CPU-accelerated explicit three-dimensional discontinuous deformation analysis with cloud computing

Xi Wang¹, Wei Wu¹,²,³*, Liu Jian⁴, Li Huaming⁵ and Hehua Zhu¹,²,³

¹ College of Civil Engineering, Tongji University, Shanghai China
² State Key Laboratory of Disaster Reduction in Civil Engineering, Tongji University, Shanghai China
³ Key laboratory of Geotechnical and Underground Engineering of Ministry of Education, Tongji University, Shanghai, China
⁴ PowChina Hebei Transportation Highway Investment Development Co., Ltd., Shijiazhuang, 050081, People’s Republic of China
⁵ Si Chuan Lehan Expressway Co., Ltd., Leshan 614000, People’s Republic of China

*Corresponding author: weiwu@tongji.edu.cn

Abstract. Prompt feedback based on a numerical simulation, such as discontinuous deformation analysis, plays an important role in digitalized geotechnical engineering. Cloud computing also prevails owing its flexible configuration and computational power. This study proposes an explicit three-dimensional discontinuous deformation analysis method using explicit contact forces that can circumvent the difficulty of not converging the open–close iteration and the inefficiency of classic implicit discontinuous deformation analysis. The serial code is ready to be parallelized by relocating the postjudgment of the contact status and using LDL decomposition to solve fixed-size linear equations. We propose herein the block-wise parallelization concept, which significantly simplifies the work of parallel computation. No modification under the block level is needed. In addition, one can realize parallel computing by adding just three lines of the new code with the help of OpenMP. A simple virtual cloud machine is used to run the program to validate the improved efficiency of the newly developed algorithms. A 3.5 speedup ratio is achieved with four CPU cores and eight processors.

1. Introduction

Discrete numerical methods such as the discontinuous deformation analysis (DDA) and DEM, play an important role in analyzing the discontinuous behaviors of rock masses, especially landslides, earthquakes, tunneling, and pit excavations. DDA [1] is a widely used discrete numerical method that keeps developing in both two (2D)- [2], [3] and three (3D)-dimensional [4], [5] cases. DDA features rigorous contact analysis but also suffers from relevant inefficiency. Recently, the advent of the...
contact theory laid a solid theoretical foundation for the contact analysis in DDA. The arising multicover method [4], last entrance plane method [5], and local convex decomposition [6] have also provided an efficient method of materializing the contact theory [7]. 2D DDA is widely applied in landslides [8], [9] and tunneling [10], [11]. Meanwhile, 3D DDA is still limited to some small-scale scenarios [12] due to the bottleneck in efficiency.

3D DDA consumes a formidable amount of time when dealing with large-scale engineering cases because of its implicit contact force calculation and open–close iteration (OCI). An implicit contact force will link the unknowns of two contacting blocks, making the global stiffness matrix nondiagonal. To determine the contact status, an extra loop is needed until the open or closed status remains unchanged. Much time will be needed because solving the global equation is a tough task and might be repeated many times. To circumvent this problem, an explicit contact force will be adopted in this work.

Hardware is also used to accelerate DDA calculations. 2D DDA has been modified to use hardware acceleration. Fu et al. (2016) [13] and Song et al. (2017) [14] adopted efficient equation-solving algorithms and parallelized the equation solver with OpenMP and CUDA. Peng et al. (2020b) [15] and Yu et al. (2020) [16] achieved a high degree of parallelization by adding OpenMP instructions to the program under the block level, such as distance and angle calculations in contact detection, as well as matrix assembly and equation solvers. Xiao et al. (2017) [17] parallelized 2D DDA with CUDA. However, little work has been done on accelerating 3D DDA. In particular, Peng et al. (2019) [18] used OpenMP [19] to parallelize the equation solver and the entire stage of 3D DDA. Only a few studies have been reported as regards the explicit 3D DDA parallelization. The explicit 3D DDA is more appropriate for parallelization because the global stiffness matrix is now block-diagonal.

Cloud computing is a prevailing technology due to its flexible configuration and computing power. A very few studies have used cloud virtual machines to deploy discrete numerical programs. In this study, we adapt the explicit 3D DDA to be parallelized and create a simple virtual machine to run the parallelized program. The block-wise parallelization is realized using OpenMP directives, which leads to a very practicable parallelization scheme.

2. 3D DDA with explicit contact forces

```plaintext
Serial DDA:
For i, j:
    // Part I: Contact detection (Block i, Block j)
    Detect (Block i, Block j);
    Save_contacts (detected contacts);
For i:
    // Part II: Update (Block i)
    Assemble_matrix (Block i);
    Solve_displacements (Block i);
    Update_block (Block i);
```

![Figure 1. Serial DDA program.](image-url)

2.1. Displacement matrix
DDA takes displacements as unknowns. A block has 12 unknowns since the first-order displacement hypothesis is adopted. The displacement \((u,v,w)\) of an arbitrary point \((x,y,z)\) of a block is presented as

\[
(u,v,w) = T(x,y,z) \cdot D
\]  
\[
D = \begin{bmatrix} u_c & v_c & w_c & r_x & r_y & r_z & \varepsilon_x & \varepsilon_y & \varepsilon_z & Y_{yz} & Y_{zx} & Y_{xy} \end{bmatrix}^T
\]  
\[
T(x,y,z) = \begin{bmatrix} 1 & 0 & 0 & 0 & Z & -Y & X & 0 & 0 & 0 & Z/2 & Y/2 \\ 0 & 1 & 0 & -Z & 0 & X & 0 & Y & 0 & Z/2 & 0 & X/2 \\ 0 & 0 & 1 & Y & -X & 0 & 0 & 0 & Z & Y/2 & X/2 & 0 \end{bmatrix}
\]

\[
X = x - x_c, Y = y - y_c, Z = z - z_c
\]

where, \(D\) with shape \((12 \times 1)\) is the displacement matrix; \((u_c,v_c,w_c)\) are the centroid translations along the \(x, y,\) and \(z\) axes, respectively; \((r_x,r_y,r_z)\) are the rotations about the \(x, y,\) and \(z\) axes, respectively, of the rigid body; \((\varepsilon_x,\varepsilon_y,\varepsilon_z)\) and \((Y_{yz},Y_{zx},Y_{xy})\) are the normal and shear strains, respectively; and \((x_c,y_c,z_c)\) is the block centroid.

2.2. Time integration

\[
MD_{n+1} + CD_{n+1} + KD_{n+1} = F_{n+1}
\]

\[
\begin{align*}
\hat{K}D_{n+1} &= \hat{F} \\
\hat{R} &= \frac{2}{(\Delta t)^2}M + \frac{2}{\Delta t}C + K \\
\hat{F} &= F_{n+1} + (\Delta t^2 M + C) \hat{D}_n
\end{align*}
\]

The motion equation of DDA is depicted as Eq. (5). Constant acceleration is assumed in DDA; thus, the relation among the displacement, velocity, and acceleration can be determined using Eq. (6).

2.3. Contact detection

The classical direct search method was adopted herein. Neighbor search was first performed, followed by refined search that determines the contact point and direction. In contact detection, we only need to read the geometry of two blocks and write to a global contact array upon finding a potential contact. The postjudgment of the contact status after solving the global equations is usually performed in the end of a time step, which also needs the information of two contacting blocks. We relocated the postjudgment of the contact status right before saving the detected contacts to unify the data access pattern. Thus, the serial DDA was divided into two stages according to data access (Figure 1). The motion equation of DDA is presented as Eq. (5). Constant acceleration was assumed in DDA; thus, the relation among the displacement, velocity, and acceleration can be determined using Eq. (6).

2.4. Solving block equations

The contact constraints are treated as forces; hence, each block can now be solved independently. After assembling the matrix for each block [20], the LDL decomposition was adopted to obtain the displacements [Eq. (7)].

\[
\begin{align*}
A &= LDL' \\
Ly &= b \\
DL'x &= y
\end{align*}
\]
2.5. Explicit contact forces

The contact forces are now treated as forces. The relevant submatrices are calculated as follows:

\[ F_n = \begin{cases} 
- (p_n d_n + c_m n_{min} v_n) T_i(x_i, y_i, z_i) \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} & \rightarrow F_i \\
(p_n d_n + c_m n_{min} v_n) T_j(x_j, y_j, z_j) \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} & \rightarrow F_j 
\end{cases} \]  

(8)

\[ F_{lock} = \begin{cases} 
- p_{t} d_{t} T_i(x_i, y_i, z_i) \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} & \rightarrow F_i \\
p_{t} d_{t} T_j(x_j, y_j, z_j) \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} & \rightarrow F_j 
\end{cases} \]  

(9)

\[ F_{slide} = \begin{cases} 
- \mu p_n d_n T_i(x_i, y_i, z_i) \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} & \rightarrow F_i \\
\mu p_t d_t T_j(x_j, y_j, z_j) \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} & \rightarrow F_j 
\end{cases} \]  

(10)

3. Block-wise parallelization

We can readily obtain the parallelized DDA program after inserting several OpenMP directives (Figure 2). This scheme is more practicable than ever because we do not need to dive into the detailed algorithms under the block level.

4. Virtual machine on the cloud

We used the Google Cloud Platform for this study. A free trial is available for everyone with $300.
credits, which will guarantee that other researchers will be able to replicate our studies without spending extra money. However, the hardware quota during the free trial is limited; hence, we created two simple virtual machines to be used in the next section.

To facilitate the notation, we called the first virtual machine as the serial machine and the second one as the parallel machine. Tables 1a and b list the CPU and the memory parameters. The allocated CPUs for these two machines were slightly different. For a better comparison, we ran a serial program on both machines and used two threads. We then explored the speedup ratio of the parallel machine using different numbers of threads.

Table 1a. Serial and parallel virtual machines.

| Parameters          | Serial machine | Parallel machine |
|---------------------|----------------|-----------------|
| CPU name            | Intel(R) Xeon(R) CPU | Intel(R) Xeon(R) CPU |
| CPU GHz             | 2.3 GHz         | 3.1 GHz         |
| # CPU cores         | 1               | 4               |
| CPU processes       | 2               | 8               |
| Cache size          | 46080 KB        | 25344 KB        |

Table 1b. Serial and parallel virtual machines.

| Running time (hours) | 1 thread | 2 threads | 4 threads | 8 threads |
|----------------------|----------|-----------|-----------|-----------|
| Serial machine       | 25.6     | 17.2      | 17.0      | 17.7      |
| Parallel machine     | 10.7     | 5.5       | 3.8       | 3.1       |

5. Numerical cases

5.1. Validation of the explicit DDA with an inclined slope sliding case

The block slide case is one of the most classical cases for validating discontinuous numerical algorithms. We set the following parameters: mass density $\rho = 2.0 \times 10^3$ kg/m$^3$; Young’s modulus $E = 30$ GPa; Poisson’s ratio $\nu = 0.3$; inclination angle $\alpha = 30^\circ$; friction angle $\varphi = 15^\circ$; number of time steps $5 \times 10^5$; time step $\Delta t = 1 \times 10^{-5}$ s; damping coefficient $\zeta_n = 100$; normal spring stiffness $k_n = 20E$; and shear spring stiffness $k_s = 0.5k_n$.

Figure 4 plots the displacements of the numerical and analytical methods. The results matched well;
thus, we can conclude that the explicit DDA is accurate in replicating the sliding block case.

![Figure 3. Block slide case.](image)

![Figure 4. Comparison between the numerical and analytical displacements.](image)

5.2. Efficiency test on the cloud virtual machines

A multiblock fall test was performed, wherein a block wall comprising of 2526 blocks falls freely under gravity. We adopted the following parameters for this test: mass density $\rho = 2.0 \times 10^3$ kg/m$^3$; Young’s modulus $E = 30$ GPa; Poisson’s ratio $\nu = 0.3$; friction angle $\varphi = 0^\circ$; time step $\Delta t = 2 \times$
10^{-6}$ s; number of time steps $6 \times 10^5$; damping coefficient $c_n = 100$; normal spring stiffness $k_n = 20E$; and shear spring stiffness $k_s = 0.5k_n$.

The falling process was similar to that presented in Figure 5. We used different threads for the serial virtual and parallel machines. The serial machine only had one CPU core with two processes; thus, the time will not decrease in the presence of more threads. On the contrary, the parallel machine had four CPU cores and eight processes; thus, a larger speedup ratio can be achieved.

6. Conclusions

This study used a block-wise parallelization scheme to parallelize the explicit 3D DDA, which is a very practicable scheme. The OCI was circumvented, and the global equation was decomposed into many small block-level linear equations. With the OpenMP directives, only three lines of the new code were needed to realize the parallelization.

Two cloud virtual machines were created on the Google Cloud Platform to validate the accelerated performance of the new algorithm. The max speedup ratios of 1.5 and 3.5 were achieved by the serial and parallel machines, respectively. The cloud computing technology endows us with flexible computational power. We can use the elementary serial computer to complete small-scale tasks. The high-performance parallel machine is also available for computation-intensive tasks.

7. References

[1] G. Shi, “Discontinuous deformation analysis: a new numerical model for the statics and dynamics of deformable block structures,” Eng. Comput., 1992.
[2] H. Bao and Z. Zhao, “An alternative scheme for the corner-corner contact in the two-dimensional Discontinuous Deformation Analysis,” Adv. Eng. Softw., vol. 41, no. 2, pp. 206–212, 2010.
[3] X. Wang, W. Wu, H. Zhu, J.-S. Lin, and H. Zhang, “Contact detection between polygonal blocks based on a novel multi-cover system for discontinuous deformation analysis,” Comput. Geotech., vol. 111, pp. 56–65, 2019.
[4] X. Wang, W. Wu, H. Zhu, J.-S. Lin, and H. Zhang, “Acceleration of contact detection between arbitrarily shaped polyhedra based on multi-cover methods in three dimensional discontinuous deformation analysis,” Int. J. Rock Mech. Min. Sci., vol. 132, p. 104387, 2020.
[5] X. Wang, W. Wu, H. Zhu, H. Zhang, and J.-S. Lin, “The last entrance plane method for contact indeterminacy between convex polyhedral blocks,” Comput. Geotech., vol. 117, p. 103283, 2020.
[6] H. Zhang et al., Angle-Based Contact Detection in Discontinuous Deformation Analysis. 2020.
[7] G. H. Shi, “Contact theory,” Sci. China Technol. Sci., vol. 58, no. 9, pp. 1450–1496, 2015.
[8] G. Ma, H. Matsuyama, S. Nishiyama, and Y. Ohnishi, “Practical studies on rockfall simulation by DDA,” J. Rock Mech. Geotech. Eng., vol. 3, no. 1, pp. 57–63, 2011.
[9] W. Wu, X. Wang, H. Zhu, K.-J. Shou, J.-S. Lin, and H. Zhang, “Improvements in DDA program for rockslides with local in-circle contact method and modified open-close iteration,” Eng. Geol., vol. 265, p. 105433, 2020.
[10] X. Fu, Q. Sheng, Y. Zhang, Y. Zhou, and F. Dai, “Boundary setting method for the seismic dynamic response analysis of engineering rock mass structures using the discontinuous deformation analysis method,” Int. J. Numer. Anal. Methods Geomech., vol. 39, no. 15, pp.
1693–1712, 2015.

[11] H. K. Law and I. P. Lam, “Evaluation of seismic performance for tunnel retrofit project,” J. Geotech. geoenvironmental Eng., vol. 129, no. 7, pp. 575–589, 2003.

[12] Z. Li et al., “Simulating the damage extent of unreinforced brick masonry buildings under boulder impact using three-dimensional discontinuous deformation analysis (3-D DDA),” Eng. Fail. Anal., vol. 93, pp. 122–143, 2018.

[13] X. Fu, Q. Sheng, Y. Zhang, and J. Chen, “Investigation of highly efficient algorithms for solving linear equations in the discontinuous deformation analysis method,” Int. J. Numer. Anal. Methods Geomech., vol. 40, no. 4, pp. 469–486, 2016.

[14] Y. Song, D. Huang, and B. Zeng, “GPU-based parallel computation for discontinuous deformation analysis (DDA) method and its application to modelling earthquake-induced landslide,” Comput. Geotech., vol. 86, pp. 80–94, 2017.

[15] X. Peng, P. Yu, G. Chen, M. Xia, and Y. Zhang, “CPU-accelerated explicit discontinuous deformation analysis and its application to landslide analysis,” Appl. Math. Model., vol. 77, pp. 216–234, 2020.

[16] P. Yu, X. Peng, G. Chen, L. Guo, and Y. Zhang, “OpenMP-Based Parallel Two-Dimensional Discontinuous Deformation Analysis for Large-Scale Simulation,” Int. J. Geomech., vol. 20, no. 7, p. 4020083, 2020.

[17] Y. Xiao, M. Huang, Q. Miao, J. Xiao, and Y. Wang, “Architecting the discontinuous deformation analysis method pipeline on the GPU,” Proc. - 2017 IEEE 31st Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2017, pp. 1188–1197, 2017.

[18] X. Peng et al., “Parallel computing of three-dimensional discontinuous deformation analysis based on OpenMP,” Comput. Geotech., vol. 106, pp. 304–313, 2019.

[19] X. Peng, G. Chen, P. Yu, Y. Zhang, H. Zhang, and L. Guo, “A full-stage parallel architecture of three-dimensional discontinuous deformation analysis using OpenMP,” Comput. Geotech., vol. 118, p. 103346, 2020.

[20] Y. H. Hatzor, G. Ma, and G. Shi, Discontinuous deformation analysis in rock mechanics practice. CRC Press, 2017.