INTERNATIONAL JOURNAL FOR NUMERICAL AND ANALYTICAL METHODS IN GEOMECHANICS
*Int. J. Numer. Anal. Meth. Geomech.* (2011)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nag.1085

# Parallelization of the distinct lattice spring model

Gao-Feng Zhao[1,*,†], Jiannong Fang[2], Liang Sun[3] and Jian Zhao[3]

[1]*School of Civil and Environmental Engineering, The University of New South Wales (UNSW), Sydney, Australia*
[2]*Ecole Polytechnique Federale de Lausanne (EPFL), Laboratory of Engineering and Environmental Geology, Station 18, CH-1015 Lausanne, Switzerland*
[3]*Ecole Polytechnique Federale de Lausanne (EPFL), Laboratory of Rock Mechanics, Station 18, CH-1015 Lausanne, Switzerland*

## SUMMARY

The distinct lattice spring model (DLSM) is a newly developed numerical tool for modeling rock dynamics problems, i.e. dynamic failure and wave propagation. In this paper, parallelization of DLSM is presented. With the development of parallel computing technologies in both hardware and software, parallelization of a code is becoming easier than before. There are many available choices now. In this paper, Open Multi-Processing (OpenMP) with multicore personal computer (PC) and message passing interface (MPI) with cluster are selected as the environments to parallelize DLSM. Performances of these parallel DLSM codes are tested on different computers. It is found that the parallel DLSM code with OpenMP can reach a maximum speed-up of 4.68× on a quad-core PC. The parallel DLSM code with MPI can achieve a speed-up of 40.886× when 256 CPUs are used on a cluster. At the end of this paper, a high-resolution model with four million particles, which is too big to handle by the serial code, is simulated by using the parallel DLSM code on a cluster. It is concluded that the parallelization of DLSM is successful. Copyright © 2011 John Wiley & Sons, Ltd.

Received 30 August 2010; Revised 24 June 2011; Accepted 28 June 2011

KEY WORDS: parallelization; 3D model; lattice spring model; OpenMP; MPI

## 1. INTRODUCTION

Dynamic fracturing of heterogeneous materials such as rock and concrete cannot be modeled realistically without appealing to their microstructures. However, the classical mesh-based numerical methods, e.g. FEM, are difficult to use in building such kind of complex geometry model, especially in three-dimensional (3D) cases. The distinct lattice spring model (DLSM) [1] is a 3D particle-based model proposed to solve dynamic fracturing problems involved in rock mechanics and rock engineering. DLSM has advantages of representing the diversity of Poisson's ratio, directly using macroscopic parameters and allowing general lattice structures to be adopted. To solve engineering problems at large scales, a large number of particles have to be used. In this case, the simulation becomes demanding in terms of memory and computing time and cannot be performed using a normal personal computer (PC). However, this problem can be overcome through parallel implementation of the DLSM code, which will be presented in this paper. The basic idea of parallelization is to distribute computations to several processors and to execute the distributed works simultaneously. The implementation of a parallel code is much different from that of a serial code. Fortunately, with the development of technologies in computer science, this is becoming easier and easier. So far, there exist three popular choices for parallel computing. The first choice is the multicore PC. The quad-core CPU

*Correspondence to: Gao-Feng Zhao, School of Civil and Environmental Engineering, The University of New South Wales (UNSW), Sydney, Australia.
†E-mail: gaofeng.zhao@unsw.edu.au

is very common now, and even the 80-core CPU prototype has already been developed [2]. So performing parallel computation in PC is no longer just a dream. Graphics processing unit (GPU) computing [3] is the second choice. It has been reported that more than 100× speed-up is achieved by using GPU for some applications [4]. The last choice is the computer cluster, which is available in many universities and research institutes. Cluster is a high-level parallelization system [5] that is made of many computer nodes (each node could be a multicore or GPU computer). In this paper, instead of giving a verbose review on the parallel computer history like the classification made by Flynn in 1966 [6], a review on the three parallel computer systems mentioned above and the corresponding software development environments will be presented. The reason is that these three choices are the currently available and popular solutions for parallelization implementation.

Personal computer refers to any general-purpose computer whose size and capabilities are small and whose price is low enough to make it acceptable to individuals. PC is also called a microcomputer, which means that its computing power is much less than a supercomputer. However, with the development of computer hardware and software, the PC nowadays becomes the dominant tool in performing scientific computing and numerical modeling. The main reason is that the application software and operation system in the PC are much friendly to users. Another reason is that with the improvement of the CPU and the memory used in PCs, some engineering problems can also be solved on a normal PC. For example, a laptop equipped with a 2-GHz CPU and 2GB of memory is enough to run DLSM with a half million particles. Recently, a new term, personal high-performance computing (PHPC) [7], is proposed. PHPC aims to use a normal PC in running problems that previously could only be handled on a supercomputer. This may become true in the near future if the 80-core CPU and the 64-bit operation system are mature enough. The future PC, equipped with an advanced multicore processor, will surely provide adequate computing power and memory space for scientific computing. The multicore processor aims at providing better performance. It includes multiple execution units, and the instructions per cycle can be executed separately in different cores. The typical structure of a quad-core processor is shown in Figure 1. The advantage of the multicore PC is that it can handle multiple tasks at the same time. The amount of gained performance by using the multicore processor is strongly dependent on the code implementation. Many typical applications, however, do not consider parallelization on a multicore PC, which remains to be an important on-going topic for research. Fortunately, parallel programming environments such as OpenMP [8], pThreads [9] and threading building blocks [10] can be used to implement the multicore version of an existing code. Normally, the parallelization of a code on a multicore PC is relatively simple, as it only needs to deal with the shared memory environment. It does not need to consider the task distribution and communication between different processors. For multicore PCs, the OpenMP has widely been used to parallelize different numerical codes; for example, Tarmyshov and Muller-Plathe [11] applied it to parallelize the molecular dynamics (MD) for chemical simulation, Zsaki [12] used it to speed up the generation process of the discrete element method (DEM) for geomechanics, Gao and Schwartzentruber [13] applied it on the Monte Carlo simulation for fluid mechanics and Williams et al. [14] adopted it to parallelize smoothed particle hydrodynamics for fluid flow problems. However, there also exist some

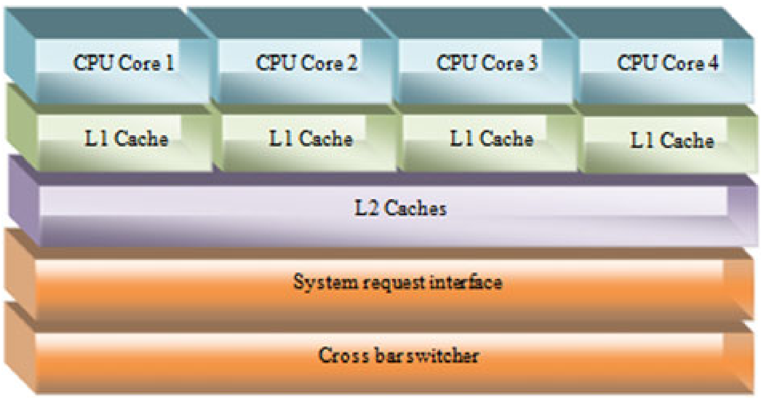


Figure 1. The diagram of a generic quad-core processor.

disadvantages of a multicore processor [15,16]. First, adjustments of the existing code are required to allow maximum utilization of the computing resources. Second, it is more difficult to manage the thermal problem than using a single-chip design. Third, a multithread code often requires complex coordination of threads and makes it difficult to find bugs. Moreover, the interaction between different threads can also cause safety problems. Even so, our experience tells us that parallel computing using a multicore processor is stable and promising, at least for research purpose. In this paper, the multicore implementation of DLSM will be presented, and its performance will be tested in multicore PCs.

Recently, GPU computing [3] is becoming an interesting topic in high-performance computing. The most attractive aspects of this new technology are the extremely high speed-up for some scientific computing problems and the price of a GPU computer system, which is much cheaper than a supercomputer. GPU was originally used as a specialized processor to deal with 3D graphics rendering. Very recently, a new concept, general purpose GPUs, was proposed to allow the GPU to perform massive floating-point computing. The basic idea of GPU is to put a large number of specified computing units on a single board and interpret hundreds of thousands of threads. These threads can deal with the calculation simultaneously. The architecture of a typical GPU computing card [17] is shown in Figure 2. It has 128 thread processors, and each thread processor has a single-precision floating-point unit and 1024 registers. These thread processors could process different data at the same time. The framework of memory communication is also different from the conventional parallel computer and could largely increase the parallel efficiency; for example, 100× speed-up is achieved when the shared memory scheme is used [3]. There are different kinds of available tools for developing a GPU-based code. They are OpenGL[18], OpenCL [19] and some special programming toolkits for GPU computing which are listed by Elsen et al. [20] as Sh (Michael McCool, University of Waterloo), Brook (Pat Hanrahan, Stanford University), CUDA (NVIDIA), and CTM (AMD). These toolkits provide a very useful environment for the development of a GPU code. They have been widely used by researchers; for example, Anderson et al. [21] used CUDA to develop GPU-based MD code, Elsen et al. [20] adopted Brook to implement a large-scale computation code for fluid flow, Takahashi and Hamada [22] applied CUDA to accelerate BEM for the 3D Helmholtz equation and Joldes et al. [23] presented the application of a CUDA-based nonlinear FEM for real-time simulation of the neurosurgical process. Normally, the implementation of a GPU code will require certain knowledge of the operation of GPU at hardware layer. Overall, GPU computing is a sheared memory system, and it is a promising solution for PHPC. It also provides a solution for real-time numerical simulation. However, the hardware and software platforms of GPU computing are still under development. For example, the double precision GPU card will be available in a few months, and CUDA

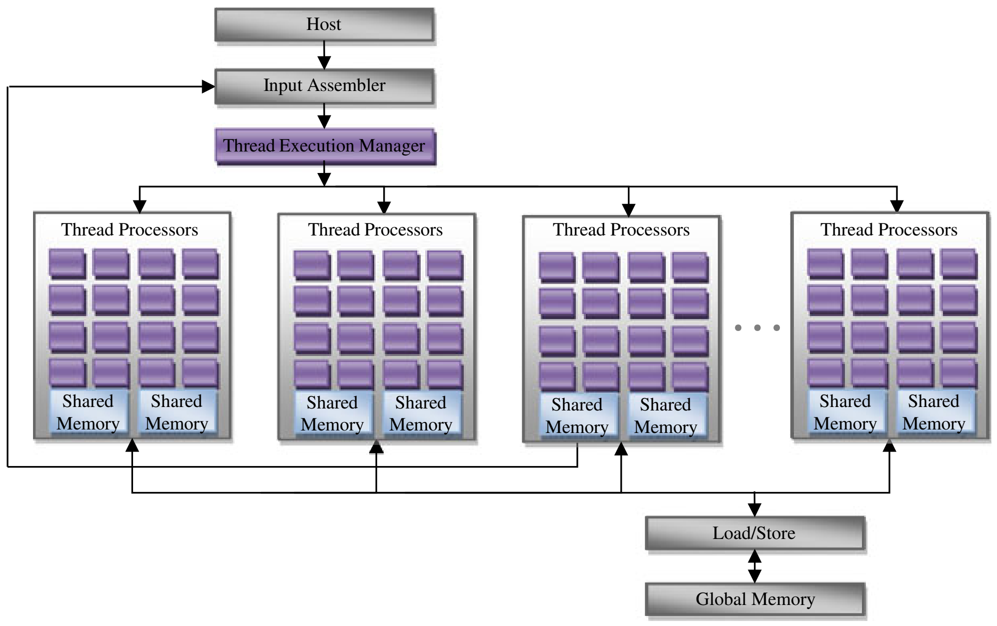


Figure 2. The NVIDIA GeForce 8 graphics processor architecture (redrawn based on [17]).

will support C++ in the future. Taking into account this delay, waiting for the technique to mature is still a good choice.

Modern supercomputer often refers to a computer cluster, which is a collection of computers connected through a high-speed network. A cluster computer is a high-level parallelization system. The most powerful computers in the world always involve clusters [24]. Newly developed technologies on high-performance computing (e.g. multicore CPU and GPU) can always be merged into a cluster system. For example, the fifth-ranked Tianhe-1 supercomputer has integrated multicore CPUs with GPUs [24]. The architecture of clusters is normally based on a modular concept, which can be simply regarded as a group of specific computers connected through the Internet for working together. For example, the cluster used in this work, Pleiades2 [25] at Ecole Polytechnique Federale de Lausanne (EPFL), is built on a gigabit Ethernet network, as shown in Figure 3. The message passing interface (MPI) [26] and the parallel virtual machine [27] are programming tools for parallelization implementation under a cluster environment. In this study, the free MPI library MPICH (developed at Argonne National Lab) [28] was used. There are different kinds of parallel numerical codes for geomechanics that are developed under a cluster environment using MPI; for example, Maknickas et al. [29] implemented a parallel DEM code for modeling granular media, Singh and Jain [30] developed a parallel meshless method EFG code for fluid flow simulation, Komatitsch et al. [31] designed a high-order FEM parallel code for seismic wave propagation and Wang et al. [32] used MPI to parallelize the FEM for thermo-hydro-mechanical–coupled problems in porous media. Recently, some researchers also try to combine different parallel techniques; for example, Tang et al. [33] coupled MPI with OpenMP for groundwater simulation and Yang et al. [34] proposed a hybrid CUDA, OpenMP and MPI programming approach. As a cluster is a distributed memory system, the model decomposition and communications between different processors (computer nodes) should be handled explicitly. Thus, a certain amount of modification of the original DLSM code is required. Moreover, the operation system used in a cluster is different from that in a PC. For example, a standard SUSE Linux is used as the operation system in a Pleiades2 cluster. How to integrate different operation systems in a PC and in a cluster is also a problem facing the parallelization of DLSM.

In this paper, DLSM will be parallelized both for a multicore PC based on OpenMP and for a cluster based on MPI. After a brief description of DLSM, the implementation of the parallel DLSM under different platforms will be presented. Then, the performance of the parallel DLSM codes will be tested on different computers. Finally, some conclusions on the parallelization of DLSM will be derived.

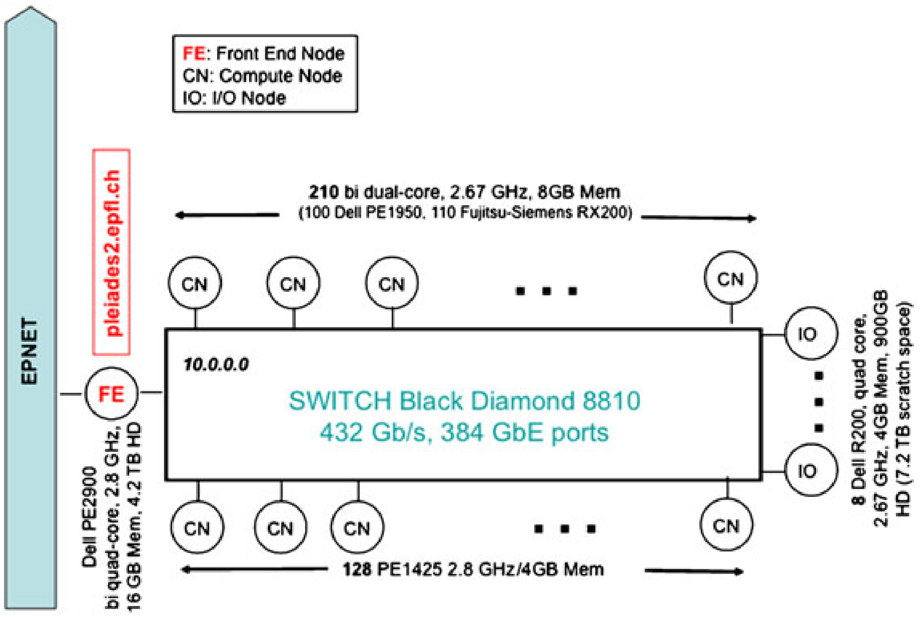


Figure 3. Current configuration of the Pleiades2 cluster from EPFL [25].

## 2. DLSM

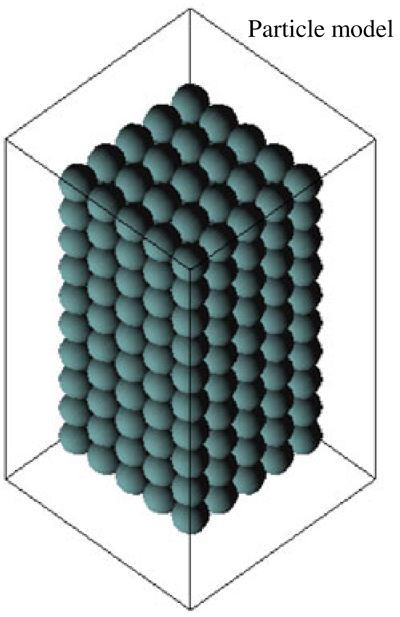
The origin of lattice spring model (LSM) may be traced back to the work of Hrennikoff [35] in 1941. However, the method had little success due to the limitation of computer power during those times. In recent years, researchers have renewed their interest in LSM modeling techniques, as they found that these models are very suitable for the modeling of failure (e.g. [36,37]). It is known that Poisson's ratio obtains, by classical LSM, in the limit of an infinite number of particles, a fixed value, which is 0.25 for 3D cases and 0.33 for 2D cases. Recently, this restriction is solved in DLSM [1,38], in which shear springs based on local strain rather than the particle displacements are introduced to keep the rotational invariance. DLSM has the following characteristics: (i) materials are discretized into particles that are connected through spring-type forces; (ii) the macro-mechanical response is derived from the microscopic interactions between particles; (iii) the material failure at the continuous level is captured naturally from the spring failure at the microdiscontinuous level; and (iv) complex constitutive relationship and contact mechanisms are readily implemented.

### *2.1. Physical model and system equations*

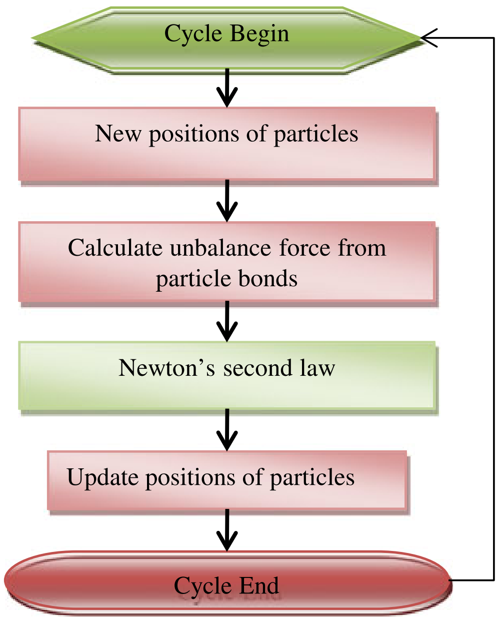
In DLSM, the material is discretized into mass particles with different sizes. The two particles are linked together through a bond between their center points (as shown in Figure 4(a)), which consists of normal and shear springs. The bond will turn into contacts between the particles when failure happens. The particles and bonds form a network system representing the material. For this system, its equation of motion can be expressed as

$$[\mathbf{K}]\mathbf{u} + [\mathbf{C}]\underline{\dot{\mathbf{u}}} + [\mathbf{M}]\,\ddot{\mathbf{u}} = \mathbf{F}(t) \tag{1}$$

where **u** represent the vector of particle displacement, [**K**] the stiffness matrix, [**M**] the diagonal mass matrix, [**C**] the damping matrix, and **F**($t$) the vector of external force. Equation (1) is solved by using Newton's second law. The calculation cycle is illustrated in Figure 4(b). Given the particle displacements, new contacts and broken bonds are detected. The list of neighboring particles for each particle is updated. Then, contact and spring forces between particles are calculated according to the prescribed force–displacement relations. The particle velocity is advanced individually as



(a) The physical model of DLSM


Cycle Begin
New positions of particles
Calculate unbalance force from particle bonds
Newton's second law
Update positions of particles
Cycle End

(b) Calculation cycle

Figure 4. The physical model and the calculation cycle of DLSM.

$$\dot{\mathbf{u}}_i^{(t+\Delta t/2)} = \dot{\mathbf{u}}_i^{(t-\Delta t/2)} + \frac{\sum \mathbf{F}_j^{(t)}}{m_p}\Delta t \tag{2}$$

where $\dot{\mathbf{u}}_i^{(t+\Delta t/2)}$ is the particle velocity at $t+\Delta t/2$, $\dot{\underline{\mathbf{u}}}_i^{(t-\Delta t/2)}$ the particle velocity at $t-\Delta t/2$, $m_p$ the particle mass, $\sum \mathbf{F}_j^{(t)}$ the sum of forces acting on the particle $i$ including applied external forces, and $\Delta t$ the time step. Finally, the new displacement of particle is obtained as

$$\mathbf{u}_i^{(t+\Delta t)} = \mathbf{u}_i^{(t)} + \dot{\mathbf{u}}_i^{(t+\Delta t/2)}\Delta t \tag{3}$$

where $\mathbf{u}_i^{(t+\Delta t)}$ is the displacement at $t+\Delta t$ and $\mathbf{u}_i^{(t)}$ the displacement at $t$. It should be mentioned that the calculation cycle in Figure 4(b) is performed independently for all the particles. This feature means that the model is very suitable for parallelization.

## 2.2. *Interactions between particles*

Figure 5(a) shows the forces exerted on one particle. These forces are made up of the external force and contact force between particles. The interaction between linked particles is represented by one

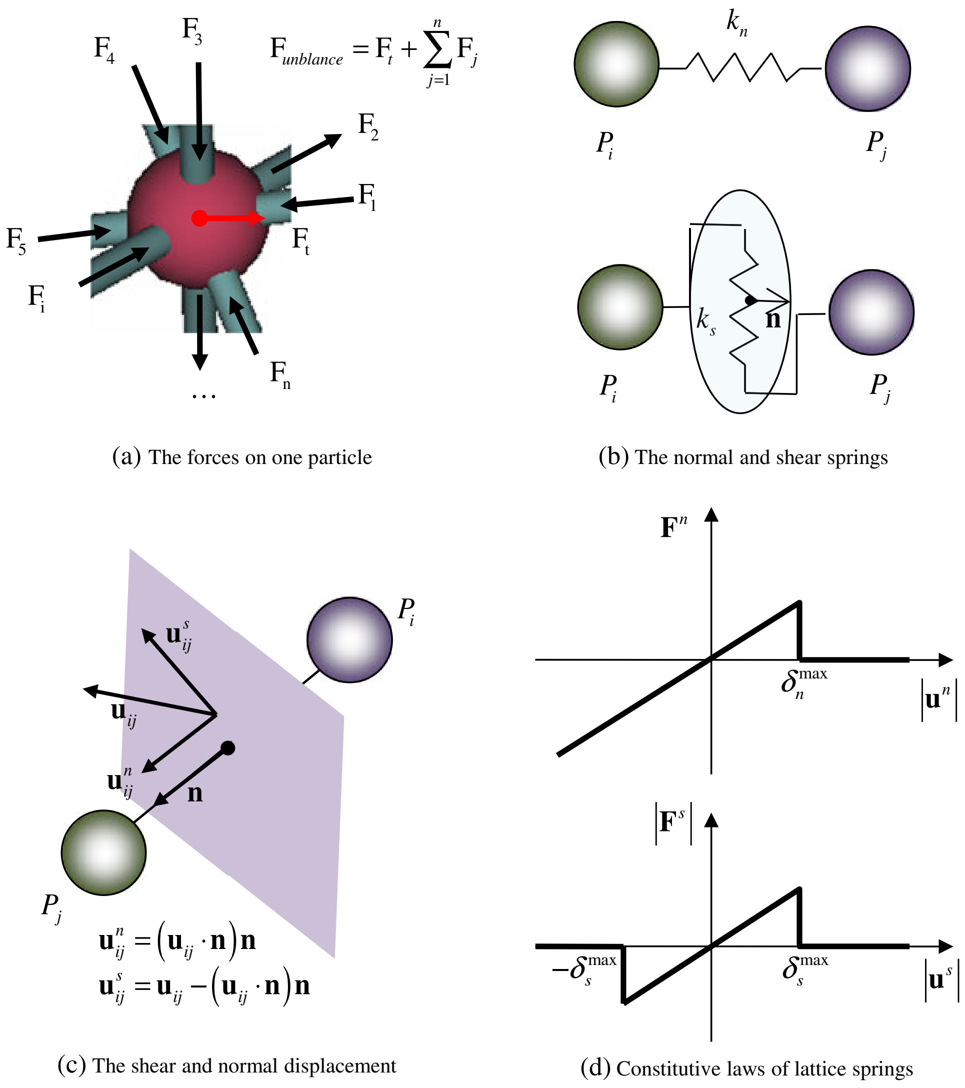

$F_{unblance} = F_t + \sum_{j=1}^{n} F_j$
F4
F3
F2
F1
Ft
F5
Fi
Fn
...
(a) The forces on one particle
$k_n$
$P_i$
$P_j$
$k_s$
n
$P_i$
$P_j$
(b) The normal and shear springs
$P_i$
$\mathbf{u}_{ij}^s$
$\mathbf{u}_{ij}$
$\mathbf{u}_{ij}^n$
n
$P_j$
$\mathbf{u}_{ij}^n = (\mathbf{u}_{ij}\cdot\mathbf{n})\mathbf{n}$
$\mathbf{u}_{ij}^s = \mathbf{u}_{ij} - (\mathbf{u}_{ij}\cdot\mathbf{n})\mathbf{n}$
(c) The shear and normal displacement
$\mathbf{F}^n$
$\delta_n^{\max}$
$|\mathbf{u}^n|$
$|\mathbf{F}^s|$
$-\delta_s^{\max}$
$\delta_s^{\max}$
$|\mathbf{u}^s|$
(d) Constitutive laws of lattice springs

Figure 5. The force and displacement relationships between two particles and the microconstitutive laws.

normal spring and one shear spring, as illustrated in Figure 5(b). The normal force between the two particles is defined as

$$\mathbf{F}_{ij}^{n} = k_n \mathbf{u}_{ij}^{n} \tag{4}$$

where $k_n$ is the stiffness of the normal spring and $\mathbf{u}_{ij}^{n} = (\mathbf{u}_{ij} \bullet \mathbf{u})\mathrm{n}$ is the vector of normal displacement, the normal unit vector $\mathbf{n}{=}(n_x, n_y, n_z)^{\mathrm{T}}$ pointing form particle $i$ to particle $j$ and $\mathbf{u}_{ij}{=}\mathbf{u}_j - \mathbf{u}_i$ (see Figure 5(c)). Then, the shearing force between the two particles reads

$$\mathbf{F}_{ij}^{s} = k_s \hat{\mathbf{u}}_{ij}^{s} \tag{5}$$

where $k_s$ is the stiffness of the shear spring and $\hat{\mathbf{u}}_{ij}^{s}$ is the shear displacement vector obtained from a local strain-based method to keep it rotationally invariant as

$$\hat{\mathbf{n}}_{ij}^{s} = [\boldsymbol{\varepsilon}]_{bond} \cdot \mathbf{n}l \text{-} \left(\left([\boldsymbol{\varepsilon}]_{bond} \cdot \mathbf{n}l\right) \cdot \mathbf{n}\right)\mathbf{n} \tag{6}$$

where $[\boldsymbol{\varepsilon}]_{bond}$ is the strain state of the bond and $l$ is the bond length. In DLSM, the local strain of one particle is evaluated by a least-square scheme that only uses its displacement and other particles that have intact bonds with the particle (see Figure 5(a)). By doing so, discontinuities (e.g. fracture/crack) could be directly considered without using the 'visibility criterion' adopted by most meshless methods. Details and validation examples of DLSM on elastic and elastic dynamic problems can be found in [1,38].

## 2.3. *Constitutive law in DLSM*

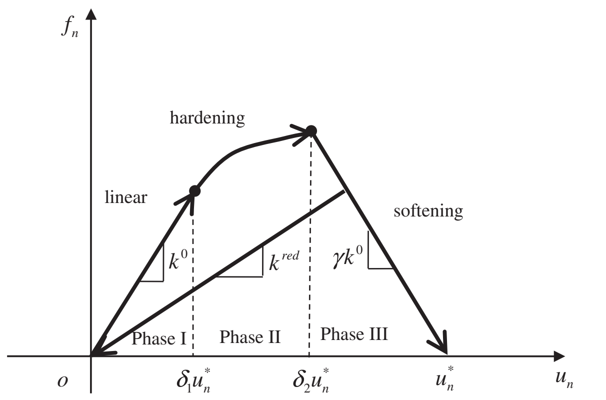
Constitutive law for the bond spring in DLSM can be the simplest brittle linear one (see Figure 5(d)). Yet, it is not enough to describe the complex mechanical behavior of rock material. A more complex microconstitutive law can be implemented easily, as presented in [38]. Firstly, consider the force–deformation relationship of the normal spring satisfying the curve shown in Figure 6(a), where $u_n$ represents the normal deformation of the bond spring, $u_n^*$ is the ultimate deformation, $\delta_1$ is the ratio of the deformation at the hardening point to the ultimate deformation, and $\delta_2$ is the ratio of the deformation at the softening point to the ultimate deformation. It can be seen that the curve can fully represent the linear stage, the hardening stage and the softening stage of the normal bond spring. Instead of directly providing the force–displacement relationship, a damage variable function is defined as

$$D(u_n) = 1 - \frac{\bar{k}(u_n)}{k_0} \tag{7}$$

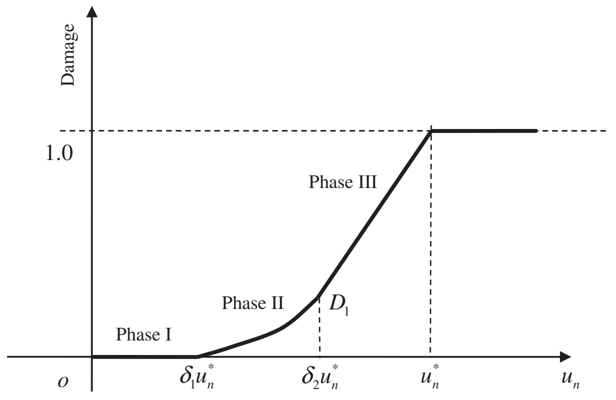
where $k_0$ is the initial stiffness and $\bar{k}(u_n)$ is the secant modulus when the bond deformation is $u_n$. The damage variable is initially equal to zero when the spring is intact and finally turns to 1 when the spring is totally broken. The damage variable function corresponding to Figure 6(a) is shown in Figure 6(b). Given a damage variable function, the force–displacement relationship can easily be obtained as

$$f(u_n) = (1 - D(u_n))k_0 u_n \tag{8}$$

where $f(u_n)$ is the spring interaction force when the spring deformation is $u_n$. Different microconstitutive laws can be realized by developing different damage variable functions. In this section, displacement is used as the synonym of deformation. Microparameters $u_n^*, \delta_1, \delta_2,$ and$K^{red}$ are selected to identify the damage variable function for the normal spring. $K^{red}$ is the ratio of secant modulus at the softening point to the initial stiffness. Damage variable functions are constructed based on these parameters.



(a) Force-deformation curve of normal spring



(b) Damage variable function

Figure 6. The force–deformation relationship and the damage variable function for the normal spring.

For example, a trilinear microconstitutive law for the normal spring is given as

$$D(u_n) = D^{'}\left(\frac{u_n^*}{u_n}\right) = D^{'}(\alpha) = \begin{cases} 0 & 0\leqslant\alpha\leqslant\delta_1 \\ 1-\alpha\delta_1-\left(K^{red}\delta_2-\delta_1\right)(1-\alpha\delta_1)/(\delta_2-\delta_1) & \delta_1<\alpha\leqslant\delta_2 \\ 1-K^{red}\delta_2(\alpha-1)/(1-\delta_2) & \delta_2<\alpha\leqslant 1 \end{cases} \tag{9}$$

where $\alpha$ is introduced to simplify the formulation of the equation. Equation (9) can be rewritten in the force–displacement form as

$$f = \begin{cases} k_0 u_n & u_n\leqslant\delta_1 u_n^* \\ k_0 u_n^*\delta_1 + \dfrac{\left(k_{red}u_n^*\delta_2 - k_0 u_n^*\delta_1\right)\left(u_n - u_n^*\delta_1\right)}{u_n^*\delta_2 - u_n^*\delta_1} & \delta_1 u_n^* < u_n\leqslant\delta_2 u_n^* a \\ k_{red}u_n^*\delta_2\dfrac{u_n^*-u_n}{u_n^*-u_n^*\delta_2} & \delta_2 u_n^* < u_n\leqslant u_n^* \end{cases} \tag{10}$$

where $k_{\text{red}}=k_0K^{red}$. Assuming $k_0=1$, the force–displacement relationship given by Equation (10) is plotted in Figure 7. Different constitutive models can be obtained by setting $K^{red}$ to different values (see Figure 7). $K^{red}$ is a dimensionless parameter, which can be regarded as the secant stiffness at the softening point when $k_0=1$. The brittle linear constitutive law is the special case of the trilinear constitutive law when $\delta_1=\delta_2=1.0$ and $K^{red}=0$. The widely used bilinear constitutive law is obtained when $\delta_2=1.0$ and $K^{red}=0$. Using the damage variable function, nonlinear microconstitutive law has also been developed [38]. An example is given as follows:

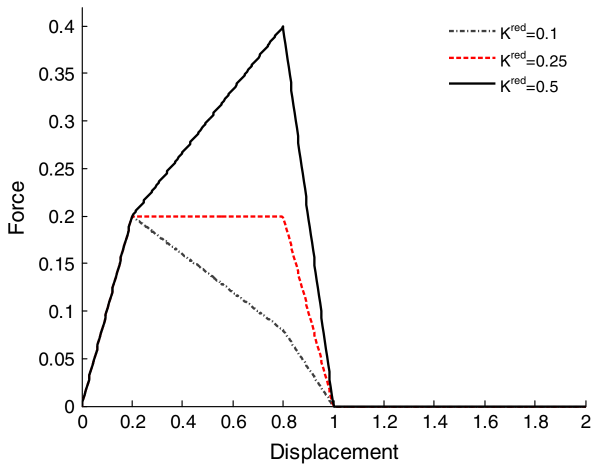


Figure 7. Force–displacement curve of the trilinear constitutive law under different values of $K^{red}$.

$$D(u_n)=D'\left(\frac{u_n}{u_n^*}\right)=D'(\alpha)=\begin{cases}0 & 0\leqslant\alpha\leqslant\delta_1\\ 1-\alpha\delta_1-\beta\alpha\left(K^{red}\delta_2-\delta_1\right)de^{1+d} & \delta_1<\alpha\leqslant\delta_2\\ 1-(1-\beta)\alpha\delta_1-\beta\delta_2K^{red}(\alpha-1)/(1-\delta_2) & \delta_2<\alpha\leqslant 1\end{cases}\qquad(11)$$

where $\beta$=0.3 and $d$=($\alpha-\delta_1$)/($\delta_2-\delta_1$). The corresponding force–displacement relationship for $k_0$=1 is shown in Figure 8.

The constitutive law for shear spring can be introduced following the same damage variable function and the corresponding nondimensional parameters as those for the normal spring. The proposed nonlinear constitutive laws can take into account the nonlinear behavior of bonds in DLSM. The influence of these parameters of the microconstitutive model on the final macromechanical behavior of DLSM was presented in [38]. Yet, it is found that the ratio of compressive strength to tensile strength for DLSM is lower than that for rock materials (typically around 10–12). It is around seven for the regular lattice model and three for the random lattice model. A similar problem was observed in the particle-based DEM code. The reason is attributed to the smooth shape of circular particles [39]. In this paper, the developed high-performance DLSM will also be used to provide a microscopic explanation of this phenomena.

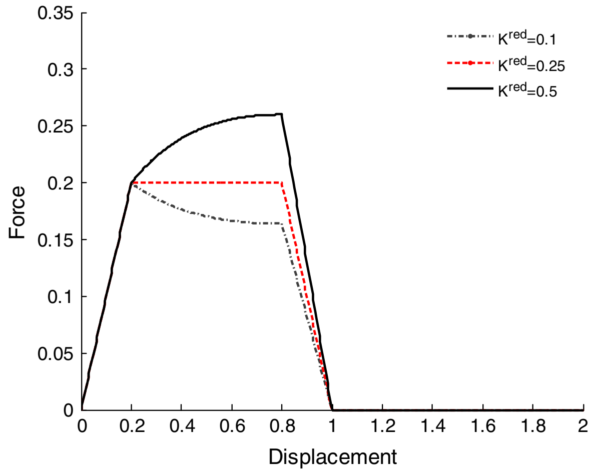


Figure 8. Force–displacement curves of the nonlinear constitutive law under different values of $K^{red}$.

## 3. PARALLELIZATION OF DLSM

In DLSM, Equations (2) to (6) are the main computational cost procedures. Thus, in order to parallelize DLSM, we need to have these equations calculated in parallel. In this article, two parallelization methods (OpenMP and MPI) are used to realize this task.

### *3.1. Multicore DLSM*

This section will present the parallel implementation of the DLSM code based on OpenMP. The motivation is to reduce computational time on multicore PCs. As DLSM is an explicit method in time, only minor changes are needed to parallelize the code. Quad-core PC is quite common now, but the serial code cannot efficiently utilize its computing resources. OpenMP provides a useful tool to parallelize software for a multicore environment. It is an application program interface that is composed of compiler directives, runtime library routines and environment variables. It can work under the compiler environments of FORTRAN, C and C++. A fork-joint model is used in OpenMP to parallelize a task. Hereafter, the parallel DLSM code based on OpenMP is named the multicore DLSM. As the shared memory strategy is used in multicore DLSM, Equations (2) to (6) can be simply parallelized without communication between different CPUs.

The work scheme of the serial and multicore DLSM are shown in Figure 9. It can be seen that the serial DLSM code has only one main thread and that the force and displacement of particles are calculated sequentially (as shown in Figure 9(a)). The multicore DLSM uses the fork-joint model to let one cycle be calculated by more than one processor (see Figure 9(b)). The parallel DLSM works as follows. Firstly, the master thread is activated when DLSM begins execution. Then, when the master thread executes the points where parallel operations are required, the master thread forks and additional threads are used to realize parallel computing.

In multicore DLSM, the force calculation and the displacement update are the only procedures needed to be parallelized. Only a few macros are added to produce a fork for a single loop. For example, the code

```
for(int i=0;i<N;i++)
{
    calculate_the_particle_force/displacement;
}
```

can be easily OpenMP-paralleled as

```
int i;
#pragma omp parallel for
for(i=0;i<N;i++)
{
    calculate_the_particle_force/displacement;
}
```

It can be seen that only a few modifications are made for the OpenMP implementation. These modifications can greatly speed up the computational performance of the original DLSM code. Thus, from an input point of view, the achieved benefits are actually very attractive. It is important to fully utilize the cache memory in the OpenMP implementation. In multicore DLSM, it is roughly achieved by using local variables for the massive complex operation, i.e. the local strain computation.

### *3.2. Cluster DLSM*

The multicore DLSM aims at the full utilization of the computing resources on a multicore PC. Although the 80-core CPU already exists in prototype and may be available for practical usage in the near future, the limitation on available cores and memory in a normal PC cannot be removed completely. Speed-up of the multicore DLSM shall be limited eventually. Moreover, the shared memory strategy also limits the modeling capability of the multicore DLSM. In this section, the

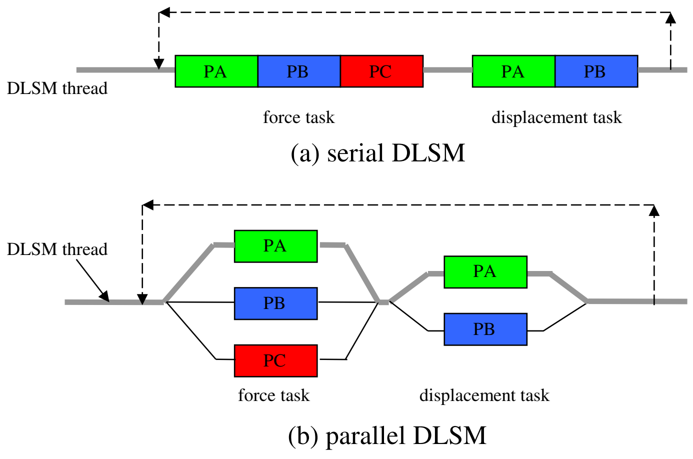


Figure 9. Scheme of serial and parallel implementation of DLSM.

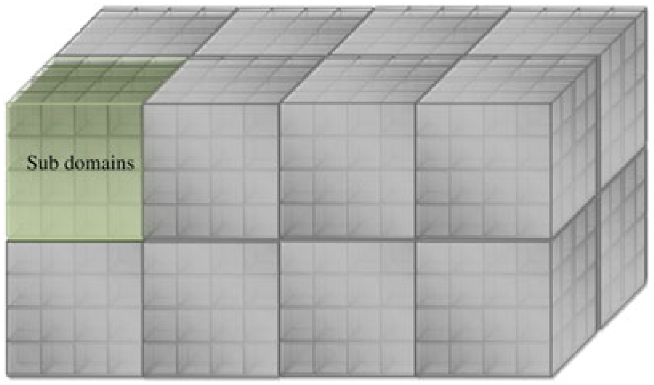


Figure 10. Decomposition of the simulation domain $\Omega$ into 16 subdomains.

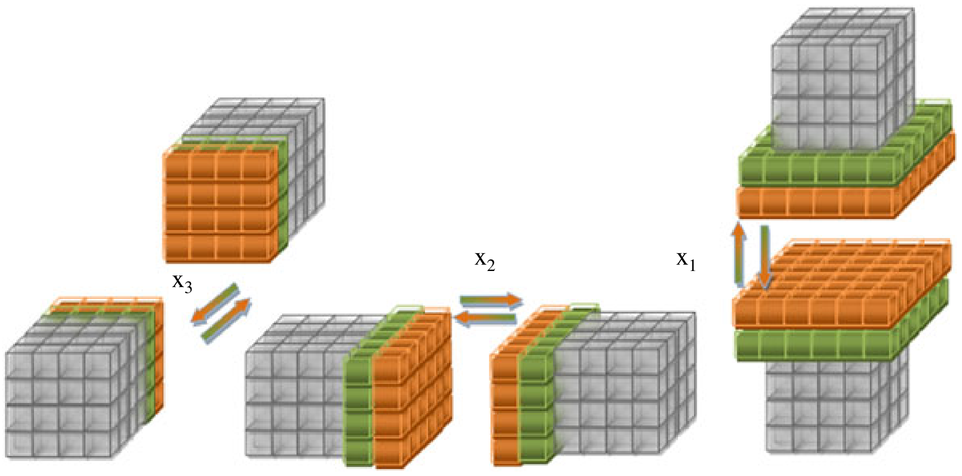
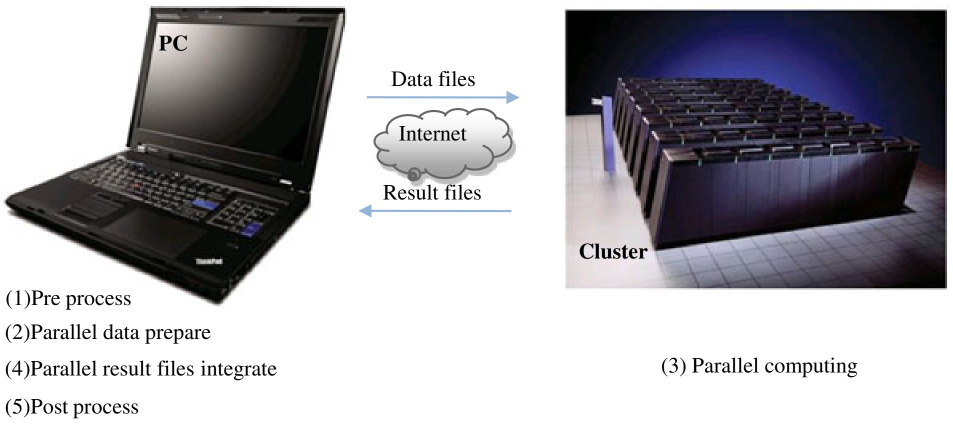


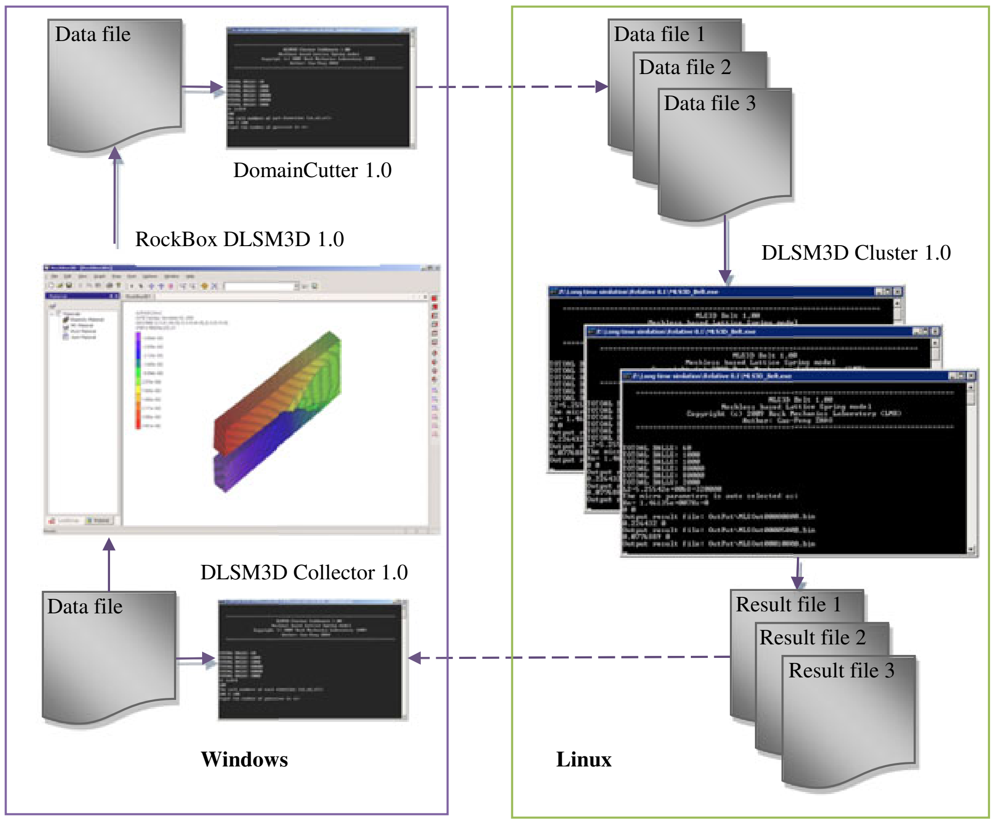
Figure 11. Communication scheme used in cluster DLSM.

MPI-based parallelization of DLSM on cluster will be presented to solve these problems. It is named the cluster DLSM in order to distinguish it from the previous one for multicore PC.

*3.2.1. Parallelization strategy.* The domain decomposition is used as the parallelization strategy for the cluster DLSM. Firstly, the simulation domain is divided into many small cubic cells (see Figure 10). Each cell contains a list of the particles under it. Secondly, the simulation domain is divided into a number of subdomains (larger cubes) based on these small cubes. Each subdomain contains a number of small cubes. Particles in each subdomain are distributed to a processor to be



(a) work flow at hardware layer



(b) work flow at software layer

Figure 12. Work flow of parallel DLSM under a cluster enviroment.

calculated separately from the others. This scheme is called the linked cell method in MD parallelization [40,41]. The number of subdomains is equal to the number of processors used in the simulation. We have the following relation:

$$np = np_x \times np_y \times np_z \tag{12}$$

where $np$ is the number of total processors (subdomains) and $np_x$, $np_y$, $np_z$ are the number of dividing in each direction of the model. A decomposition code (*domain cutter*) is designed to produce, for each subdomain, the data files, the information of corresponding neighbors and the index of particles needed to be communicated. The decomposition can be finished automatically after $np_x$, $np_y$, $np_z$ are given.

In cluster DLSM, the force calculation procedure has to use the information of the particles that do not belong to the current processor. Communication is needed to exchange the necessary information between different processors. In a 3D case, a typical subdomain has 26 neighbors. This will cause a large number of communication operations to be performed. By using a proper communication methodology [41], this number can be reduced to six. The communication strategy is shown in Figure 11. First, the data are exchanged in x3 direction (left); then, the data are exchanged in x2 direction (middle), and finally, the data are exchanged in x1 direction (right). In Figure 11, green cells always send the data to the yellow cells, and the yellow cells always receive the data from the green cells. Data are exchanged mutually between two neighboring face-to-face subdomains. The range of exchanging cells in each direction is different so that communication between the corner-to-corner neighbors is avoided. It can be seen that only six sending and receiving operations need to be performed. The exchanged data include position, velocity, displacement and strain state of the neighbor particles.

*3.2.2. Implementation.* In this section, the MPI implementation of DLSM on cluster will be presented. The parallel implementation includes not only the MPI communication part but also the model preprocessing, solving and postprocessing. Figure 12 shows the work flow of cluster DLSM. Since the PC is user-friendly and the cluster is much powerful in computing, the basic idea of this design is to let the PC deal with the preprocessing and postprocessing parts while the cluster deals with the solving part. At the hardware layer, a server/client mode is used. The PC is used as the client, and the cluster is used as the server for parallel computing (see Figure 12(a)). At the software layer, the computing task is done through cooperation between the different codes running in Windows and Linux operating systems (see Figure 12(b)). Firstly, the input data files are prepared by using a GUI program (RockBox DLSM3D) developed for Windows. When these data files are ready, they are sent to the cluster through a network. Then, the parallel DLSM solver in the cluster reads these files, solves the problem and produces the corresponding result files. Finally, the result files are copied to the PC through the network and are transformed by a postprocessing code (DLSM3D Collector) into a format that can be processed on the PC using RockBox DLSM3D. This design makes the whole parallelization work focus only on the MPI implementation of the solver. The preprocessor and postprocessor still use the serial version developed for PC. By doing so, the respective advantages of different machines (cluster and PC) and different operation systems (Windows and Linux) are fully utilized.

In the following discussion, the MPI implementation of DLSM will be presented. The goal is to run DLSM on a number of allocated processors in cluster through the domain decomposition approach. Data communication between different processors is realized through an MPI programming environment. MPI provides a library that allows the simultaneous initiation of a given number of processes and the assigning of a unique identity number for each process. It also provides communication functions that can be called to exchange the data between different processors. There are more than one hundred functions provided in the MPI library. Fortunately, the parallelization of DLSM only uses seven of them. They are MPI_Init, MPI_Comm_size, MPI_Comm_rank, MPI_Barrier, MPI_Isend, MPI_Recv and MPI_Finalize. A few modifications are needed for the parallelization of DLSM based on these MPI functions. For the main function of the DLSM code, three MPI functions are used as follows:

```
int myid, numprocs
double start,finish
MPI_Init(&argc,&argv)
MPI_Comm_rank(MPI_COMM_WORLD,&myid)
MPI_Comm_size(MPI_COMM_WORLD,&numprocs)
DLSM_Main_Function
MPI_Finalize()
```

The other two important MPI functions for data communication are used as

```
double pBuffer [nCount];
MPI_Recv(pBuffer, nCount, MPI_DOUBLE, iD_Send, iTag, MPI_COMM_WORLD, &status)
Read_Buffer_Data_To_Model;
```

```
double pBuffer [nCount];
Read_Buffer_From_Model;
MPI_Isend(pBuffer, nCount, MPI_DOUBLE, iD_Receive, iTag, MPI_COMM_WORLD, &request);
```

When the force calculation procedure is executed in cluster DLSM, particle information will be exchanged between processors by using the above subroutines according to the communication scheme shown in Figure 11. Currently, the case of a particle moving out of the present processor and entering into another processor is not considered because the communication of bond information between different processors is difficult. Problems involving dynamic contact detection can also be solved if the relative deformation between any two neighboring subdomains is not too large compared to the size of the cell, so that it would be sufficient to use only the neighboring particles as the cushion layer between the two subdomains.

In cluster DLSM, contact detection and particle position update, failure treatment and results output will be processed separately for different processors. During the calculation, each process outputs its own results to a separate file, which is identified by the process number. These files can be combined into a single file and be postprocessed on a PC.

## 4. PERFORMANCE EVALUATION

In this section, the different parallel DLSM codes are tested on different parallel computers. There are a large number of commonly used performance measures in evaluating a parallel code. In this article, the speed-up $S$ is adopted. It is defined as the ratio between the parallel runtime for a given number of CPUs and the serial runtime [42], i.e.

$$S=\frac{t_p}{t_s} \tag{13}$$

where $t_s$ is the runtime of the serial code using the best optimization and $t_p$ is the runtime of the parallel code for the same problem. Another important index is the efficiency, $E^{cpu}$, which is the ratio between the speed-up and the number of used CPUs, i.e.

$$E^{cpu}=\frac{S}{n} \tag{14}$$

It is helpful in determining the proper $n$ to be used. The speed-up $S$ can never exceed the number of used CPUs. Thus, $E^{cpu}$ should satisfy

$$0\leqslant E^{cpu}\leqslant 1 \tag{15}$$

In this paper, both multicore PC and cluster are equipped with a different core CPU (e.g. 1/2/4/8 core CPU were used in the Pleiades2 cluster). In order to maintain consistency, the term CPU is used to refer to the computing core rather than the physically separate CPU.

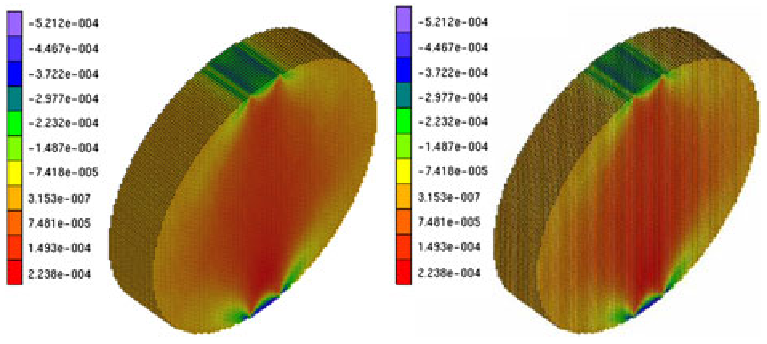
## 4.1. *Multicore DLSM*

A Brazilian disc model with 157,200 particles is calculated on two types of quad-core PCs. The elastic properties of the modeled material are $E=36GPa$ and $\nu=0.2$. The diameter of the disc is taken as 50 mm and the thickness is 10 mm. The parameters of the two multicore PCs used are listed in Table I. Figure 13 shows the simulation results obtained by the serial and multicore DLSM codes. It can be seen that the results obtained by the two codes are identical. This indicates that the parallel implementation is correct. It is found that the serial code cannot take full advantage of the multicore PC. Only 8% computing resource is used for the serial DLSM code, but this number increases to 88% for the multicore DLSM code. This means that OpenMP implementation is effective and that the computing resources can be fully utilized. The speed-up of the multicore DLSM has been tested on the first quad-core PC. The second one is used only to obtain the maximal speed-up of the multicore DLSM using the available PCs in LMR. It is attributed to the super thread technology used in the second PC. Yet, when the multicore DLSM code is running on this computer, it is hard to control, and it displays the type of the computing unit used (super thread or CPU core). Thus, results from the second PC are not suitable for speed-up analysis.

The Brazilian disc model is simulated on the first quad-core PC. The computing time of the multicore DLSM is compared with that of the serial DLSM, and the results are given in Figure 14. It can be seen that the serial code is a little faster than the multicore code when only one CPU is used.

Table I. Parameters of the quad-core PCs used.

| CPU name | Cores | Super thread | Speed | Memory |
|---|---|---|---|---|
| Intel Xeon | 4 | No | 2.40 GHz | 3 GB |
| Intel Core i7 950 | 4 | Yes | 3.07 GHz | 6 GB |



(a) Serial code (b) Parallel code (4 CPUs)

Figure 13. Simulation results obtained from the serial and parallel DLSM codes (contour map of $\varepsilon_{xx}$).

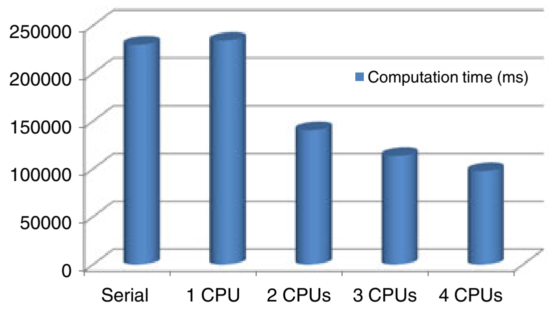


Figure 14. Computational time of the multicore DLSM with different CPUs.

This is because the parallel code inserts some instructions into the original code, which requires some additional computing time. However, when two cores are used, the speed of the multicore DLSM obviously becomes faster.

In order to study the influence of model size on speed-up, three models, Model A (2445 particles), Model B (19,760 particles) and Model C (157,200 particles), are computed by using both the serial DLSM code and the parallel DLSM code on the first PC. The speed-up of the parallel code is shown in Figure 15. Results show that the speed-up of the multicore DLSM varies with the size of the simulated model nonmonotonically. Overall, the trend is the same for different model sizes, and a speed-up value of around 2 could be achieved using the first PC.

In order to know the maximal speed-up of the multicore DLSM code, a Brazilian disc model with 78,500 particles is calculated on the second PC. It is a static simulation, and in order to obtain the equilibrium state, 20,000 cycles are calculated. The computing time is 86.16 minutes for the serial code, whereas it is reduced to 18.43 minutes for the parallel code. A speed-up of 4.68× is achieved. It is much higher than that obtained in the first PC. This is due to the fact that the CPU used in the second PC is more advanced than that in the first one, e.g. larger cache and the super thread technique. A 4.68× speed-up is desirable for practical application, e.g. a simulation previously taking 4 days could now be finished in 1 day. Now, it can be concluded that the implementation of the multicore DLSM is successful.

## 4.2. *Cluster DLSM*

In this section, the performance of the cluster DLSM code is tested. The test problem is shown in Figure 16. DLSM is used to simulate the fragmentation process of a rock specimen under one tunnel boring machine (TBM) cutter. The particle size is 1 mm, and the model dimension is 400 mm × 5 mm × 200 mm. The

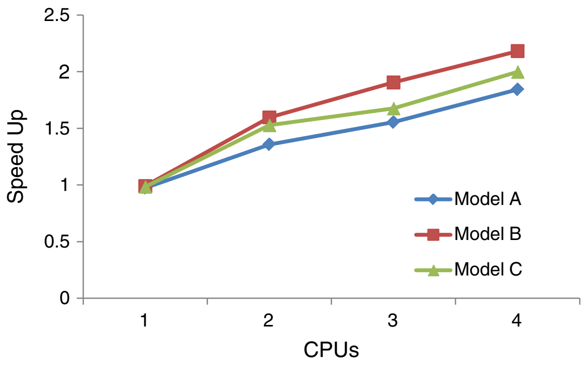


Figure 15. Speedup of the multicore DLSM code.

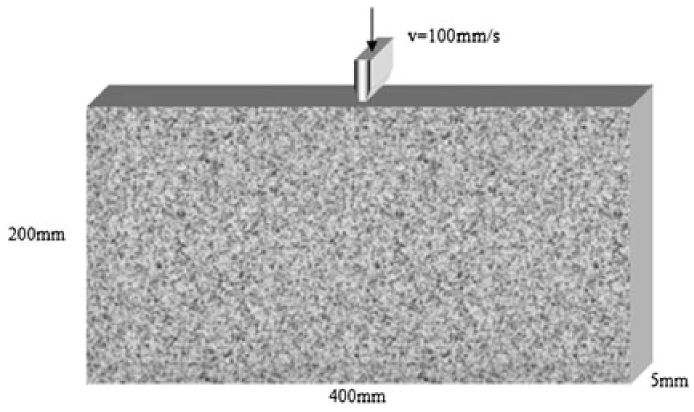


Figure 16. Scheme of single TBM cutter–induced fragmentation problem.

material properties of the rock are taken as follows: the elastic modulus is 40 GPa, the Poisson's ratio is 0.25 and the density is 2650 kg/m$^3$. The simple brittle constitutive law is adopted here. The ultimate bond deformation is given as 1.3e–4 mm. Except for the top loading surface, all other surfaces are fixed of the displacement in their normal directions. The applied loading rate is 100 m/s, which is applied on a small surface with a width of 12 mm for a duration of 10 ms. The time step is taken as $10^{-7}$ s; thus, around 100,000 cycles are needed for the simulation. The model is composed of 400,000 particles. The decomposition of DLSM for different cases is shown in Figure 17. Due to the limitation of available CPUs in the cluster, the maximum number of CPUs used to evaluate the speed-up of the cluster DLSM is 256. Figure 18 shows the simulation results of cluster DLSM. It turns out that cluster DLSM can work correctly with a large number of CPUs simultaneously involved in the computing.

When a parallel job is finished, a record file will be produced. Information of computing time can be found in this file, such as the total CPU time (the summed machine time of the allocated nodes) and the wall time (the actual time used in the cluster). The code itself also prints the computing time of the processor whose rank number equals zero, which is called the code time. These data for different cases are listed in Table II. It is found that, considering only the code time, a perfect linear speed-up is obtained. However, after careful investigation, it is found that it is not scientific to calculate the speed-up through the code time because it omits the I/O operation and the communication time. For this reason, the speed-up is calculated based on the wall time spent for each case. It can be seen that a maximal speed-up of 40.88× is achieved for the cluster DLSM code.

As mentioned before, the advantage of the cluster DLSM code is not only making the computing time shorter but also making it possile to solve problems that are beyond the capacity of a normal PC. It has been found that when the number of particles in DLSM exceeds one million (more than ten million bonds), it will become unsolvable for a normal PC because of the limitation of its memory space. As distributed memory is used in cluster DLSM, this problem can be easily solved by using an adequate number of processors in the cluster. In the following, the 3D case of the TBM cutter problem (as shown in Figure 19) is chosen as a demonstration example. For this problem, at a medium discretization level, even one quarter of the model needs four million particles. It exceeds the memory limit of a normal PC. Now, the problem is solved by the cluster DLSM code using 128 CPUs on Pleiades2. The simulation results are shown in Figure 20. Currently, the maximum number of particles for cluster DLSM is around four millions due to the limitation of the preprocessor and postprocessor. In the following sections, two examples will be presented to show the ability of the parallel DLSM code on geomechanics.

## 5. APPLICATION

### 5.1. *Microcompressive failure of rock material*

The DLSM is a microstructure-based model that is made up of physical springs and Newton's second law. The failure law used in the model is also simple; that is, it is based on the distance between two

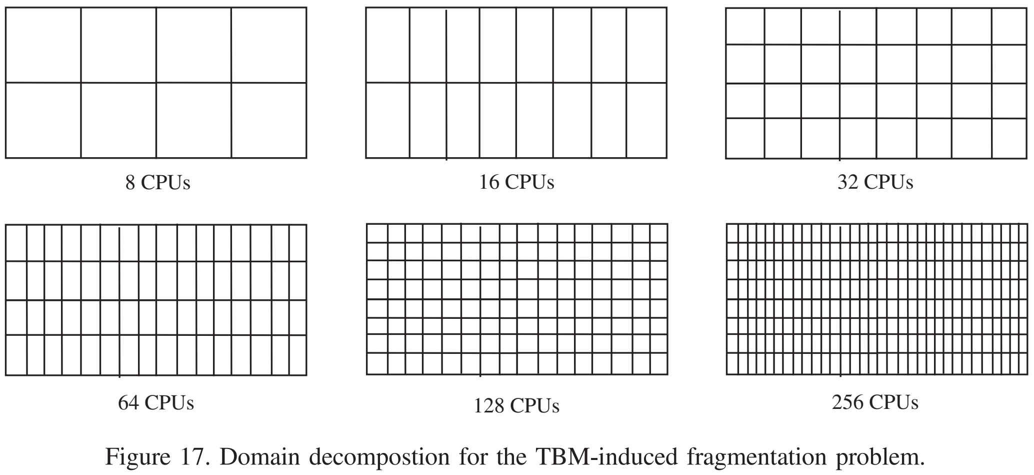
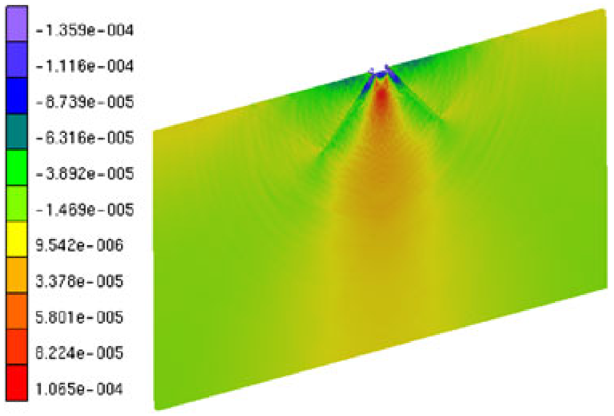
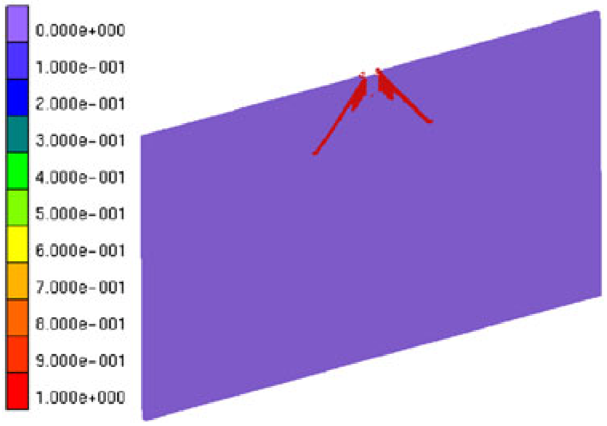


Figure 17. Domain decompostion for the TBM-induced fragmentation problem.



(a) Contour map of $\varepsilon_{xx}$



(b) Crack pattern (broken particle marked with red)

Figure 18. Simulation results of cluster DLSM using 256 CPUs.

Table II. Performance analysis results of the cluster DLSM code.

| CPUs | Total CPU time (s) | Code time (s) | Wall time (s) | $S$ | $E^{cpu}$ (%) |
|---|---|---|---|---|---|
| 1 | 2858 | 2859.61 | 2862 | 1 | 100 |
| 4 | 3557 | 893.25 | 901 | 3.1765 | 79 |
| 8 | 3508 | 435.01 | 488 | 5.8648 | 73 |
| 16 | 3308 | 199.04 | 426 | 6.7183 | 42 |
| 32 | 2990 | 89.96 | 196 | 14.602 | 46 |
| 64 | 2866 | 40.49 | 144 | 19.875 | 31 |
| 128 | 2705 | 18.87 | 88 | 32.523 | 25 |
| 256 | 2748 | 10.89 | 70 | 40.886 | 16 |

particles. For this reason, the model is suitable to use in studying some aspects of rock mechanics, e.g. the loading rate effect of rock material failure and strength. In this example, we will study the compressive failure of rock materials based on their microscopic structures. It is known that the ratio of compressive strength to tensile strength is much lower in particle-based numerical codes. In this section, a possible solution for this problem will be tested. High-resolution DLSM is built based on the microstructure information taken from a digital picture of rock material (see Figure 21). The physical size of the image is $1\,\text{mm} \times 1\,\text{mm}$. DLSM based on this image can be built as shown in Figure 21(b). Here, the particle size is $5\,\mu\text{m}$. The mechanical properties are taken as follows: the elastic modulus is

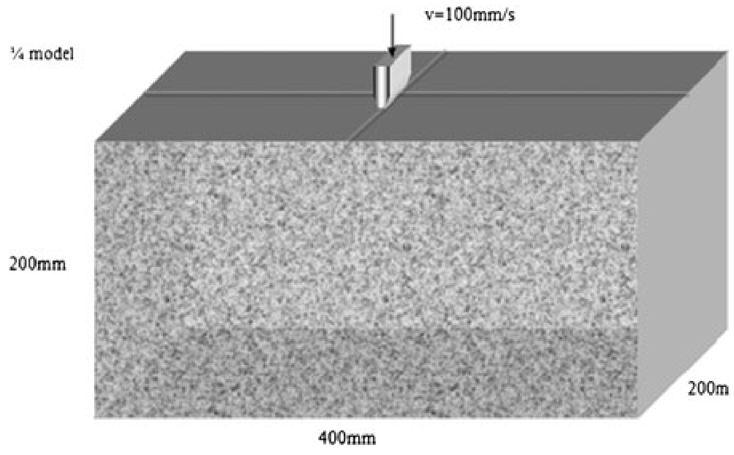


Figure 19. The 3D model of a single TBM cutter–induced fragmentation problem.

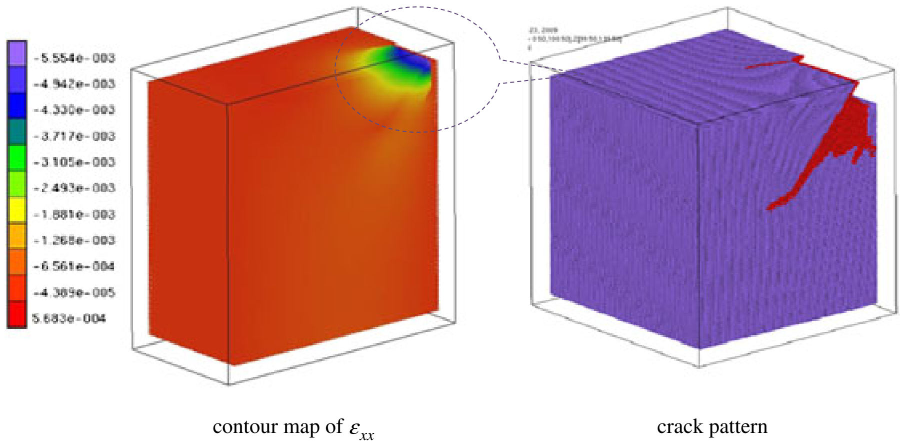


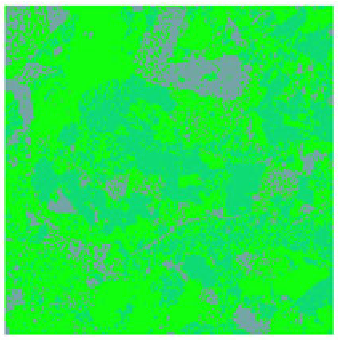
Figure 20. The 3D simulation results of the TBM cutter–induced fragmentation.

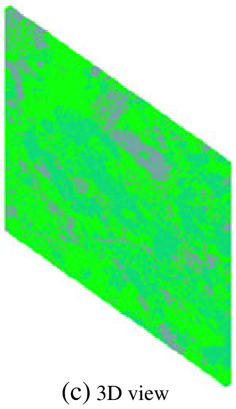
36 GPa, the Poisson's ratio is 0.25 and the density is 2450 kg/m$^3$. The simple brittle constitutive law is adopted. The ultimate bond deformation is 2.0e–4 μm for bonds formed by particles with different materials and 2.0e–3 μm for those linked with the same material. A velocity of $\pm 0.01$ mm/s is applied on the top and bottom surface to produce a piston-like uniaxial tensile/compressive loading. The corresponding strain rate is 5e–2/s, which produces a quasi-static simulation (the strain rate of conventional static tests on rock materials is around $10^{-5}$ to $10^{-1}$/s). The total particle number is 200,000. For the tensile simulation, it takes half a day in a normal PC. Yet, the compressive test will spend around 5 days, which is too long. Thus, cluster DLSM with 32 CPUs is used to tackle this problem. The obtained strain–stress curves of uniaxial tensile and compressive tests are shown in Figure 22. It shows that the ratio of compressive strength to tensile strength predicted by DLSM with detailed microstructure is 12.35. This value is actually the typical value for the ratio of compressive strength to tensile strength in rock materials. It reviews that the microstructure of rock material controls the compressive failure of rock material (without this true microstructure information, the predicted compressive strength is much lower). The different packing of particles can lead to different fracture patterns [1,38]. The regular particle packing may be unsuitable for geomaterials. Yet, using particle cluster methodology based on digital image may overcome this problem.

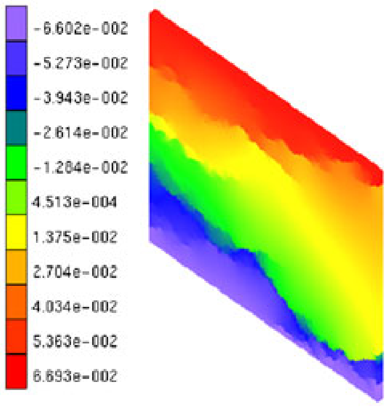
## 5.2. *Dynamic failure of tunnel under blasting loading*

A previous example shows the ability of the parallel DLSM code in saving computing time for the medium-sized numerical model. In this section, one inaccessible problem for the normal PC will be

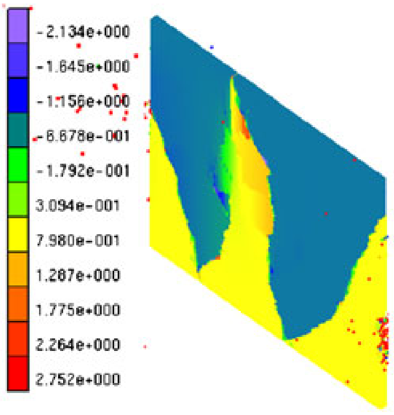
(a) Digital picture (taken from [43])


(b) Plane view


(c) 3D view

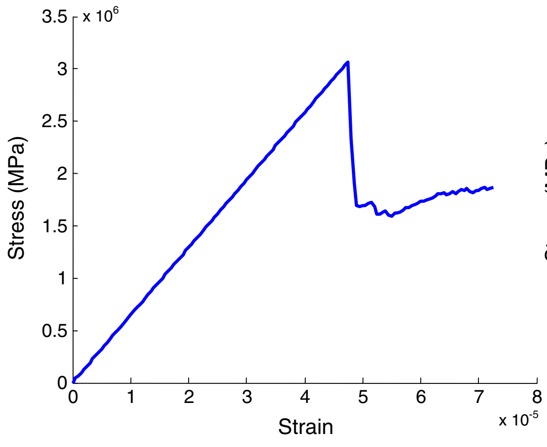


(d) Tensile failure ($u_y$)

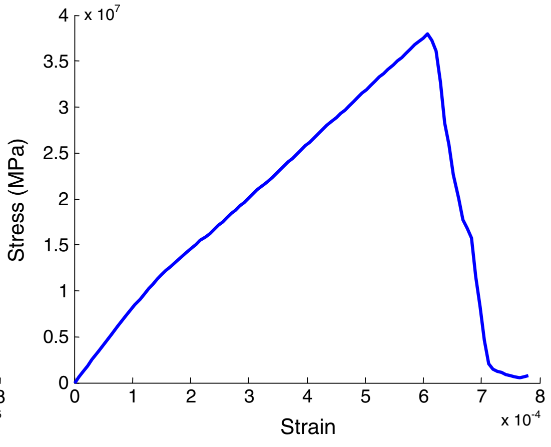


(e) Compressive failure ($u_y$)

Figure 21. The used microscopic model of rock material (the digital picture taken from [43]) and the corresponding DLSM modeling of tensileand compressive failure under uniaxial loading (contour map of displacement in y direction, unit $10^{-6}$m).



(a) Uniaxial tensile test



(b) Uniaxial compressive test

Figure 22. The strain–stress curves predicted by cluster DLSM for the uniaxial tensile and compressive tests.

handled by using cluster DLSM. The dynamic failure of a tunnel under blasting loading is an important issue for rock engineering, e.g. the safety of the existing tunnel must be well estimated when a new adjacent tunnel is undergoing blasting. In this section, one example of blasting wave

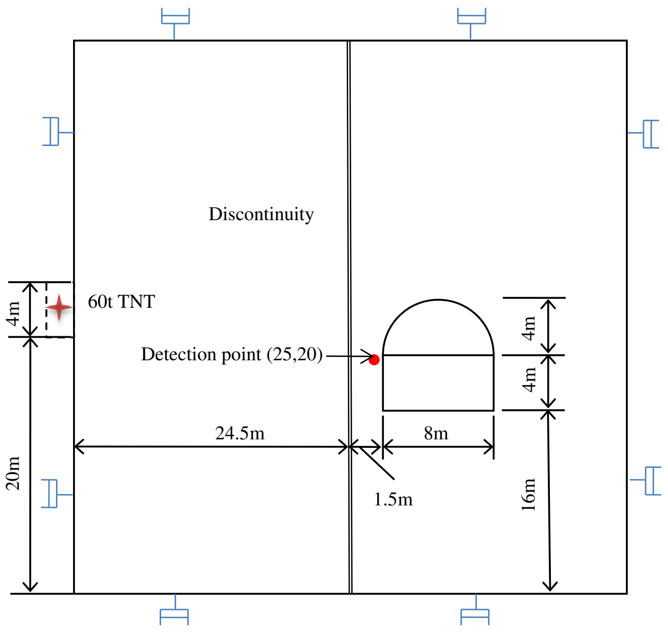


Figure 23. Computational model of the blasting wave propagation through rock tunnel problem.

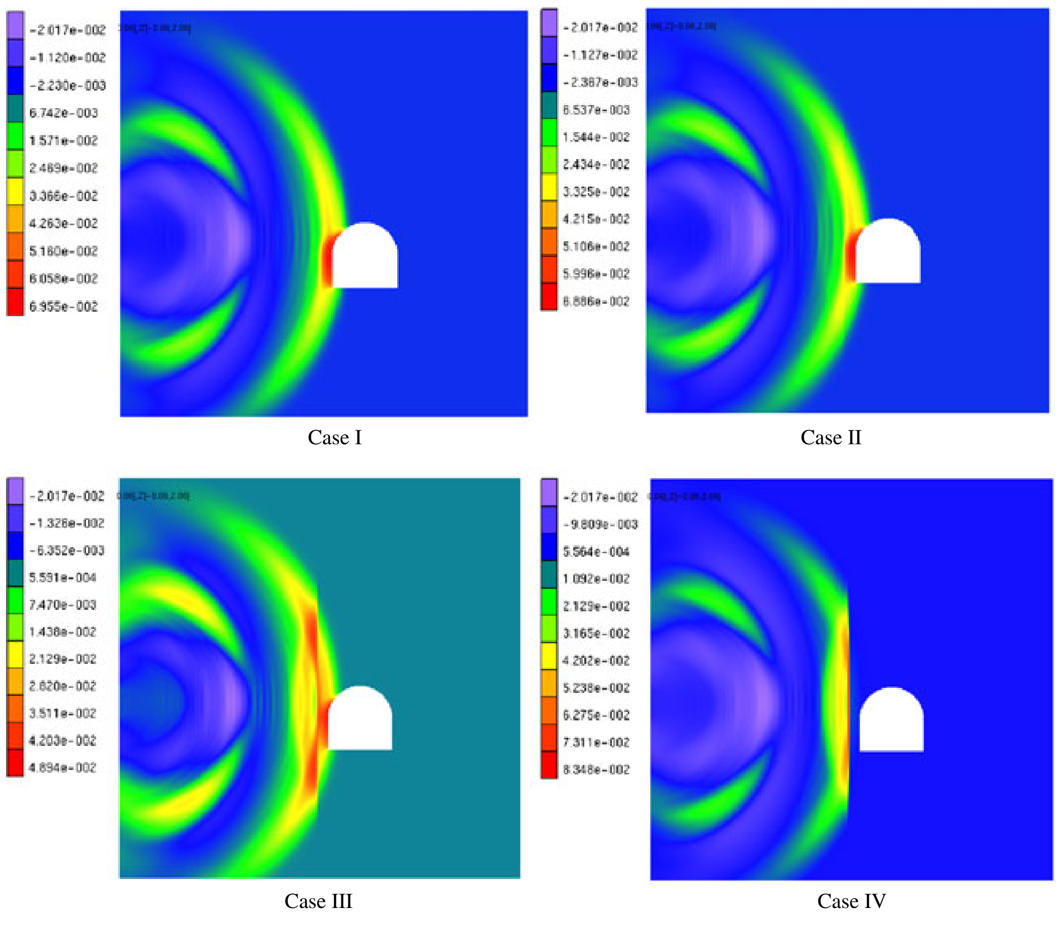


Figure 24. Contour map of the particle velocity (m/s) in x direction for different computational models at $t$=5 ms.

propagation through a tunnel will be modeled by cluster DLSM. Figure 23 shows the computational model and boundary conditions for the problem. The dimension of the model is 50 m × 50 m × 2 m, and the particle size is 0.125 m. The blasting load is applied to the left of the boundary, from 20 m to 24 m vertically, to simulate an explosion chamber of 4 m × 2 m. The blasting wave is simplified as a triangular overpressure history with two phases. The maximum overpressure $P_{\max}$ is equal to 30.23 MPa, and the duration of rise phase $t_1$ and the total duration $t_2$ are 0.5 and 2.5 ms, respectively. The material properties of the rock are taken as follows: the elastic modulus is 74 GPa, the Poisson's ratio is 0.2 and the density is 2650 kg/m$^3$. The ultimate bond deformation is taken as 2.5e–5 m, which is calculated based on the tensile strength of Bukit Timah granite. Discontinuity is represented by setting a material layer with a weaker elastic modulus, where the weakness ratios are taken as 1.0 (Case I), 0.5 (Case II), 0.1(Case III) and 0.01 (Case IV). For DLSM, more than two million particles are required to build this computational model, which is surely an inaccessible problem for the normal PC. Yet, with cluster DLSM, this problem can be tackled easily.

The blasting wave propagation through rock mass and the influence of discontinuity on the failure pattern of a tunnel under a blasting wave are simulated by cluster DLSM. Figure 24 shows the contour map of the velocity in a horizontal direction for the computational model under different cases. It can

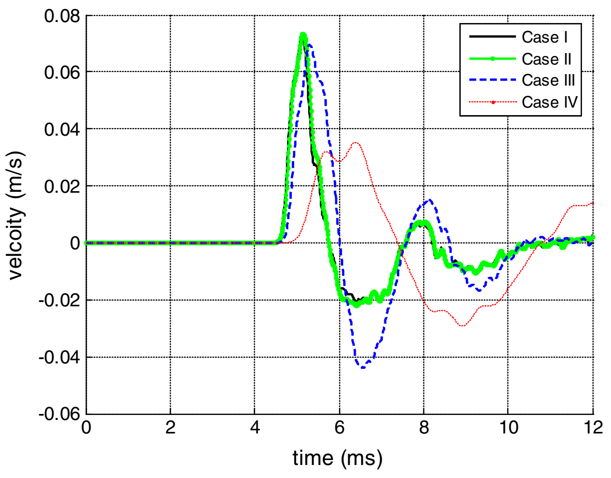


Figure 25. The velocity histories at the detection point predicted by DLSM computational models for the maximum peak pressure of 30.23 MPa.

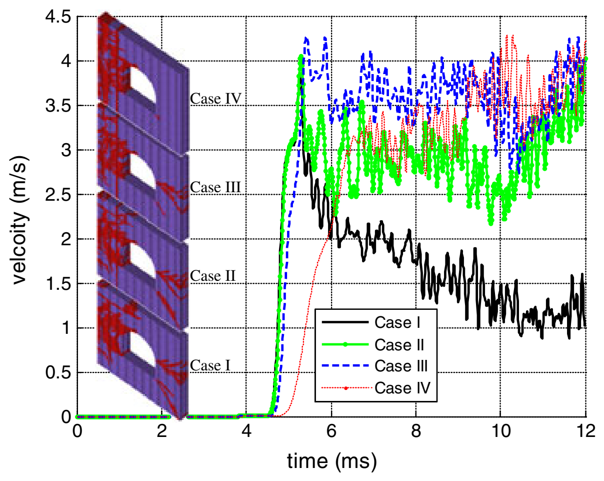


Figure 26. The velocity histories at the detection point predicted by DLSM computational models for the maximum peak pressure of 151.15 MPa.

be seen that the discontinuities can reflect the blasting wave. Moreover, the recorded velocity histories during the numerical tests (see Figure 25) reveal that the peak particle velocity (PPV) increases with the stiffness of the discontinuity. It is found that there is no apparent damage that occurred around the tunnel under the loading pressure of 30.23 MPa. In order to observe the failure pattern of different models, another set of simulations with a maximum overpressure $P_{\max}$ of 151.15 MPa is performed. The recorded velocity histories and the failure pattern of the rock around the tunnel are shown in Figure 26. It shows that PPV decreases with the stiffness of the discontinuity, whereas the damage degree of the tunnel increases. Altogether, from the simulation results of this section, it can be concluded that the damage of a tunnel under dynamic loading can be released through presetting some weak discontinuity/cavern.

## 6. CONCLUSIONS

In this paper, the parallelization of DLSM is presented. The available parallel environments are briefly introduced. Then, the parallelization of DLSM on multicore PC and cluster are presented. The OpenMP is used to parallelize the DLSM code and make it work effectively on a multicore PC. The OpenMP implementation needs only a few modifications of the original code. Examples are given to show the performance of the parallel DLSM code on a multicore PC. It is found that the implementation is effective and successful. Another version of the code, the cluster DLSM, has been developed for massive parallel computing using clusters. The parallel DLSM solver on cluster is implemented by using MPI. The whole software package is finished through the cooperation between PC and cluster. The performance of the cluster DLSM is tested, and a speed-up of 40.88 is achieved in the case using 256 CPUs in a Pleiades2 cluster. Finally, a few problems, which were previously impossible to handle using a normal PC, are successfully solved by using the developed cluster DLSM code.

## REFERENCES

1. Zhao GF, Fang J, Zhao J. A 3D distinct lattice spring model for elasticity and dynamic failure. *Int J Numer Anal Meth Geomech* 2011; **35**(8):859–885.
2. http://techfreep.com/intel-80-cores-by-2011.htm, 2010.
3. Owens JD, Houston M, Luebke D, Green S, Stone JE, Phillips JC. GPU computing. *Proceedings of the IEEE* 2008; **96**(5):879–899.
4. http://www.nvidia.com/cuda_home.html, 2010.
5. Baker M, editor. *Cluster Computing White Paper*. http://www.dcs.port.ac.uk/mab/tfcc/WhitePaper, 2000.
6. Flynn M. Very high-speed computing systems, Proc. *IEEE* 1966; **54**:1901–1909.
7. Chang V. Experiments and investigations for the personal high performance computing (PHPC) built on top of the 64-bit processing and clustering systems. In *13th Annual IEEE International Symposium*, IEEE Press: Germany, 2006; 27–30.
8. http://www.openmp.org, 2010.
9. Dick B, Jacqueline F, Bradford N. *PThreads Programming*. O'Reilly Media: Sebastopol, California, United States 1996.
10. http://www.threadingbuildingblocks.org/, 2010.
11. Tarmyshov KB. Muller-Plathe F. Parallelizing a molecular dynamics algorithm on a multiprocessor workstation using OpenMP. *Journal of Chemical Information and Modeling* 2005; **45**(6):1943–1952.
12. Zsaki AM. Parallel generation of initial element assemblies for two-dimensional discrete element simulations. *Int J Numer Anal Meth Geomech.* 2009; **33**(3):377–389.
13. Gao D, Schwartzentruber TE. Optimizations and OpenMP implementation for the direct simulation Monte Carlo method. *Comput Fluid.* 2011; **42**(1):73–81.
14. Williams JR, Holmes DW, Tilke P. Parallel Computation Particle Methods for Multi-phase Fluid Flow with Application Oil Reservoir Characterization. *Particle-Based Methods: Fundamentals and Applications*, Onate E, Owen D.R.J. (eds.). Springer, 2011; 113–134.
15. Rick M. CPU designers debate multi-core future. *EE Times*. http://www.eetimes.com/, 2008.
16. http://en.wikipedia.org/wiki/Multi-core_processor#cite_note-1, 2010.
17. Wasson S. NVIDIA's GeForce 8800 graphics processor, Tech Report, 2007.
18. http://www.opengl.org/, 2010.
19. http://www.khronos.org/opencl/, 2010.
20. Elsen E, LeGresley P, Darve E. Large calculation of the flow over a hypersonic vehicle using a GPU. *J Comput Phys*, 2008; **227**(24):10148–10161.

21. Anderson JA, Lorenz CD, Travesset A. General purpose molecular dynamics simulations fully implemented on graphics processing units. *J Comput Phys*, 2008; **227**(10):5342–5359.
22. Takahashi T, Hamada T. GPU-accelerated boundary element method for Helmholtz' equation in three dimensions. *Int J Num Meth Eng.* 2009; **80**(10):1295–1321.
23. Joldes GR, Wittek A, Miller K. Real-time nonlinear finite element computations on GPU - Application to neurosurgical simulation. *Comput Meth Appl Mech Eng.* 2010; **199**(49–52):3305–3314.
24. http://www.top500.org/, 2010.
25. http://pleiades.epfl.ch/, 2010.
26. http://www.mpi-forum.org, 2010.
27. http://www.csm.ornl.gov/pvm/, 2010.
28. http://www-unix.mcs.anl.gov/mpi/mpich., 2010.
29. Maknickas A, Kačeniauskas A, Kačianauskas R, Balevičius R, Džiugys A. Parallel DEM software for simulation of granular media. *Informatica* 2006; **17**(2):207–224.
30. Singh IV, Jain PK. Parallel meshless EFG solution for fluid flow problems. *Num Heat Trans, Part B.* 2005; **48**(1):45–66.
31. Komatitsch D, Erlebacher G, Göddeke D, Michéa D. High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster. *Journal of Computational Physics* 2010; **229**(20):7692–7714.
32. Wang W, Kosakowski G, Kolditz O. A parallel finite element scheme for thermo-hydro-mechanical (THM) coupled problems in porous media. *Computers and Geosciences* 2009; **35**(8):1631–1641.
33. Tang G, D'Azevedo EF, Zhang F, Parker JC, Watson DB, Jardine PM. Application of a hybrid MPI/OpenMP approach for parallel groundwater model calibration using multi-core computers. *Comput Geosci.* 2010; **36**(11): 1451–1460.
34. Yang CT, Huang CL, Lin CF. Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU cluster. *Computer Physics Communications* 2011; **182**(1):266–269.
35. Hrennikoff A. Solution of problems of elasticity by the framework method. *ASME J. Appl. Mech.* 1941; **8**: A619–A715.
36. Hahn M, Wallmersperger T, Kroplin BH. Discrete element representation of continua: proof of concept and determination of the material parameters. *Computational Materials Science* 2010; **50**:391–402.
37. Darve F, Nicot F. On incremental non-linearity in granular media: phenomenological and multi-scale views (Part I). *Int. J. Numer. Anal. Meth. Geomech.* 2005; **29**:1387–1409.
38. Zhao GF. *Development of micro-macro continuum-discontinuum coupled numerical method*. Phd thesis. EPFL: Switzerland, 2010.
39. Yoon JS. Application of experimental design and optimization to PFC model calibration in uniaxial compression simulation. *Int. J. Rock Mech. & Min.Sci.* 2007; **44**:871–889.
40. Kadau K, Germann TC, Lomdahl PS. Large-scale molecular dynamics simulations of 19 Billion Particles. *Int J Modern Phys C* 2004; **15**(1):193–201.
41. Michael G, Stephan K. *Gerhard Z.* Numerical simulation in molecular dynamics: Springer, 2007.
42. Kumar V, Grama A, Gupta A, Karypis G. *Introduction to parallel computing: design and analysis of Algorithms*. Benjamin/Cummings, 1994.
43. Walker I. An introduction to mineralogical terms and observations with the Leica CME. http://www.microscopy-uk.org.uk/mag/artfeb04/iwouslides.html, 2010.