A HIGH PERFORMANCE IMPLEMENTATION OF THE DISCRETE ELEMENT METHOD USING MPI

Maria Cecília Rodrigues Sena, mariacecilia@lccv.ufal.br
Eduardo Setton Sampaio da Silveira, eduardosetton@lccv.ufal.br
Laboratory of Scientific Computing and Visualization – LCCV
Center of Technology – CTEC
Federal University of Alagoas – UFAL
Campus A. C. Simões – Tabuleiro do Martins – CEP: 57.072–970 – Maceio – Alagoas – Brazil

Abstract. *The Discrete Element Method has been used a lot to simulate the mechanical behavior of discontinuous bodies. A huge problem associated with its application is the high computational cost to simulate it, because it's necessary to discretize the domain in a huge number of particles, which can make the method impracticable in some applications. In this context, this paper presents a proposal and the computational implementation of a parallel simulator by using the C++ programming language and the MPI message exchanging library. This paper shows: a summarized description about the theme, the followed methodology, some aspects about the implementation and the results achieved.*

Keywords*: Discrete Element Method, High Performance Computing, Clusters, MPI.*

1. INTRODUCTION

The Discrete Element Method (DEM) is a computational technique that simulates, mainly, the mechanical behavior of discontinuous bodies on the Granular Scale. Modeling it, the Macroscopic Scale answers result from the mechanic mechanism interaction among the elements that constitutes the studied system (Cundall, 1979). Therefore, the DEM constitutes a good research tool for the behavior study of these domains in many areas of knowledge. In Engineering, the method is applied in the simulation of granular material draining, in the modeling of soil and rocks, in fluid problems, among other applications.

In a simulation using the Discrete Element Method, the analyzed domain is subdivided into a set of particles, which constitutes the basic unit of analysis. This way, it's extremely necessary to subdivide the studied domain into a great number of particles, so it can be represented by the DEM in a satisfactory way, making the DEM simulations extremely slow. Besides that, the number of particles to be used in a simulation is limited by the memory available in the machine where the application is being executed. With this, the use of high performance computing appears as an interesting alternative that can solve the computational performance and the required memory problems which uses the DEM. The use of computer clusters has grown a lot in recent years as a low cost alternative for the solution of applications that require a high computational demand, because of a great reduction at the time of simulation and a bigger memory space for the storage of information can be obtained by using many processors and memory with low cost.

In this context, this paper presents a proposal and an implementation of a C++ object oriented simulator in an parallel computing environment, using the message passing library MPI (Message Passing Interface), just like some advantages achieved with the implementation presented in this paper.

2. THE DISCRETE ELEMENT METHOD

The Discrete Element Method consists in the representation of a domain, continuous or discontinuous, by an extremely great number of individual particles, in which the behavior of the system is determined by the movement and interaction of these particles.

Normally, these applications involve materials with inherently discrete characteristics, such as granular materials found in structural engineering problems and geomechanics. However, the use of DEM for the simulation of continuous bodies and fluids has been growing in a significant way.

The analyzed domain is subdivided in a series of particles with particular mechanical properties and defined geometries. After the understanding of these properties and the interaction behavior among the particles, the DEM allows the physical and mechanical behavior of the studied model to be evaluated in a macroscopic way.

Initially, the DEM was conceived as a numerical model capable to describe the mechanical behavior of a set of particles like disk or sphere shapes (Cundall, 1979). Currently the method is also applied for particles of many shapes and, even so, agglomerated of particles.

In the DEM, the studied domain is evaluated at each time step required by the time integration algorithm. In a general way, the method formulation consists on the application of the Newton's second Law, getting the acceleration of each particle from the forces that act in the same one, proceeding from the external domain and the interaction between them. By the integrations, it is possible to obtain the velocities and displacements from the acceleration.

Besides this, Force-Displacement Laws are also applied on the contacts, which calculate the forces in the same ones due to the relative displacements between particles.

To quantify the interaction forces between the pairs of particles, some techniques are used to allow an optimized search of contacts. These techniques must be developed in a robust and efficient way (Munjiza, 2004), so that all the contacts can be found without a high computational cost.

In the Figure 1 is presented a summarized project of the main calculation stages of the Discrete Element Method.

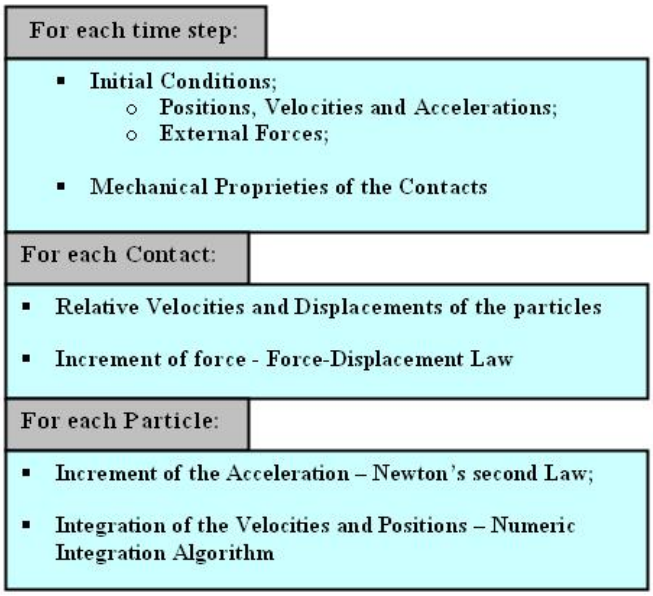

Figure 1. Algorithm for the Discrete Element Method

3. PARALLEL PROCESSING

The increasing number of computational systems that have been developed to solve Engineering problems has demanded the use of faster processors and a great capacity of computer memory. However, the required processing speed hardly can be reached with the conventional processors, with the use of only one processor and with the amount of memory available in a single machine. Generally, these simulations involve the processing of an enormous quantity of data and execute a great number of iterations. The parallel computing many times makes available some of these applications, reducing the computational time of the simulation.

Generally, two types of parallel architectures are used: machines with shared memory and machines with distributed memory. In the first type, the processors operate independently, but they share the resource of only one memory. In the second type, each processor has its own memory and the data are shared through the network, using a mechanism of message exchange (Message Passing). Currently, the distributed memory architecture has been used more in the industries and scientific communities, in virtue of the low cost of clusters of computers that use this type of architecture.

The MPI (Message Passing Interface) has become a standard when the mechanism of message exchange is used for parallel environments, especially those ones with distributed memory. In this model, generally, the execution of an application consists on the execution of many processes that communicate with each other through the library's routines that manage the sending and receiving messages process among the processes.

The performance of a parallel program is analyzed by a series of metric, such as speed-up and efficiency. Speed-up is a measurement that indicates the relative benefit when a parallel program is executed and is defined as the ratio between the wasted time in solving a problem with only one processor (T) and the required time to solve the same problem in p processes (T_P).

$$Speed\text{-}up = \frac{T}{Tp} \tag{1}$$

Efficiency (η) is defined as the ratio between speed-up and the number of processes (*Np*), corresponding to the measurement of the total execution time fraction that each process consumes to execute the application.

$$\eta = \frac{Speedup}{Np} \tag{2}$$

4. COMPUTATIONAL IMPLEMENTATION

3.1. Demoop

The computational implementation taken as the basis for the development of this application was the Demoop program (Discrete Element Method Object Oriented Programming) (Carvalho Jr. *et al*, 2005). This system is divided into three distinct parts which are responsible for the phases of pre-processing, processing and visualization, which offers some advantages related to the relationship between the system and the users or programmers.

In the processing stage, the solution algorithm used in the Demoop shows a repetition routine of processes which includes the checking of contacts between particles; the determination of the forces acting on particles, which are provoked by the contacts with the other particles and by the external actions; and the calculation of the internal forces from the linking elements (Carvalho Jr. *et al*, 2005).

The Demoop was written with the use of the C++ programming language. This choice is based on the presumption that the program code can receive future implementations and it can easily be expanded, not requiring significant changes in the content.

In the Figure 2 is shown the classes diagram of the Demoop application, emphasizing, among themselves, the *Demoop* class, which is responsible for transforming the analysis information supplied by the Input file into a numeric model; *Model* class, which characterizes the numeric model during all the analysis process; *TimeIntegration* class, which is responsible for the numeric integration process; *ContactSearch* class, which is responsible for the checking of contacts between particles of the analyzed domain; and *Particle* class, which is responsible for the representation of the discontinuous body, by the establishment of a formal definition for the entities on which the DEM operates.

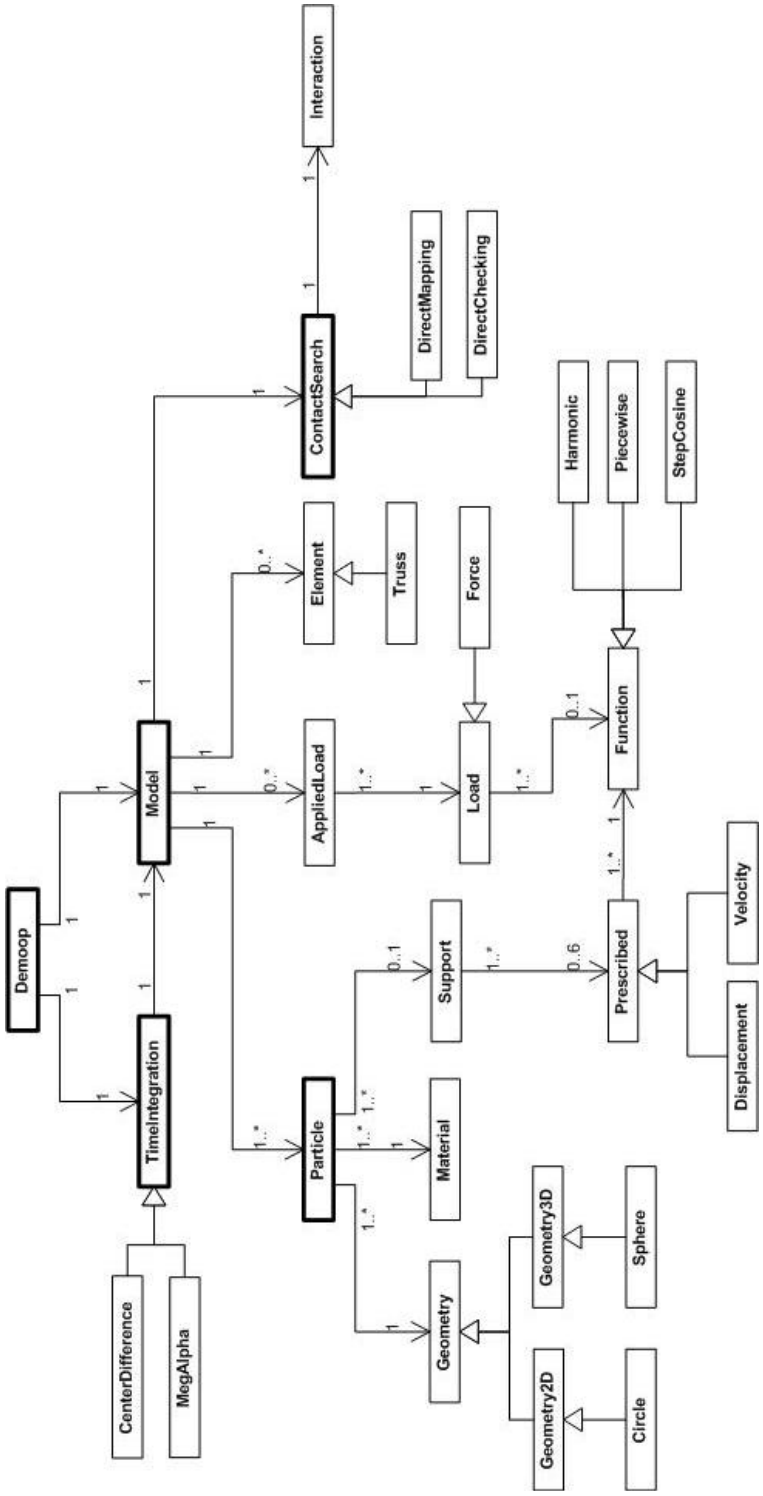

Figure 2. Demoop Classes Diagram

3.2. Parallel-Demoop

For the parallel implementation of the Discrete Element Method (P-Demoop) it was used the C++ programming language, the 7.1.2 LAN/MPI distribution and compiler MPIC++.

In the parallel version, an additional class was created (cMPI), responsible for the sending and receiving of data and for the information related to the parallel processing. The cMPI class has as its attributes the identification of each process and the total number of processes. It has also methods that control the sending and receiving of messages in the

various stages of the simulation. The parallel implementation uses the master-slave mechanism, in which the process master is responsible for the initial tasks of the application and for the distribution of some information to the slave processes. The parallel algorithm was developed through the decomposition of the domain in *n-1* sub-domains, where *n* is the number of processes (Figure 3).

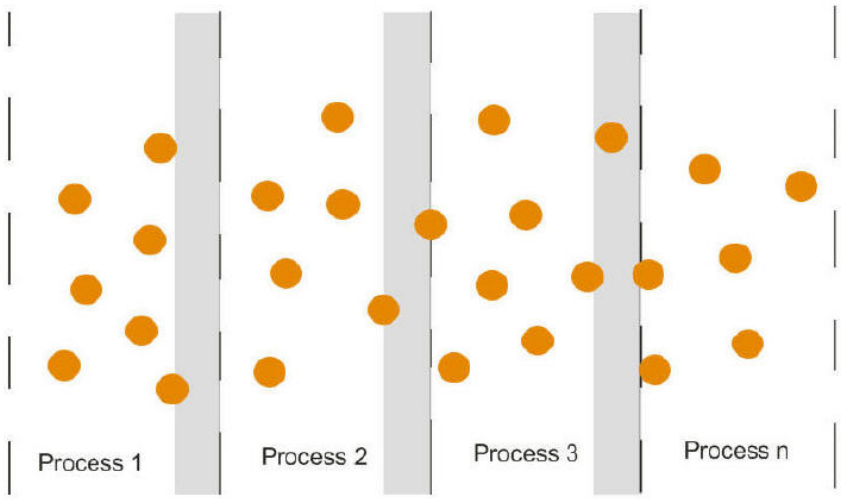

Figure 3. Domain decomposition among the processes

After the reading of the Input data, each process essentially executes the same tasks that the sequential algorithm does, however only on the particles that are found in the region of its sub-domain. To print the results at each step of impression, each process creates an Output archive and writes the positions of the particles that are in its domain.

When a particle leaves the region associated to a process and enters in another sub-domain controlled by another process, the information of this particle (position, velocities, acceleration and force) are sent from a process to another, by the MPI basic functions of sending and receiving messages (MPI_Send and MPI_Recv). From then on, the data of this particle are updated and it starts to be treated by the process in which it just entered.

The particles that are found in the contour of the domain of each process can be in contact with particles of the same process just as with particles from the neighboring process. However, in accordance with the scheme of the domain division presented, each process would possess the information of the particles that are in its domain, leaving, this way, to compute the contacts between particles of different processes. To solve this problem, each process sends to the neighboring process on its right side the particles that are in the right contour of the subdivision (particles that are in the colored bands in Figure 3). Then, the process that receives these particles, checks the contacts between them and the other particles of its domain, calculates the corresponding forces, in the case of having contacts, and it sends them to the neighboring process on its left side.

4. RESULTS

The Cluster of The Laboratory of Scientific Computing and Visualization of The Federal University of Alagoas (Figure 4) was used to evaluate the parallel program performance. This Cluster has 44 nodes, based on shared memory architecture and with the following configuration: Core 2 Duo, 1GB of Ram, HD 80GB, Linux Operational System, Fedora distribution. The Cluster nodes are connected by a Gigabit Ethernet.

Figura 4. Cluster of the Laboratory of Scientific Computing and Visualization

This section presents the results obtained with the simulation of mechanical behavior of the soil-duct system submitted to a service load (q) and the covering layers weight, as can be seen on the idealized model of the figure 5. The soil was discretized in 5000 and 10000 circular particles and the total simulation time was 2 seconds, with time step of 0.0002 seconds.

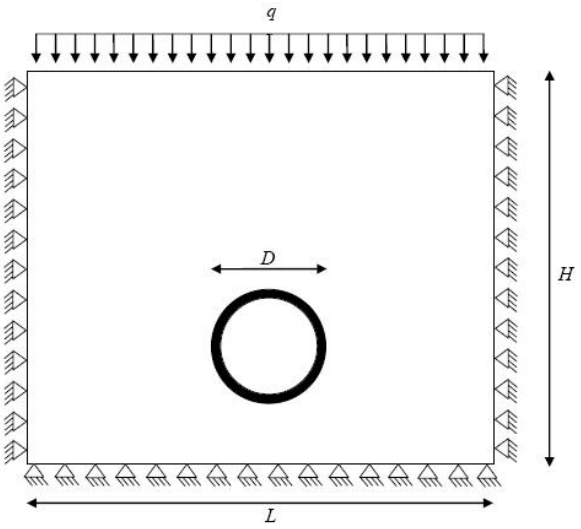

Figure 5. System soil-duct model

The analyzed examples were generated by the pre-processor *Pre-Demoop* (Cavalcante *et al*, 2006) and the results of the simulations can be visualized on the pos-processor *DemoopView* (Amorim *et al*, 2006).

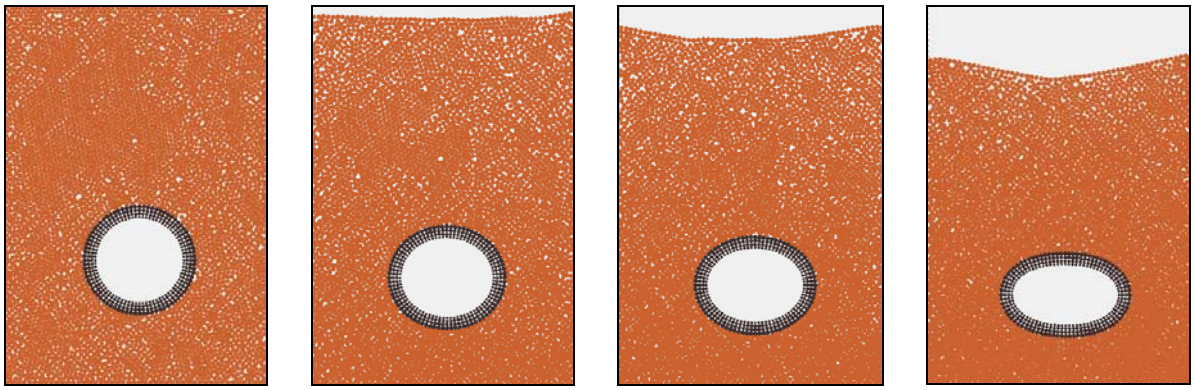

Figure 6. The soil-duct system simulation

The mechanical properties of the contacts among particles used in the soil-duct modeling, such as: normal stiffness (k_n), tangential stiffness (k_T), normal viscous damping (c_n), tangential viscous damping (c_T) and attrition coefficient (μ), are all described in Table 1.

Table 1. Mechanical properties of the discrete elements

Meio	k_n - (kN/m)	c_n- (kN.s/m)	k_t – (kN/m)	c_T – (kN.s/m)	μ
Duto	65,00	0,75	0,09	0,11	0.6
Solo	55,00	0,65	0,07	0,09	0.5

The execution time of the applications on the serial program and on the parallel program with p processors are shown on the Table 2 and on the Figure 7.

Table 2. Tempo de processamento em segundos para n particulas e p processadores.

	$p = 1$	$p = 5$	$p = 10$	$p = 15$	$p = 20$	$p = 25$	$p = 30$	$p = 35$
$n = 5000$	286,00	73,97	47,00	40,00	36,00	34,00	35,20	33,70
$n = 10\,000$	677	129,80	70,60	55,80	47,50	43,90	42,80	41,70

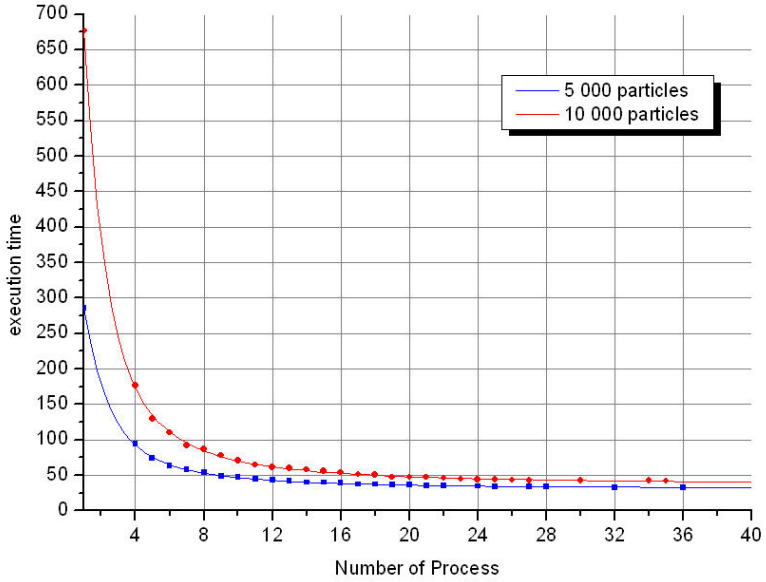

Figure 7 . Processing Time *versus* number of particles for 10 seconds of simulation

Analyzing the Table 2 and the Figure 7, it can be observed how much the execution time is reduced when the simulation is executed on the parallel version. The greater is the number of particles, greater will be the time reduction. It is related with the high processing load that is executed in only one processor, when the serial program is being used. However, when the number of process reaches a limit, the processing time starts to grow because of the increase on the data traffic through the net.

In order to have a better idea of the parallel code performance, the table 3 shows the speed-up values obtained with the simulations.

Table 3. *Speed-up* for *n* particles e *p* processors

	P = 5	*p* = 10	*p* =15	*p* = 20	*p* = 25	*p* = 30	*p* = 35
n = 5000	3,87	6,09	7,15	7,94	8,41		8,69
n = 10 000	5,22	9,59	12,13	14,25	15,42	15,82	16,24

The speed-up graphs are shown on the Figure 8.

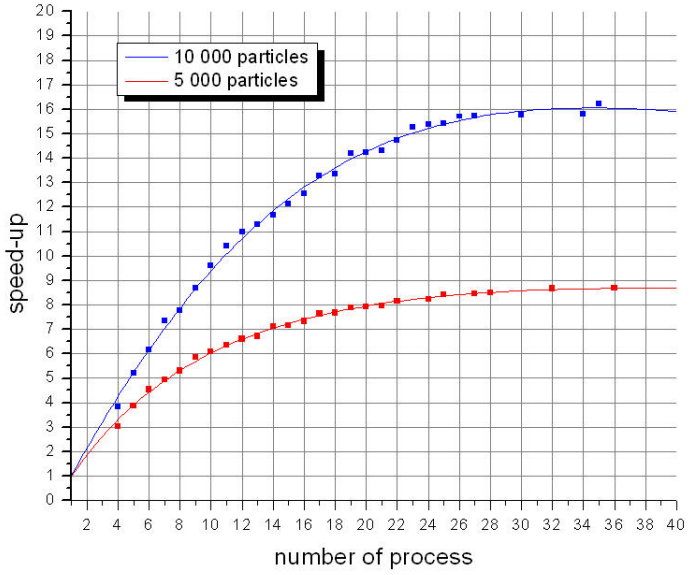

Figure 8. Speed-up *versus* number of particles for 2 seconds of simulation.

For the analyzed example, the greater is the number of particles, the better is the parallel processing performance, as can be observed on the speed-up graphs of the Figure 8. This computational profit is due to the fact that how much bigger the number of particles is, greater are the tasks that each process has to do, what compensate the time that is spent on the traffic of information among the processes through the network. Another point that can be observed from the graphs on Figure 8 is that when the number of processes increases, the value of speed-up grows until reaching a maximum value and later it becomes to decrease. The bigger is the number of processes, the smaller are the tasks that each process will do and greater is the traffic in the network, since more processes will join this communication. Therefore, there is a moment where the profit time with the parallel processing is not enough to compensate the spent time on the data transmission among the processes.

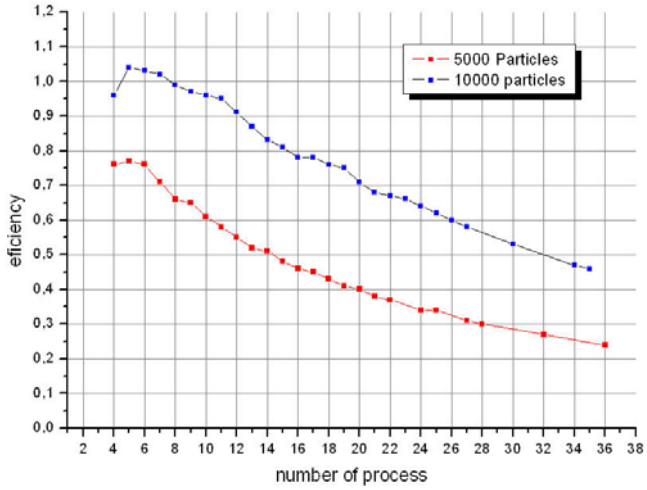

Figure 9. Efficiency *versus* number of particles for 10 seconds of simulation

The parallel processing efficiency (Figure 9) grows when the particle number becomes bigger, what was expected, since the speed-up also grows in this exactly direction. Analyzing the graphs, it is observed that when the number of processes increases, the algorithm efficiency decreases, showing that when speed-up is growing, this growth is given in a lesser ratio than the growth of the number of processes.

When the serial code performance is analyzed, it can be observed that 75% of the simulation time is spent in the search of contacts between the particles (Table 4), since this stage of the processing can have operations proportional to N^2, where N is the number of discrete elements.

Table 4. Time Percentage spent on the simulation stages

Function	Time (%)
Contact Search	75,0
Numeric integration	9,0
Forces Update	9,0
Accelerations Update	2,0
Positions Update	2,0
Velocities Update	1,0

In the parallel code, as the particles are distributed among the processes, the number of contacts to be checked can be reduced to the ratio of $(N / Np)^2$, where Np is the number of processes. However, the greater is the number of process, the bigger is the communication time between them. The Figure 10 shows the time percentage that is spends in the search of contacts and in the communication among the processes for the 5000 particles case.

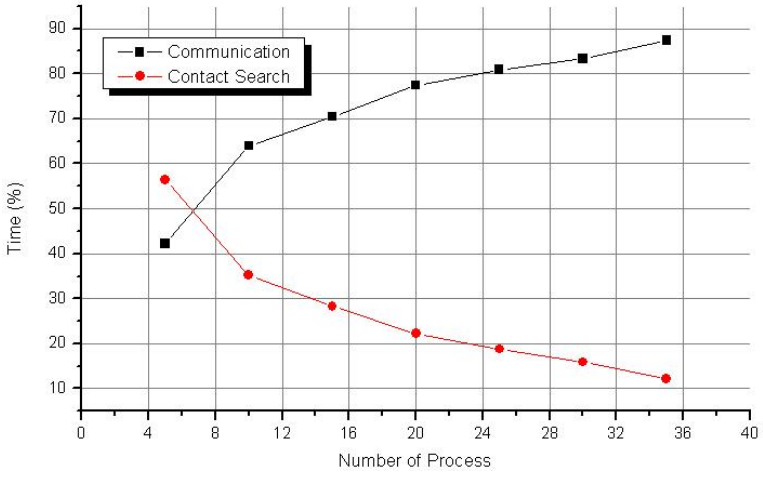

Figure 10. Time spent in Communication and Contact search.

7. CONCLUSIONS

The achieved results show a significant reduction on the execution time when the parallel implementation developed in this work is used, in comparison with the same code on its sequential version. The maximum speed-up obtained was about 16 and happened for the example in which the domain was subdivided in 10.000 particles. Although, speed-up's even greater can be obtained if problems with a bigger number of particles were simulated, because in this case the time spent in communication would be compensated by the division of a huge amount of data among the processes.

As can be observed on the graph of the Figure 10, when the number of processes increases, there is a reduction on the time of contact search, which consists on the most delayed stage of the simulation. On the other hand, the information traffic on the network increases, raising, this way, the communication time. There is a moment when the increase on the communication time doesn't compesate the processing reduction time and the speed-up starts to decline.

This Application is still being developed and some aspects that can improve even more the speed-up are being investigated. Examples with more discrete elements, such as one with one million particles, are also being evaluated.

Besides this, tests with this application are being done using effectivaly the two nucleos of the Core 2 Duo processor and using a machine with many processors and shared memory archteture.

6. REFERENCES

Cundall, P. A. and Strack, O. D. L., 1979, "A discrete numerical model for granular assemblies", Geotechnique 29, No. 1, 47-65.

Munjiza, 2004, "The Combined Finite-Discrete Element Method", John Wiley & Sons, Ltd, England.

Carvalho Jr.H. and Cintra, D.T., 2005, "Desenvolvimento de ferramentas de análise e visualização do método dos elementos discretos e suas aplicações na engenharia", Civil Engineering Course, Federal University of Alagoas.

Carvalho Jr.H., Cintra, D.T., and Silveira, E. S. S., 2006, "Desenvolvimento de um sistema orientado a objetos para análise através do Método dos Elementos Discretos", XXVII Iberian Latin American Congress on Computational Methods in Engineering, Belém, PA, Brazil.

Karniadakis, G. E. and Kirby II, R. M., 2003, "Parallel Scientific Computing in C++ and MPI", Cambridge University Press, USA.

Stephany, S., Preto, A. J., Granato E. and Ribeiro, J. A. J., "A Parallel Implementation for a drive molecular dynamics algorithm", INPE, São José dos Campos, SP, Brazil.

Cleary, P. W., Sawley, M. L., 2002, "DEM modelling of industrial granular flows: 3D case studies and the effect of particle shape on hopper discharge", Applied Mathematical Modelling 26, pp.89–111.

Cavalcante, R. B. L., Silveira, E. S. S. da, Lages, E. N. and Lira, W. W. M., 2006, "Um gerador de Elementos Discretos baseado em uma triangulação de domínio", XXVII Iberian Latin American Congress on Computational Methods in Engineering, Belém, PA, Brazil.

Amorim, J. A. de, Cintra, D. T. and Lira, W. W. M., 2006, "Uma interface gráfica-interativa para visualização dinâmica do Método dos Elementos Discretos", XXVII Iberian Latin American Congress on Computational Methods in Engineering, Belém, PA, Brazil.

7. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.