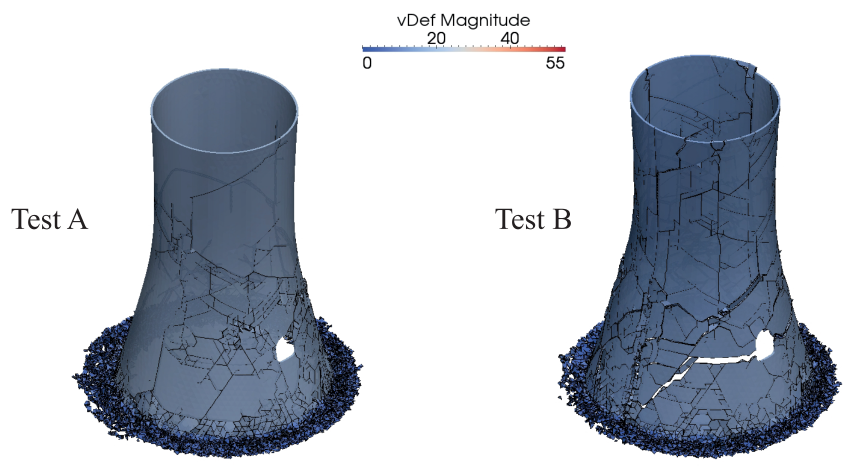


b)

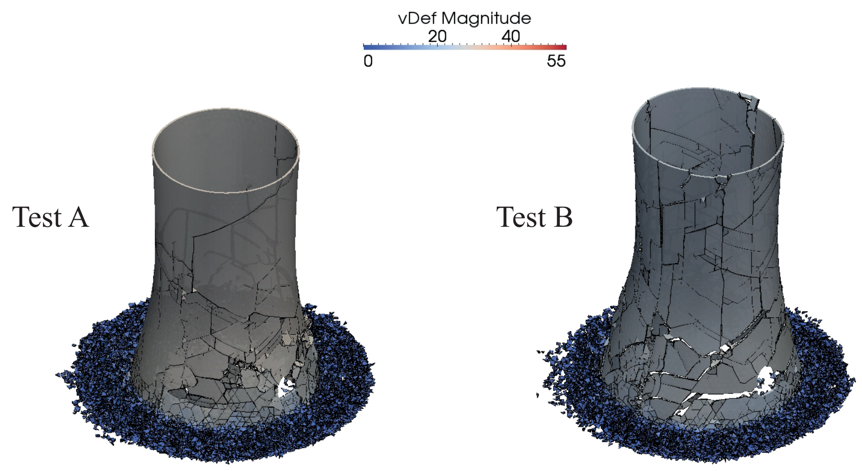


c)

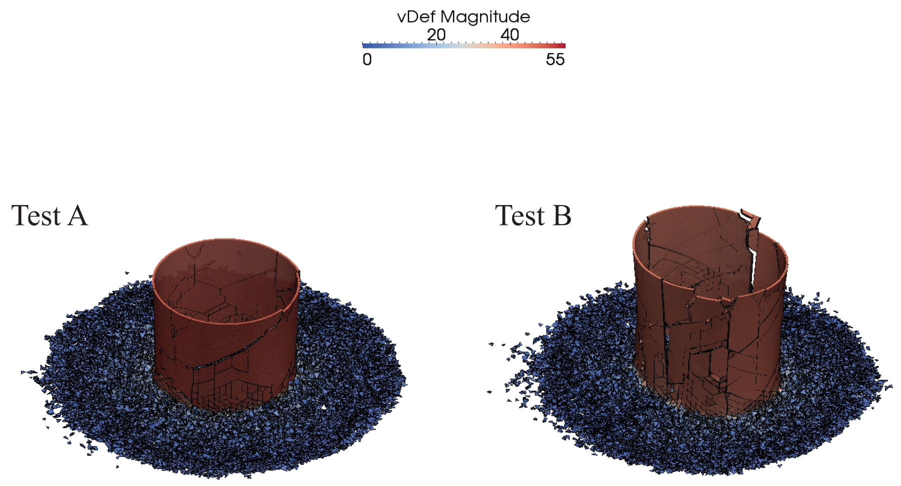
Figure 10.21: Tower collapse sequence. a)$t = 0.2460$ s. b)t=0.4928 s. c)t =0.9952 s.



a)

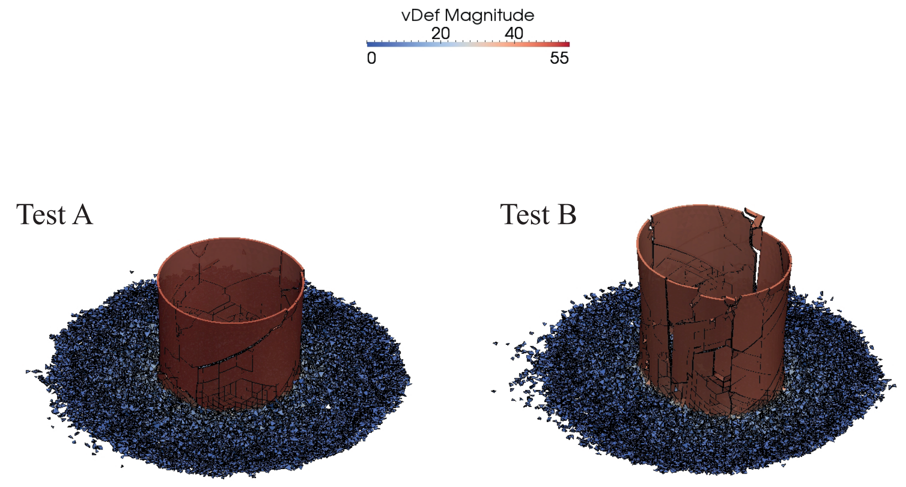


b)

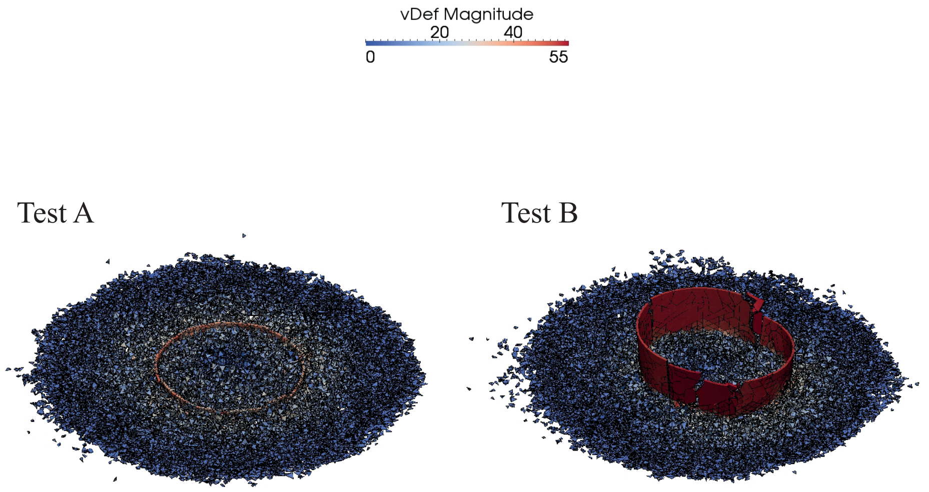


c)

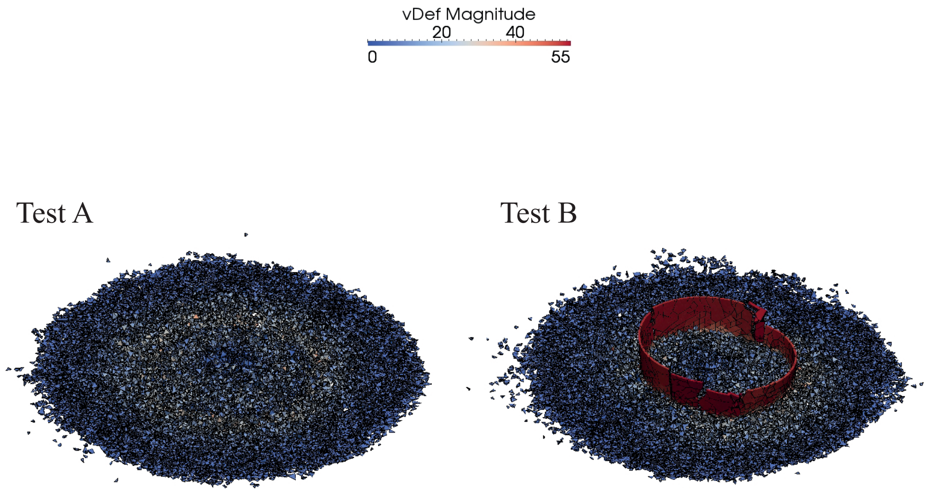
Figure 10.22: Tower collapse sequence. a)t =1.9936 s. b)t=2.9952 s. c)t =3.9936 s.



a)

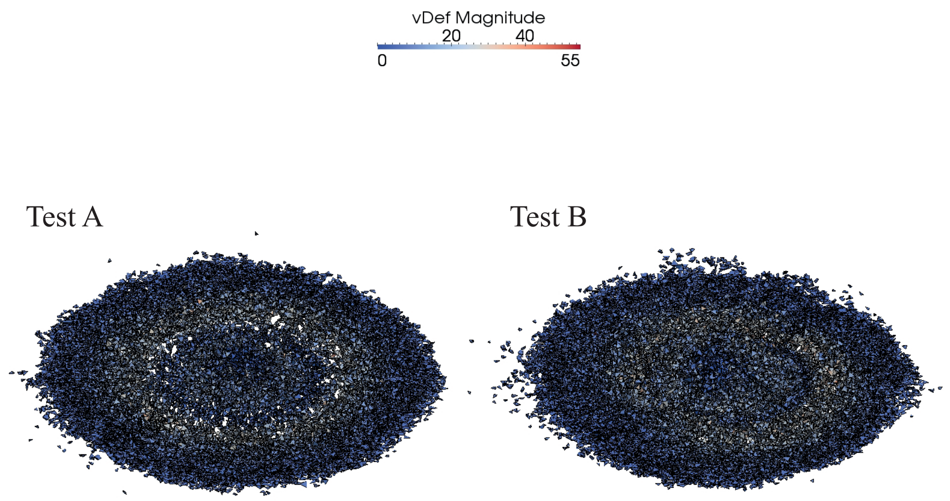


b)

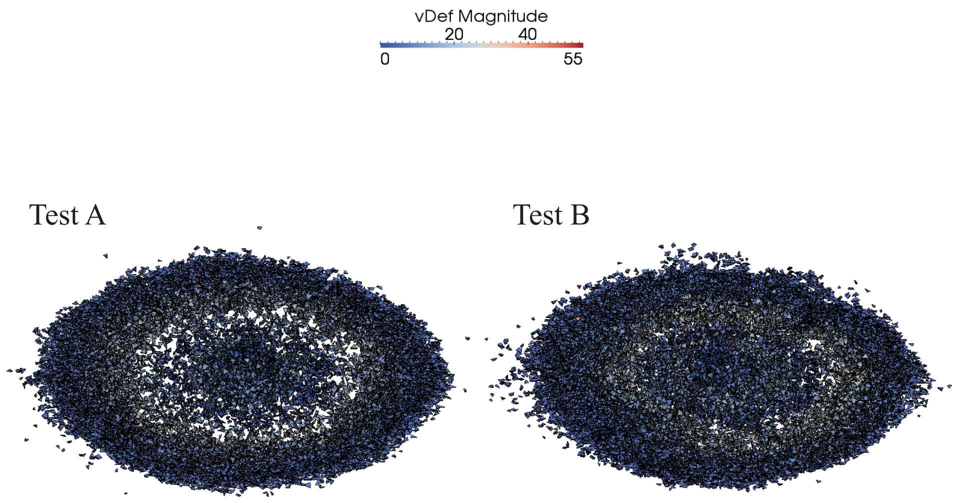


c)

Figure 10.23: Tower collapse sequence. a)t =4.9952 s. b)t=5.9936 s. c)t =6.0909 s.



a)



b)

Figure 10.24: Tower collapse sequence. a)t =6.4960 s. b)t=6.7904 s.

Chapter 11

CONCLUSIONS AND FURTHER RESEARCH

New algorithmic solutions for the Combined Finite-Discrete Element Method have been developed and tested. In this chapter a summary of this work is presented, and some possible further research directions are suggested.

Summary. In Chapter 2 some limitations for the classic continuum methods such as FEM were presented with the methods of Discontinua to overcome them. A compact historical background of the different methods of discontinua was introduced, as well as short descriptions of the following :

- DEM
- FDEM
- DDA
- MD
- SPH

Chapter 3 featured the introduction of different aspects of the FDEM. The radical concept in FDEM of *glued* deformable particles (finite elements) through joints was introduced here. The equations of motion to be applied to each finite element (FE) taking into account deformations, interaction and joints were presented. Some aspects of contact detection such as RAM memory limitation, and contacting couples were described. The equations for small strains to calculate the deformation forces on a tetrahedra were introduced here. Complex fracture processes (Mode I, Mode II, Mode III) in the joint using the stress-strain model developed by Munjiza, were presented and tested. Finally a literature review of different hardware, software and libraries for parallelization, with some important concepts such as speed-up, Amdahl's law and rounding error were described.

In chapter 4, a novel CD algorithm for bodies of similar size was explained in detail. This algorithm denominated Balance Binary Tree Schiava (BBTS) is based on the Binary Tree (BT), the spatial ordering criterion, and the contact mask in 3D. This algorithm is suited to be used in any object oriented language (C++, Java, etc.,) and should be easy to implement into any existing BT.

Chapter 5, presented a flavour of the MR algorithm denominated MR-Schiava that does not use objects as the MR algorithm but rather makes use of arrays, and is so described. This makes it suitable to be implemented in non-object oriented solutions. A description of the Non Binary Search-Munjiza (NBS) and the MR were also presented.

Chapter 6 featured a novel algorithm based on the BBTS and the MR algorithm for body-point interaction. The modifications on the BBTS-tree structure, and the new search algorithm were described in detail. This new algorithm was denominated MunjizaSchiava. A test of up to 1.331 M discrete elements (DEs) with 5.324 M interaction points was presented for sequential (one process) and parallel 2 and 4 processes. The algorithm showed some non-linear behaviour as expected, yet in the range tested there was a pseudo-linear behaviour. At the end of this chapter, a further improvement of the algorithm was presented.

In chapter 7, detailed comparisons of different algorithms for bodies of similar size were presented. The testing carried out on MR, MR-S, NBS and BBTS were described in detail. These tests were only carried out sequentially, with ghost DEs that were not able to deform and for which their positions were imposed at each time step. For bodies of similar size the fastest algorithms were the MR and MR-S.

A model previously developed in 2D and implemented in a commercial software PFC2D and later ported to the open source 2D-Y code was presented in chapter 8. The original algorithm was modified to avoid jumps in the interaction forces and implemented in 3D. The improved performance was shown in two tests at the end of the chapter.

Chapter 9 described a novel parallel solution in 3D for the FDEM based on the solutions developed in the present work. The Message Passing Interface (MPI) library was also described. Three different tests to evaluate the performance of the proposed solution using complex DEs (made of more than one FE) were presented. The speed ups were near the expected values of 2 for two processes, and 4 for four processes.

Some non-linearities attributed to the non-linear MS-CD algorithm were observed in the first two tests, while in the last test the behaviour of the algorithm in the absence of CD (CD time was almost zero) was shown with speed ups of almost 2 and 4 for two and four processes.

The following applications and examples were presented in chapter 10:

- The first test was the well-known cantilever beam under distributed load, where the performance of the algorithm was compared with the theoretical solution. As expected, as the number of finite elements increased the final displacement approached the theoretical value.

- The second test involved 3D fracture propagation on 4 processes. The performance of fracture was compared with previously published results in 2D showing a good agreement between the results.

- The third test involved the interaction between a projectile and a plain glass. Here the fracture patterns were compared to published numerical results of thin shell elements using FDEM. The algorithm developed in this work was able to capture the main features of the fracture patterns in the glass however some of the fractures near the impact point were not captured.

- The last example simulated the demolition of a Hyperboloid Cooling Tower. While there is no published data to compare with, this example showed the capacity of the parallel algorithm to produce complex fracture patterns. The material properties corresponded to the ones of a good quality concrete. While this cannot be used to simulate reinforced concrete, it is a good first step of comparable relevance.

Conclusions. The new algorithms developed in this work (Chapters 6, 8, 9) were tested under static and dynamic conditions in sequential and parallel simulations. Complex fracture structures emerged without any input from the user. The solutions developed in this work, can be used to simulate elastic materials subject to fracture processes without the presence of fluids on off-the-shelf Multicore PC's taking advantage of all the CPU power available.

Suggested further research directions. In the remainder of the chapter, some suggestions for further research are presented.

Fracture processes are still an open area of research. The single and smeared crack model (Chapter 3) implemented present limitations, as the numerical integration of the forces and crack opening are integrated on all the surfaces of the FEs.[135] The relationship between micro-fracture processes and macro-scales can be simulated using multiscale methods.[83] Multiscale methods are CPU intensive when compared with the implemented fracture algorithm. An intermediate solution may lie in the use of the single and smeared crack models in the entire domain, with the multiscale fracture method reserved for particular areas of interest.

The present work was only tested on Multicore PCs (hardware available during this work), on WindowsOS. As the amount of RAM memory and CPU power is limited on Multicore PCs, there is a need to port the present solution to LinuxOS environment to further test and develop DEM solutions on High Performance Computing (HPC).

The static domain topology implemented (Chapter 9) is only well suited for static, and semi-static simulations. In simulations of dynamic systems, discrete objects move relatively fast, migrating from one process to another, producing some overloaded processes, leaving the less loaded processes waiting idly between communications. A dynamic domain topology should be implemented to improve the performance of the solution. Lukas[112, 114, 113] utilized a modified version of the Recurse Coordinate Bisection[185] (RCB) on the 2D parallel FDEM Y-Code. The modified RCB algorithm gradually modifies the topology of the domain, updating it to the current configuration of the simulation, keeping a semi-structured topology.[114] Different law balancing can be used to modify the domain topology, the number of contact interactions, the number of FEs and the number of fluid cells incase of fluid/FDEM interaction, among others.

Currently, fluid driven fracture is of great importance as the search for new sources of energy, such as shale gas, is pushing the development of new numerical solutions. Shale gas is an unconventional gas extracted by the fracturing soil using fluids. As the prices of other conventional sources of energy increase, interest in this new source of gas is gaining momentum.[193]

Several different methods have been applied in the past to simulate fluid driven fracture.[6] Munjiza utilized a simple flow model coupled with FDEM to simulate 2D boreholes.[133, 143] Zhou and Hou[210] coupled a full Navier Stokes solver with a solid solver (FLCAC3D-Itasca) to simulate borehole fracture processes with good agreement between the numerical and empirical test. While classical fluid solid interaction using finite volume (FVM) and finite elements (FEM) methods have demonstrated their utility, parallelization of the methods presents difficult challenges, as the system of equations to solve is fragmented in different processes. On the other hand Lattice

Boltzmann (LB) is a relatively new method to solve the Navier Stokes equations that is gaining momentum in the scientific community. LB is a method based on the solution of the Boltzmann equations on a predefined lattice,[186] with the advantage that the equations to be solved are localised. There is no need to build large systems of equations (to be shared across processes) that later have to be solved as is the case with FVM and FEM, making LB the ideal solver in parallelization terms. LB has been ported to HPC and GPU parallel hardwares with linear speed up velocities.

In recent years LB have been coupled with DE, Feng et al.[46,70] coupled LB with simple DE using the Immersed Boundary Method (IBM) switching on and off cells on the LB discretization. Lomine et al.[111] applied a similar method switching on and off nodes on the lattice but applying a simple Bounce Back boundary condition to simulate piping erosion.

The possibilities of combining Lattice Boltzmann and FDEM techniques are promising, and since both methods solve their equations locally merging both onto the same parallel code is the next logical step, solving FDEM on the CPU and LB on a CUDA processor (GPU).[122,173]

-

References

[1] *International Technology Roadmap for Semiconductors 2010 Update Overview*. http://www.itrs.net/Links/2010ITRS/Home2010.htm. Version: 2010. – Accessed 29 Nov 2011

[2] *MACHAR - Dynamically Compute Machine Constants*. http://orion.math.iastate.edu/burkardt/c_src/machar/machar.html. Version: 2011. – Acceded on 20 January 2012 at 18:30

[3] *MPI-RaspberryPi*. http://www.southampton.ac.uk/~sjc/raspberrypi/. Version: Sep 2012. – Acceded on 20 Sep 2012

[4] *Ultimate Silicon Valley Perk: Custom Chips From Intel and AMD*. http://www.wired.com/wiredenterprise/2012/09/intel-amd-custom-chips/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+wired%2Findex+%28Wired%3A+Top+Stories%29. Version: 09 2012. – Acceded on 20 Sep 2012

[5] ABOU-CHAKRA, H. ; BAXTER, J. ; TÜZÜN, U. : Three-dimensional particle shape descriptors for computer simulation of non-spherical particulate assemblies. In: *Advanced Powder Technology* 15 (2004), Nr. 1, S. 63 – 77. – ISSN 0921–8831

[6] ADACHI, J. ; SIEBRITS, E. ; PEIRCE, A. ; DESCROCHES, J. : Computer simualtion of hydraulic fractures. In: *International Journal of Rock Mechanics and Mining Sciences* 44 (2007), S. 739–757

[7] AKKER, P. V. D.: *Particle-based Evaporation Models and Wall Interaction for Microchannel Cooling*, Technische Universiteit Eindhoven, Diss., 2010

[8] AL-MOHSSEN, H. A.: *An Excursion with the Boltzmann Equation at Low Speeds: Variance-Reduced DSMC*, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, Diss., 2010

[9] AN, B. ; TANNANT, D. D.: Discrete element method contact model for inelastic rock impact. In: *Computer & Geosciences* 33 (2007), S. 513–521

[10] ARM: *Hubunto Server Runs On ARM*. `http://blog.canonical.com/2011/11/02/hpmoonshot/`. Version: December 2011. – Acceded on 10th December 2011

[11] ASHMAWY, A. K. ; SUKUMARAN, B. ; HOANG, V. : Evaluating the Influence of Particle Shape on Liquefaction Behavior Using Discrete Element Modeling. In: *International Offshore and Polar Engineering Conference*, 2003. – ISBN 1 880653 60 5

[12] BAKUN-MAZOR, D. ; HATZOR, Y. H. ; DERSHOWITZ, W. S.: Modeling mechanical layering effects on stability of underground openings in jointed sedimentary rocks. In: *International Journal of Rock Mechanics and Mining Sciences* 46 (2009), S. 262 – 271

[13] BANGASH, T. ; MUNJIZA, A. : Experimental validation of a computationally efficient beam element for combined finite discrete element modelling of structures in distress. In: *Computational Mechanics* 30 (2003), S. 366–373

[14] BENHAM, P. P. ; CRAWFORD, R. J. ; ARMSTRONG, C. G.: *Mechanics of Engineering Material*. Addison Wesley Longman Limited, 1997

[15] BERGMAN, K. ; BORKAR, S. ; CAMPBELL, D. ; CARLSON, W. ; DALLY, W. ; DENNEAU, M. ; FRANZON, P. ; HARROD, W. ; HILLER, J. ; KARP, S. ; KECKLER, S. ; KLEIN, D. ; LUCAS, R. ; RICHARDS, M. ; SCARPELLI, A. ; SCOTT, S. ; SNAVELY, A. ; STERLING, T. ; WILLIAMS, R. S. ; YELICK, K. ; BERGMAN, K. ; BORKAR, S. ; CAMPBELL, D. ; CARLSON, W. ; DALLY, W. ; DENNEAU, M. ; FRANZON, P. ; HARROD, W. ; HILLER, J. ; KECKLER, S. ; KLEIN, D. ; KOGGE, P. ; WILLIAMS, R. S. ; YELICK, K. : *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems Peter Kogge, Editor Study Lead*. `http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf`. Version: 2008. – Accessed on 29 Nov 2011 at 20:00

[16] BEYABANAKI, S. A. R. ; MIKOLA, R. G. ; HATAMI, K. : Three-dimensional discontinuous deformation analysis (3-D DDA) using a new contact resolution algorithm. In: *Computers and Geotechnics* 35 (2008), S. 346 – 356

[17] BOLTON, W. : *Mechanical Science*. Blackwell Scientific Publications, 1993

[18] BONET, J. ; PERAIRE, J. : An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems. In: *, Int. J. Numer. Meth. Eng* 31 (1991), S. 1–17

[19] BORKAR, S. ; CHIEN, A. : The Future of Microprocessors. In: *Communications of the ACM* 5 (2011), S. 67–77

[20] BRARA, A. ; CAMBORDE, F. ; KLEPACZKO, J. ; MARIOTTI, C. : Experimental and numerical study of concrete at high strain rates in tension. In: *Mechanics of Materials* 33 (2000), S. 33–45

[21] CAMPBELL, C. S.: Rapid granular flows. In: *Annual Review of Fluid Mechanics* 22, S. –92

[22] CAMPBELL, C. S.: Granular material flows An overview. In: *Powder Technology* 162 (2006), S. 208–229

[23] CAMPBELL, C. S. ; BRENNEN, C. E.: Computer simulations of granular sher flows. In: *Journal of Fluid Mechanics* 151 (1985), S. 167–188

[24] CASTER, B. R. ; HOWES, L. ; KAELI, D. ; MISTRY, P. ; SCHAA, D. : *Heterogeneous Computing with OpenCL.* Elsevier, 2011

[25] CHANDRA, R. ; DAGUM, L. ; KOHR, D. ; MAYDAN, D. ; MCDONALD, J. ; MENON, R. : *Parallel programming in OpenMP.* San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2001. – ISBN 1–55860–671–8

[26] CHATIRI, M. ; SCHUTZ, T. ; MATZENMILLER, A. : An Assessment of the New LS-DYNA Multi-Layeded Solid Element: Basicss, Patch Simulation and its Potential for Thick Composite Structural Analysis. In: *11th International LS-Dyna Users Conference*, 2010

[27] CHIEN, A. A. ; SNAVELY, A. ; GAHAGAN, M. : 10x10: A General-purpose Architectural Approach to Heterogeneity and Energy Efficiency. In: *Procedia CS* (2011), S. 1987–1996

[28] CLEARY, P. W.: DEM simulation of industrial particle flows: case studies of dragline excavators, mixing in tumblers and centrifugal mills. In: *Powder Technology* 109, Nr. 2-4, S. 83–104

[29] CLEARY, P. W.: The effect of particle shape on simple shear flows. In: *Powder Technology* 179 (2008), S. 144–163

[30] CLEARY, P. W.: DEM prediction of industrial and geophysical particle flows. In: *Particuology* 8 (2010), Nr. 2, S. 106–118

[31] CLEARY, P. W. ; SINNOTT, M. ; MORRISON, R. : Prediction of slurry transport in SAG mills using SPH fluid flow in a dynamic DEM based porous media. In: *Minerals Engineering* 19 (2006), Nr. 15, S. 1517 – 1527

[32] CLOUGH, R. W.: The finite element method in plane stress analysis. In: *Proceedings 2nd ASCE Conference on Electronic Computation (Pittsburgh, PA).*, 1960, S. 345–378.

[33] COLLINGS, J. A.: *Failure of Materials in Mechanical Design*. John Wiley Sons, Ltd., 1993

[34] CUNDALL, P. A.: A computer model for simulating progressive large scale movements in block rock systems. In: *Proceedings of the symposium of international society of rock me- chanics*, 1971

[35] CUNDALL, P. A. ; STRACK, O. D. L.: A discrete numerical model for granular assemblies. In: *Geotechnique* 29 (1979), Nr. 1, S. 47–65

[36] CUNDALL, P. A. P. A. A. P. A. ; HART, R. D.: Numerical Modelling of Discontinua. In: *Engineering Computations* 9 (1992), S. 101–113

[37] DAS, N. : *Modeling three-dimensional shape of sand grains using Discrete Element Method*, University of South Florida Scholar Commons, Diss., 2007

[38] DELANEY, G. W. ; CLEARY, P. W. ; SINNOTT, M. D. ; MORRISON, R. D.: Novel application of DEM to modelling comminution processes. In: *IOP Conference Series: Materials Science and Engineering* 10 (2010), Nr. 1, S. 012–099

[39] DJINEVSKI, L. ; MISHKOVSKI, I. ; TRAJANOV, D. : Accelerating Clustering Coefficient Calculations on a GPU Using OPENCL. In: *ICT Innovations 2010* (2011), S. 276–285

[40] DZIUGYS, A. ; PETERS, B. : An approach to simulate the motion of spherical and non-spherical fuel particles in combustion chambers. In: *Granular Matter* 3 (2001), Nr. 4, S. 231–266

[41] EIJKHOUT, V. : *Introduction to High Performance Scientific Computing*. http://tacc-web.austin.utexas.edu/staff/home/veijkhout/public_html/istc/istc.html. Version: Dec 2011. – Acceded on 11 Dec 2011

[42] EL-REWINI, H. ; ABD-EL-BARR, M. ; ZOMAYA, A. (Hrsg.): *Advanced Computer Architecture and Parallel Processing*. Wiley, 2005

[43] FAN, L. S. ; TAI, Y. ; MULLER, R. S.: IC- processed electrostatic micromotors. In: *Tech. Dig. Int. Electron Dev. Meet. (IEDM)*, 1988

[44] FARBER, R. : *Numerical Precision: How Much is Enough?* http://www.scientificcomputing.com/article-hpc-Numerical-Precision-How-Much-is-Enough-063009.aspx#. Version: Aug 2012. – Accessed 8 Aug 2012 at 22:01

[45] FELDMAN, J. ; BONET, J. : Dynamic refinement and boundary contact forces in SPH with applications in fluid flow problems. In: *International Journal for Numerical Methods in Engineering* 72 (2007), Nr. 3, S. 295–324. – ISSN 1097–0207

[46] FENG, Y. T. ; HAN, K. ; OWEN, D. R. J.: Int. J. Numer. Meth. Eng. In: *Coupled lattice Boltzmann method ad discrete element modelling of particle transport in turbulent fluid flows: Computational Issues* 72 (2007), S. 1111–1134

[47] FENG, Y. T. ; OWEN, D. R. J.: An augmented spatial digital tree algorithm for contact detection in computational mechanics. In: *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING* 55 (2002), S. 159–176

[48] FERELLEC, J.-F. ; MCDOWELL, G. : A method to model realistic particle shape and inertia in DEM. In: *Granular Matter* 12 (2010), S. 459–467

[49] FERREZ, J.-A. ; LIEBLING, T. : Parallel DEM Simulations of Granular Materials. In: HERTZBERGER, B. (Hrsg.) ; HOEKSTRA, A. (Hrsg.) ; WILLIAMS, R. (Hrsg.): *High-Performance Computing and Networking* Bd. 2110. Springer Berlin / Heidelberg, 2001, S. 211–220

[50] FORUM, M. P. I.: *MPI: A Message-Passing Interface Standard Version 3.0*. http://www.mpi-forum.org/docs/docs.html. Version: September 2012. – Acceced 28 Feb 2013

[51] FRAIGE, F. Y. ; LANGSTON, P. A. ; MATCHETT, A. J. ; DODDS, J. : Vibration induced flow in hoppers: DEM 2D polygon model. In: *Particuology* 6 (2008), S. 455 – 466

[52] FRENKEL, D. ; SMIT, B. : *Understanding Molecular Simulation From Algorithms to Applications*. Academic Press, 2002

[53] FUJUN, W. ; FENG, Y. T. ; OWEN, D. ; JING, Z. ; YANG, L. : Parallel analysis of combined finite-discrete element systems on PC cluster. In: *Acta Mechanica Sinica* 20 (2004), S. 534–540

[54] GARCIA, X. ; AKANJI, L. T. ; BLUNT, M. J. ; MATTAHAI, S. K. ; LATHAM, J. P.: Numerical study of the effects of particle shape and polydispersity on permeability. In: *Physical Review* 80 (2009), S. 47–56

[55] GDOUTOS, E. E. ; GLADWELL, G. M. L. (Hrsg.): *Fracture Mechanics An Introduction*. Springer, 2005

[56] GETHIN, D. ; YANG, X. ; LEWIS, R. : A two dimensional combined discrete and finite element scheme for simulating the flow and compaction of systems comprising irregular particulates. In: *Computer Methods in Applied Mechanics and Engineering* 195 (2006), Nr. 41-43, S. 5552–5565

[57] GOLDHABER-GORDON, D. ; MONTEMERLO, M. S. ; CHRISTOPHER LOVE, J. ; GREGORY, J. O. ; ELLENBOGEN, J. C.: Overview of Nanoelectronic Devices. In: *PROCEEDINGS OF THE IEEE* 4 (1997), S. 521–540

[58] GOLDHIRSCH, I. : RAPID GRANULAR FLOWS. In: *Annual Review of Fluid Mechanics* 35 (2003), S. 267–293

[59] In: GREWE, D. ; OBOYLE, M. : *A Static Task Partitioning Approach for Heterogeneous Systems Using OpenCL*. Bd. 6601. Springer Berlin Heidelberg, 286–305

[60] GRIEBEL, M. ; KNAPEK, S. ; ZUMBUSCH, G. ; TIMOTHY J. BARTH, D. E. R. M. D. R. T. S. Michael Griebel G. Michael Griebel (Hrsg.): *Numerical Simulation in Molecular Dynamics*. Springer, 2007

[61] GROPP, W. ; LUSK, E. ; THAKUR, R. : *Using MPI-2 Advanced Features of the Message-Passing Interface*. Cambridge, MA : MIT Press, 1999

[62] GRUBER, G. ; KLEIN, D. ; WARTZACK, S. : *A modified approach for simulating complex compound structures within early desing steps*. http://www.dynalook.com/8th-european-ls-dyna-conference/session-8/Session8_Paper3.pdf. Version: May 2011. – Accessed 15 Sep 2012

[63] GU, J. ; ZHAO, Z. : Considerations of the discontinuous deformation analysis on wave propagation problems. In: *International Journal for Numerical and Analytical Methods in Geomechanics* 33 (2009), Nr. 12, S. 1449–1465

[64] GUISES, R. : *Numerical Simulation and Characterisation of the Packing of Granular Material*, Imperial College London, Diss., 2008

[65] GUISES, R. ; XIANG, J. ; LATHAM, J. P. ; MUNJIZA, A. : Granular packing: numerical simulation and the characterisation of the effect of particle shape. In: *Granular Matter* 11 (2009), S. 281–292

[66] HAFF, P. K. ; WERNER, B. T.: Computer simulation of the mechanical sorting of grains. In: *Powder Technology* 48 (1986), S. 239–245

[67] HAILE, J. M.: *Molecular Dynamics Simulation Elementary Methods*. JOHN WILEY SONS, INC., 1992

[68] HAK, M. el: The fluid mechanics of microdevice The Freeman Scholar Lecture. In: *J. Fluids Eng* 121 (1999), S. 5–33

[69] HAK, M. G. (Hrsg.): *MEMS Introduction and Fundamentals*. CRC Press, 2006

[70] HAN, K. ; FENG, Y. T. ; OWEN, D. : Coupled lattice Boltzmann and discrete element modelling of fluid-particle interaction problems. In: *Computer and Structures* 85 (2007), S. 1080–1088

[71] HARIRI, S. (Hrsg.) ; PARASHAR, M. (Hrsg.): *Tools and Environments for Parallel and Distributed Computing*. Wiley, 2004

[72] HARTE, R. ; KRATZIG, W. : Large-scale cooling towers as part of an efficient and cleaner energy generating technology. In: *Thin Walled Structures* 40 (2002), S. 651–664

[73] HARTE, R. ; WITTEK, U. : Recent developments of cooling tower desing. In: *Proceedings of the International Association for Shell and Spatial Structures*, 2009, S. 198–210

[74] HATZOR, Y. ; ARZI, A. ; ZASLAVSKY, Y. ; SHAPIRA, A. : Dynamic stability analysis of jointed rock slopes using the DDA method: King Herod's Palace, Masada, Israel. In: *International Journal of Rock Mechanics and Mining Sciences* 41 (2004), S. 813 – 832

[75] HERBST, J. A. ; POTAPOV, A. V.: Making a Discrete Grain Breakage model practical for comminution equipment performance simulation. In: *Powder Technology* 143-144 (2004), S. 144 – 150

[76] HOLMES, D. ; WILLIAMS, J. R. ; TILKE, P. : An events based algorithm for distributing concurrent tasks on multi-core architectures. In: *Computer Physics Communications* 181 (2010), S. 341–354

[77] HP.: *HP ARM servers*. `http://www.wired.com/wiredenterprise/2011/10/hp-arm-servers/`. Version: 2011. – Acceded on 10 December 2011

[78] HP: *Project Moonshot, low-energy servers*. `http://h17007.www1.hp.com/us/en/iss/110111.aspx`. Version: Dec 2011. – Acceded on 10th December 2011

[79] HWU, W. mei ; RYOO, S. ; UENG, S. zee ; KELM, J. H. ; GELADO, I. ; STONE, S. S. ; KIDD, R. E. ; BAGHSORKHI, S. S. ; MAHESRI, A. A. ; TSAO, S. C. ; NAVARRO, N. ; LUMETTA, S. S. ; FRANK, M. I. ; PATEL, S. J.: Implicitly parallel programming models for thousand-core microprocessors. In: *In DAC*, IEEE, 2007, S. 754–759

[80] IRGENS, F. : *Continuum Mechanics*. Springer, 2008

[81] JANEZIC, D. ; BORSTNIK, U. ; PRAPROTNIK: Parallel Approaches in Molecular Dynamics Simulations. In: *Parallel Computing Numerics, Applications, and Trends*. Springer, 2009

[82] JING, L. ; STEPHANSSON, O. : *Fundamentals of Discrete Element Methods for Rock Engineering*. Elsevier, 2007

[83] KACZMARCZYK, L. ; J. PEARCE, C. J. ; BICANIC, N. ; SOUZA NETO, E. de: Numerical multiscale solution strategy for fracturing heterogeneous materials. In: *Computer Methods in Applied Mechanics and Engineering* 199 (2010), S. 1100–1113

[84] KAIDI, S. ; ROUAINIA, M. ; OUAHSINE, A. : Stability of breakwaters under hydrodynamic loading using a coupled DDA/FEM approach. In: *Ocean Engineering* 55 (2012), S. 62 – 70

[85] KARAKASIDIS, T. E. ; CHOLEVAS, N. S. ; LIAKOPOULOS, A. B.: Parallel short range molecular dynamics simulations on computer clusters: Performance evaluation and modeling. In: *Mathematical and Computer Modelling* 42 (2005), S. 783–798

[86] KARNIADAKIS, G. E. ; KIRBI II, R. : *Parallel Scientific Computing in C++ and MPI A seamless approach to parallel algorithms and their implementation.* Cambridge University Press, 2003

[87] KENNETH HAMBLING, W. ; CHRISTIANSEN, E. H.: *Earth's Dynamic Systems Web Edition 1.0.* `http://earthds.info`. Version: Oct 2011

[88] KERNIGHAN, B. W. ; RITCHIE, D. M.: *The C Programming Language (2nd Edition).* Prentice Hall; 2 edition, 1988

[89] KIM, H. ; BUTTAR, W. G. ; PARLT: *Investigation of fracture toughening mechanisms of asphalt concrete using the clustered discrete element method.* `http://www.strc.ch/conferences/2008`. Version: October 2011

[90] KIM, H. ; WAGONER, M. ; BUTTLAR, W. : Micromechanical fracture modeling of asphalt concrete using a single-edge notched beam test. In: *Materials and Structures* 42 (2009), S. 677–689

[91] KIRK, D. B. ; W. HWU, W. mei: *Programming Massively Parallel Processors: A Hands-on Approach.* Elsevier, 2010

[92] KITTI, R. : *Coupled DEM-FEM for dynamic analysis of granular systems in bending*, University of Nebraska, Diss., 2010

[93] KNOTT, G. : A balanced tree storage and retrieval algorithm. In: *Annual ACM Conference on Research and Development in Information Retrieval*, 1971, S. 175–196

[94] KOGGE, P. : *Next-Generation Supercomputers.* `http://spectrum.ieee.org/computing/hardware/nextgeneration-supercomputers/0`. Version: Feb 2011. – Accessed 26 Feb 2013

[95] Kolisnki, A. (Hrsg.): *Multiscale Approaches to Protein Modeling*. Springer, 2011

[96] Latham, J. P. ; Munjiza, A. : The modelling of particle systems with real shapes. In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 362, Nr. 1822, S. 1972

[97] Latham, J.-P. ; Munjiza, A. ; Garcia, X. ; Xiang, J. ; Guises, R. : Three-dimensional particle shape acquisition and use of shape library for DEM and FEM/DEM simulation. In: *Minerals Engineering* 21 (2008), Nr. 11, S. 797 – 805

[98] Leach, A. R.: *Molecular Modelling PRINCIPLES AND APPLICATIONS*. Prentice Hall, 2001

[99] Lei, Z. ; Zang, M. ; Munjiza, A. : Implementation of combined single and smeared crack model in 3D Combined Finite-Discrete Element Ana. In: *Discrete Element Methods. Simulations of Discontinua: Theory and Applications*, 2010

[100] LeVeque, R. : *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. Society for Industrial and Applied Mathematics, 2007 (Classics in Applied Mathematics Classics in Applied Mathemat). – ISBN 9780898716290

[101] Lewis, R. W. ; Gethin, D. T. ; Yang, X. S. ; Rowe, R. C.: A combined finite-discrete element method for simulating pharmaceutical powder tableting. In: *International Journal for Numerical Methods in Engineering* 62 (2005), Nr. 7, S. 853–869

[102] Li, C. F. ; Feng, Y. T. ; Owen, D. R. J.: SMB: Collision detection based on temporal coherence. In: *Computer Methods in Applied Mechanics and Engineering* 195 (2006), Nr. 19-22, S. 2252 – 2269

[103] Li, X. ; Chu, X. ; Sheng, D. C.: A saturated discrete particle model and characteristic-based SPH method in granular materials. In: *International Journal for Numerical Methods in Engineering* 72 (2007), Nr. 7, S. 858–882

[104] Lippman, S. B.: *Essential C++ (The C++ In-depth Series)*. Addison Wesle, 1999

[105] LISJAK, A. ; GRASSELLI, G. : Rock Impact Modelling Using FEM/DEM. In: *Discrete Element Methods: Simulation of Discontinua Theory and Applications*, 2010, S. 269–274

[106] LIU, G. R. ; LIU, M. : *Smoothed Particle Hydrodynamics: A Meshfree Particle Method*. World Scientific Publishing Company, 2003

[107] LIU, M. B. ; LIU, G. R. ; LAM, K. Y. ; ZONG, Z. : Smoothed particle hydrodynamics for numerical simulation of underwater explosion. In: *Computational Mechanics* 30 (2003), S. 106–118

[108] LIU, M. ; LIU, G. ; ZONG, Z. ; LAM, K. : Computer simulation of high explosive explosion using smoothed particle hydrodynamics methodology. In: *Computers &amp; Fluids* 32 (2003), S. 305 – 322

[109] LIU, W. K. ; KARPOV, E. G. ; ZHANG, S. ; PARK, H. S.: An introduction to computational nanomechanics and materials. In: *Comput. Methods Appl. Mech. Engrg.* 193 (2004), S. 1529 1578

[110] LOGAN, L. D.: *A First Course in the Finite Element Method*. Thomson, 2007

[111] LOMINE, F. ; SCHOLTES, L. ; SIBILLE, L. ; POULLAIN, P. : Modeling of fluid-solid interaction in granular media with coupled lattice Bolzmann/discrete eleen methods:Apliaciton to pinping erosion. In: *Int. J. Numer. Anal., Meth Geomech* 37 (2013), S. 577–596

[112] LUKAS, T. : *Space decomposition based parallelization solutions for the Combined Finite-Discrete Element Method in 2D*, Queen Mary University of London SEMS, Diss., 2014

[113] LUKAS, T. ; MUNJIZA, A. : Parallelization of an open-source FEM/DEM code Y2D. In: *Simulations of Discontinua Theory and Applications -DEM 5*, 2010, S. 206 211

[114] LUKAS, T. ; MUNJIZA, A. : Space decomposition based parallelization solutions for the Combined Finite-Discrete Element Method in 2D. In: *DEM 6*, 2013, S. 234–239

[115] LURIE, A. I.: *Theory of Elasticity*. Springer, 2005

[116] MACLAUGHLIN, M. ; DOOLIN, D. : Review of validation of the discontinuous deformation analysis (DDA) method. In: *International Journal for Numerical and Analytical Methods in Geomechanics* 30 (2006), S. 271–305

[117] MACNEAL, R. H. ; HARDER, R. L.: A proposed standard set of problems to test finite element accuracy. In: *Finite Elements in Analysis and Design* 1 (1985), Nr. 1, S. 3 – 20

[118] MADRON, P. ; REMINGTON, E. : *CISC 2006 Parallel Programming MPI REDUCE*. `http://www.eecis.udel.edu/~pollock/372/f06/`. Version: 2006. – Accessed 27 Feb 2013

[119] MAHABADI, O. K.: *Investigating the influence of micro-scale heterogeneity and microstructure on the failure and mechanical behaviour of geomaterials*, Graduate Department of Civil Engineering University of Toronto, Diss., 2012

[120] MAHABADI, O. K. ; RANDALL, N. X. ; ZONG, Z. ; GRA: A novel approach for micro-scale characterisation and modeling of geomaterials incorporating actual material heterogeneity. In: *Geophysical Research Letters* 39 (2012), S. 1–6

[121] MANZELLA, I. ; LISJAK, A. ; MAHABADI, O. K. ; GRASSELLI, G. : Influence of initial block packing on rock avalanche flow and emplacement mechanisms through FEM/DEM simulations. In: *Actes de la 14eme Conference Pan-American on Soil Mechanics and Geotechnical Engineering*, 2011

[122] MAWSON, M. J.: *S3270: Fast Fluid-structure Interaction Using Lattice Boltzmann and Immersed Boundary Methods*. `http://nvidia.fullviewmedia.com/gtc2013/0320-231-S3270.html`. Version: Apr 2013. – Accessed Apr 2013

[123] MCDOWELL, G. ; LI, H. ; LOWNDES, I. : *The importance of particle shape in discrete-element modelling of particle flow in a chute*

[124] MEHRA, V. ; CHATURVEDI, S. : High velocity impact of metal sphere on thin metallic plates: A comparative smooth particle hydrodynamics study. In: *Journal of Computational Physics* 212 (2006), S. 318 – 337

[125] METROPOLIS, N. A. ; ROSENBLUTH, A. ; ROSENBLUTH, M. N. ; TELLER, A. H. ; TELLER, E. : Equation of State Calculations by Fast Computing Machines. In: *J. Chem. Phys.* 21 (1953), S. 1087–1092

[126] MISHRA, B. K.: A review of compuer simulation of tublming mills by the discrete element method Part II–Practical applicationS. In: *Int. J.* 83 (2003), S. 95–112

[127] MISHRA, B. K.: A review of computer simulation of tumbling mills by the discrete element method Part I- contact mechanics. In: *Int. J. Miner. Process* 71 (2003), S. 73–93

[128] MISHRA, B. ; THORNTON, C. : Impact breakage of particle agglomerates. In: *International Journal of Mineral Processing* 61 (2001), S. 225 – 239

[129] MOHAMMADI, S. : *Discontinuum mechanics: using finite and discrete elements*. WIT Press, 2003

[130] MORRIS, J. ; GLENN, L. ; BLAIR, S. ; HEUZE, F. : The Distinct Element Method - Application to Structures in Jointed Rock. In: *International workshop Meshfree Methods of Partial Differential Equations*, 2001

[131] MORRISON, R. ; CLEARY, P. : Using DEM to model ore breakage within a pilot scale SAG mill. In: *Minerals Engineering* 17 (2004), Nr. 11-12, S. 1117–1124

[132] MULLER, M. J. ; BRISEBARRE, N. ; DINECHIN, F. ; LEFEVRE, V. ; MELQUIOND, G. ; REVOL, D. N. S. N. Stehle ; TORRES, S. : *Handbook of Floating-Point Arithmetic*. Birkhauser, 2009

[133] MUNJIZA, A. : *The Combined Finite Discrete Element Method*. Wiley, 2004

[134] MUNJIZA, A. ; ANDREWS, K. R. F.: Penalty function method for combined finite discrete element systemscomprising large number of separate bodies. In: *International Journal for Numerical Methods in Engineering* 49 (2000), Nr. 11, S. 1377–1396

[135] MUNJIZA, A. ; ANDREWS, K. R. F. ; WHITE, J. K.: Combined single and smeared crack model in combined Finite-Discrete Element Analysis. In: *Int* 44 (1999), S. 41–57

[136] MUNJIZA, A. ; ANDREWS, K. : NBS contact detection algorithm for bodies of similar size. In: *Int. J. Numer. Meth. Eng* 46 (1998)

[137] MUNJIZA, A. ; BANGASH, T. ; JOHN, N. W. M.: The combined finite-discrete element method for structural failure and collapse. In: *Engineering Fracture Mechanics* 71 (2004), S. 469–483

[138] MUNJIZA, A. ; CARNEY, T. ; KNIGHT, E. ; SWIFT, R. P. ; GREENING, D. ; STEEDMAN, D. : The roots of possible chaoic behaviour in modelling and simulation of discontinua. In: *IASS-IACM 2008*, Internet-First University Press

[139] MUNJIZA, A. ; JOHN, N. W. M.: Towards one billion Particle System. In: *The 3rd MIT conference on computational fluid and solid mechanics*, 2005

[140] MUNJIZA, A. ; JOHN, N. : Mesh size sensitivity of the combined FEM/DEM fracture and fragmentation algorithms. In: *Engineering Fracture Mechanics* 69 (2002), S. 281–295

[141] MUNJIZA, A. ; KNIGHT, E. ; E., R. : *Computational Mechanics of Discontinua*. Wiley, 2011

[142] MUNJIZA, A. ; LATHAM, J. P.: Some computational and algorithmic developments in computational mechanics of discontinua. In: *Phil. Trans. R,. Soc. Lond. A* 362 (2004), S. 1817–1833

[143] MUNJIZA, A. ; LATHAM, J. P. ; ANDREWS, K. R. F.: Detonation gas model for combined finite-discrete element simulation of fracture and fragmentation. In: *International Journal for Numerical Methods in Engineering* 49 (2000), Nr. 12, S. 1495–1520

[144] MUNJIZA, A. ; LATHAM, J. P. ; JOHN, N. W. M.: 3D dynamics of discrete element systems comprising irregular discrete elements integration solution for finite rotations in 3D. In: *International Journal for Numerical Methods in Engineering* 56 (2003), S. 35–55

[145] MUNJIZA, A. ; OWEN, D. R. J. ; BICANIC, N. : A combined finite-discrete element method in transient dynamics of fracturing solids. In: *Eng. Comput.* 12 (1995), S. 145–174

[146] MUNJIZA, A. ; OWEN, D. R. J. ; CROOK, A. J. L.: An M(M-1K)m proportional damping in explicit integration of dynamic structural systems. In: *International Journal for Numerical Methods in Engineering* 41 (1998), Nr. 7, S. 1277–1296. – ISSN 1097–0207

[147] MUNJIZA, A. ; ROUGIER, E. ; JOHN, N. : Virtual experimentation in the service of theoretical and experimental science. In: *Third MIT Conference on Computational and Solid Mechanics*, 2005

[148] MUNJIZA, A. ; ROUGIER, E. ; JOHN, N. : MR linear contact detection algorithm. In: *Int. J. Numer. Meth. Eng* 66 (2006), S. 46–71

[149] MUNJIZA, A. ; XIANG, J. ; GARCIA, X. ; LATHAM, J. P. ; SCHIAVA D'ALBANO, G. G. ; JOHN, N. : The Virtual Geoscience Workbench, VGW: Open Source tools for discontinous systems. In: *Particuology* 8 (2010), S. 100–105

[150] MUNJIZA, A. ; ZHOU, Z. ; DIVIC, V. ; PEROS, B. : Fracture and fragmentation of thin shells using the combined finite discrete element method. In: *Int J Num Meth in Eng* (accepted for publication) (2012)

[151] MUNSHI, A. ; GASTER, B. R. ; MATTSON, T. G. ; FUNG, J. ; GINSBURG, D. : *OpenCL Programming Guide*. Addison Wesley, 2011

[152] NELSON, M. T. ; HUMPHREY, W. ; GURSOY, A. ; DALKE, A. ; KALE, L. V. ; SKEEL, R. D. ; SCHULTEN, K. : NAMD: a Parallel, Object-Oriented Molecular Dynamics Program. In: *International Journal of High Performance Computing Applications* 10 (1996), S. 251–268

[153] NGUYEN: *GPU Gems 3*. Addison Wesley Professional, 2007

[154] NOH, H. C. ; CHOI, C. H.: Ultimate behaviour of reinforced concrete cooling tower: Evaluation and comparion of design guidelines. In: *Structural Engineering and Mechanics* 22 (2006), S. 223–240

[155] NVIDIA (Hrsg.): *OpenCL Programming Guide for the CUDA Architecture Version 3.1*. 2010

[156] ONATE, E. (Hrsg.) ; OWEN, R. (Hrsg.): *Computational Methods in Applied Sciences*. Bd. 25: *Particle-Based Methods Fundamentals and Applications*. Springer, 2011

[157] OWEN, D. R. J. ; PERIC, D. ; PETRINIC, N. ; BROOKES, C. L. ; JAMES, P. J.: Finite/discrete element models for assessment and repair of masonry structures. In: SINOPOLI, A. (Hrsg.): *Arch bridges: history, analysis, assessment, maintenance and repair*, 1996

[158] OWENS, J. D. ; HOUSTON, M. ; LUEBKE, D. ; GREEN, S. ; STONE, J. E. ; PHILLIPS, J. C.: GPU Computing. In: *Proceedings of the IEEE* 96 (2008), S. 879–899

[159] PACHECO, P. : *Parallel Programing with MPI*. Bd. g. Morgan Kaufmann, 1995

[160] PACHECO, P. : *An Introduction to Parallel Programming*. Morgran Kaufmann, 2011

[161] PAJAROLA, R. : Large Scale Terrain Visualization Using The Restricted Quadtree Triangulation. In: *Proceedings Visualization 98*, 1998

[162] PARKER, A. P.: *The Mechanics of Fracture and Fatigue*. E. & F. N. Spon Ltd, 1981

[163] PERKINGS, P. ; WILLIAMS, J. : C-grid: Neighbour searching for many body simulations. In: *ICADD-4*, 2001

[164] PETERSEN, W. J.: *Patch Testing of finite Element Software*, Department of Civil And Enviromental Engineering. College of Engineering And Technology Brigham Young University, Diplomarbeit, 2004

[165] PODGORELEC, D. ; KLAJNVEK, G. : Acceleration of sweep-line technique by employing smart quicksort. In: *Inf. Sci. Inf. Comput. Sci.* 169 (2005), S. 383–408

[166] PRESS, W. H. ; TEUKOLSKY, S. A. ; T., W. ; P., F. B.: *Numerical Recipes in Fortran 77 The Art of Scientific Computing Second Edition*. Press Syndicate of the University of Cambridge, 1997

[167] PROJECT, S. : *Seti Project*. `http://setiathome.berkeley.edu/sah_participate.php`. – Accessed Feb 26 2013

[168] QUINN, M. J.: *Parallell Programming in C with MPI and OpenMP*. Mc Graw Hill Higher Education, 2003

[169] RABCZUK, T. ; EIBL, J. : Simulation of high velocity concrete fragmentation using SPH/MLSPH. In: *International Journal for Numerical Methods in Engineering* 56 (2003), S. 1421–1444

[170] RADEKE, K. J. A.C.: WET-MIXING OF POWDERS, A LARGE-SCALE GPU IMPLEMENTATION. In: *Simulations of Discontinua Theory and Applications*, 2010, S. 212–221

[171] RAPAPORT, D. : *The Art of MOLECULAR DYNAMICS SIMULATION Second Editon*. CAMBRIGE UNIVERSITY PRESS, 2004

[172] RICHARDS, K. ; BITHELL, M. ; DOVE, M. ; HODGE, R. : Discrete-element modelling: methods and applications in the environmental sciences. In: *Phil. Trans. R. Soc. Lond.* 362 (2004), S. 1797–1816

[173] RINALDI, P. R. ; DARI, E. A. ; VENERE, M. J. ; CLAUSSE, A. : A Lattice-Boltzmann solver for 3D fluid simulation on GPU. In: *Simulation Modelling Practice and Theory* 25 (2012), S. 163–171

[174] ROJEK, J. ; ONATE, E. ; LABRA, C. ; KARGL, H. : Discrete Element Modelling of Rock Cutting. In: *Particle-Based Methods*. Springer, 2011, S. 247–267

[175] ROUGIER, E. : *Discrete Element Method for Simulation of Gas Micro-Flows*, Queen Mary University Of London, Diss., 2009

[176] RUIZ, G. ; ZHANG, X. X. ; TARIFA, M. ; YU, R. C. ; CAMARA, M. : Fracture energy of high-strength concrete under different loading rates. In: *Anales de Mecanica de la Fractura* 26 (2009), S. 513–518

[177] RUMP, S. M.: Reliability in computing: the role of interval methods in scientific computing. Academic Press Professional, Inc., 1988, Kapitel Algorithms for verified inclusions theory and practice, S. 109–126

[178] SADUS, R. J.: *Molecular Simulation of Fluids Theory, Algorithms and Object-Orientation*. Elsevier, 1999

[179] SALLAM, A. M.: *Studies on modeling angular soil particles using the discrete element method*, University of South Florida Scholar Commons, Diss., 2004

[180] SENGUPTA, S. ; HARRIS, M. ; GARLAND, M. ; OWENS, J. D.: Efficient Parallel Scan Algorithms for many-core GPUs. In: KURZAK, J. (Hrsg.) ; BADER, D. A. (Hrsg.) ; DONGARRA, J. (Hrsg.): *Scientific Computing with Multicore and Accelerators*. Taylor & Francis, 2011 (Chapman & Hall/CRC Computational Science), Kapitel 19, S. 413–442

[181] SHAFFER, C. A. ; SAMET, H. : Optimal Quadtree Construction Algorithms. In: *Computer Vision, Graphics and Image Processings* 37 (1987), S. 402–419

[182] SHI, G. : *Discontinuous deformation analysis a new numerical model for statics and dynamics of block systems*, University of California Berkeley, Diss., 1988

[183] SIGALOTTI, L. D. G. ; LÓPEZ, H. ; DONOSO, A. ; SIRA, E. ; KLAPP, J. : A shock-capturing SPH scheme based on adaptive kernel estimation. In: *J. Comput. Phys.* 212 (2006), S. 124–149

[184] SITHARAM, T. G.: Numerical simulation of particulate materials using discrete element modelling. In: *Current Science* 78 (2000), S. 876–886

[185] SRINIVASAN, S. G. ; ASHOK, I. ; JONSSON, H. ; KALONKI, G. ; ZAHORJAN, J. : Dynamic-domain decomposition parallel molecular dynamcs. In: *Computer P* 102 (1997), S. 44–58

[186] SUCCI, S. : *The Lattice Boltzmann Equation for Fluid dynamic and Beyond*. Oxford Sience Publications, 2001

[187] SUTTER, H. ; LARUS, J. : Software and the concurrency revolution. In: *Queue - Multiprocessors* 3 (2005), S. 54–62

[188] TROBEC, R. (Hrsg.) ; VAJTERSIC, M. (Hrsg.) ; ZINTERHOF, P. (Hrsg.): *Parallel Computing Numerical, Applications, and Trends*. Springer, 2009

[189] UNGERER, P. ; NIETO-DRAHI, G. ; ROUSSEAU, B. ; AHUNBAY, G. ; LACHET, V. : Molecular simulation of thermophysical properties of fluids: From understanding toward quantitative predictions. In: *Journal of Molecular Liquids* 134 (2007), S. 71 89

[190] VIDAL, Y. ; BONET, J. ; HUERTA, A. : Stabilized updated Lagrangian corrected SPH for explicit dynamic problems. In: *International Journal for Numerical Methods in Engineering* 69 (2007), S. 2687–2710

[191] VIOLEAU, D. ; ISSA, R. : Numerical modelling of complex turbulent free surface flows with the SPH method: an overview. In: *International Journal for Numerical Methods in Fluids* 53 (2007), S. 277–304

[192] VVEDENSKY, D. D.: Multiscale modelling of nanostructures. In: *Journal of Physics: Condensed Matter* 16 (2004), S. R1537–R1576

[193] WANG, J. ; RYAN, D. ; ANTHONY, E. J.: Reducing the greenhouse gas footprint of shale gas. In: *Energy Policy* 39 (2011), S. 8196–8199

[194] WANG, L. ; PARK, J.-Y. ; FU, Y. : Representation of real particles for DEM simulation using X-ray tomography. In: *Construction and Building Materials* 21 (2007), S. 338 – 346

[195] WANG, M. : On the Necessity And Suffiency of the Patch Test for Convergence of Nonconforming Finite Elements. In: *Siam J. Numer. Anal.* 39 (2001), S. 363–384

[196] WEN-MEI, H. ; KEUTZER, K. ; MATTSON, T. : The Concurrency Challenge. In: *Design Test of Computers, IEEE* 25 (2008), S. 312–320

[197] WHITAKER, M. ; LAGSTON, P. ; S., H. ; AZZOPARDI, B. : Particle shpe effects in medical syringe needles - Eperiments and simulations for polymer mocropartile injection. In: *Discrete Element Methods Simulations of Discontinua: Theory and Applications*, 2010

[198] WILKINSON, B. ; ALLEN, M. : *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers (2nd Edition)*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 2004. – ISBN 0131405632

[199] WILLIAM, H. ; TEUKOLSKY, S. A. ; VETTERLING, W. T. ; FLANNERY, B. P.: *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2002

[200] WILLIAMS, J. ; O'CONNOR, R. : Discrete element simulation and the contact problem. In: *Archives of Computational Methods in Engineering* 6 (1999), S. 279–304

[201] WILLIAMS, J. R. ; HOLMES, D. ; TILKE, P. : Multi-Core Strategies for Particle Methods. In: *Discrete Element Methods Simulations of Discontinua: Theory and Applications*, 2010

[202] WILLIAMS, J. R. ; HOLMES, D. ; TILKE, P. : Muli-core Stragegies For Particle Methods. In: *Discrete Element Methods Simulations of Discontinua: Theory and Applications*, 2010

[203] WILLIAMS, J. ; PENTALND, A. : Superquadrics and modal dynamics for discrete elements in concurrent desing. / 1991. MIT. – Technical Report No. IESL91-12, Intelligent Engineering Systems Laboratory

[204] WITTENBURG, J. ; TEUBNER, B. G. (Hrsg.): *Dynamics of systems of rigid bodies*. Teubner, 1977. – ISBN 9783519023371

[205] XIANG, J. : *Investigation of particle motion in dense phase pneumatic conveying*, School of Engineering, Science and Desing. Glasgow Caledonian University, Diss., 2004

[206] XIANG, J. ; MUNJIZA, A. ; LATHAM, J.-P. : Finite strain, finite rotation quadratic tetrahedral element for the combined finite discrete element method. In: *International Journal for Numerical Methods in Engineering* 79 (2009), S. 946–978

[207] YAMADA, Y. ; SAKAI, M. ; M., T. ; HIRAYAMA, S. : Development of a Rotational Resistance Model In the Discrete Element Method. In: *Discrete Element Methods Simulations of Discontinua: Theory and Applications*, 2010

[208] YANG, B. ; JIAO, Y. ; LEI, S. : A study on the effects of microparameters on macroproperties for specimens created by bounded particles. In: *International Journal for Computer-Aided Engineering and Software* 23 (2006), S. 607–631

[209] YOU, Z. ; DAI, Q. : Update on the Discrete Element Method in Engineering Education. In: *2006 IJME-INTERTECH*

[210] ZHOU, L. ; HOU, M. Z.: A new numerical 3D-model for simulation of hydraulic fracturing in consideration of hydro-mechanical coupling effects. In: *International Journal of Rock* 60 (2013), S. 370–380

[211] ZIENKIEWICZ, O. ; TAYLOR, R. L.: *The Finite Element Method.* Butterworth Heinemann, 2000

[212] ZSAKI, A. M.: Parallel generation of initial element assemblies for two-dimensional discrete element simulations. In: *Int J Numer Anal Methods Geomech* 33 (2009), S. 377–389