Development of an efficient parallelization scheme for fully implicit discontinuous deformation analysis (dda)

Tatsuki Tokuda
Hiroshima University, Hiroshima, Japan

Ryota Hashimoto
Kyoto University, Kyoto, Japan

ABSTRACT: In this study, we propose a parallelization scheme for the fully implicit Discontinuous Deformation Analysis (FI-DDA) to establish an accurate and efficient numerical method for the dynamic stability analysis of rock slopes. Specifically, a parallel iterative linear equation solver was newly introduced to the FI-DDA, and then, efficient contact detection and contact stiffness matrix assembly algorithms suitable for parallel processing were proposed and introduced to the FI-DDA. The analysis code was parallelized with OpenMP and the performance of the developed method was verified by simulating a centrifugal model shaking table experiment of discontinuous rock slope.

Keywords: DDA, OpenMP, rock slope, parallelization.

1 BACKGROUND AND PURPOSE OF THIS STUDY

The seismic safety of rock slopes around nuclear power plants in Japan has been evaluated by limit equilibrium method based on the stress state obtained by equivalent linear analysis using the finite element method (The Japan Electric Association, 2016). Recently, however, there has been a need to evaluate the dynamic behavior of collapsed rock masses and the residual displacement as the input earthquake and ground motion levels used in the design increase.

One method for assessing the dynamic behaviors of discontinuous rock masses is Discontinuous Deformation Analysis (DDA), an implicit type discontinuum-based numerical method for elastic block systems (Shi & Goodman, 1989). The DDA has been widely used in the rock engineering field. However, it has been reported that when performing seismic response analysis with the DDA, it is difficult to find proper parameters to obtain an accurate solution (Koyama et al., 2009).

Hashimoto et al. (2021) pointed out that the updating algorithm of the friction between the blocks in the original DDA overestimates the friction force and induces computational instability, and developed a fully implicit DDA (hereafter, FI-DDA) incorporating an implicit friction updating method (return mapping algorithm), that enables robust and accurate analysis of the sliding behaviors along the joints. On the other hand, the FI-DDA has a disadvantage in computational efficiency. Since the return mapping procedure causes residuals in the equilibrium of the entire system due to the correction of friction forces, the Newton-Raphson iteration was also introduced (Fig. 1). While

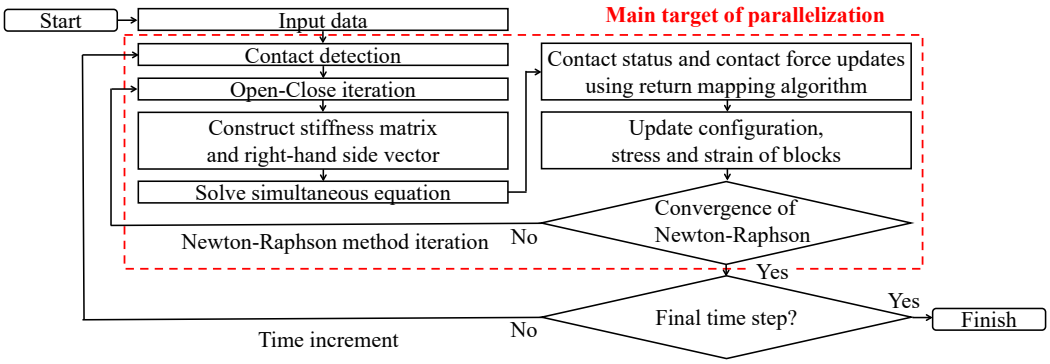

Figure 1. Flowchart of fully implicit DDA.

this allows for more accurate analysis with larger time increments than the original DDA, it increases the computational cost per time step. The increase in overall computation time is significant especially when a large rock slope with many blocks is analyzed.

Therefore, in the present study, we propose a parallelization scheme for the FI-DDA to establish an accurate and efficient method for the dynamic stability evaluation of rock slopes. A parallel iterative linear equation solver was newly introduced to the FI-DDA, and then, an efficient contact search and contact stiffness matrix assembling algorithms were proposed and implemented. The analysis code was parallelized using Open Multi-Processing (OpenMP), and the performance of the developed method was verified by simulating a series of centrifugal model shaking table experiments of a jointed rock slope.

2 OVERVIEW OF THE DEVELOPED METHOD

2.1 Theoretical Overview of Fully Implicit DDA

The DDA is an implicit type method for discontinue that analyzes the dynamic behaviors of a block system by solving the equations of motion for the whole analytical domain considering the friction.

$$\begin{pmatrix} [K_{11}] & [K_{12}] & \cdots & [K_{1n}] \\ [K_{21}] & [K_{22}] & \cdots & [K_{2n}] \\ \vdots & \vdots & \ddots & \vdots \\ [K_{n1}] & [K_{n2}] & \cdots & [K_{nn}] \end{pmatrix} \begin{pmatrix} \{\Delta d_1\} \\ \{\Delta d_2\} \\ \vdots \\ \{\Delta d_n\} \end{pmatrix} = \begin{pmatrix} \{F_1\} \\ \{F_2\} \\ \vdots \\ \{F_n\} \end{pmatrix} \tag{1}$$

Where $[K_{im}](i, m = 1, 2, \ldots, n)$ is a 6×6 matrix, and $\{\Delta d_i\}$ and $\{F_i\}$ are 6-component vectors. A diagonal submatrix $[K_{ii}]$ includes the mass and elastic stiffness of the block Ω_i, and the contact stiffness term when a contact with other block exists, and$[K_{im}](i \neq m)$ is a submatrix appears only when the blocks Ω_i and Ω_m contact each other. $\{\Delta d_i\}$ is the unknown variables vector for the block Ω_i including the translation, rotation, and strain components of the block. $\{F_i\}$ is the residual force vector consisting of the external, internal, and contact forces. Though both the original and FI-DDA construct and solve the equation of this form, the stiffness matrix has different characteristics.

The common feature between the stiffness matrix of the original DDA and that of the FI-DDA is that the submatrices at non-diagonal position (i.e., $[K_{im}](i \neq m)$) appear only when the block pair is in contact and is zero otherwise. Therefore, the stiffness matrix is generally sparse, and the linear equation solver suitable for sparse stiffness matrices is needed to solve Eq. (1). The differences appear when the sliding between the blocks occurs. In the original DDA, since the contact stiffness matrix's shear component is removed after the sliding starts, the stiffness matrix is always symmetric. On the other hand, in the FI-DDA, the linearization of the friction force is considered to achieve the quadratic convergence of the Newton-Raphson iteration, and consequently, the stiffness matrix becomes asymmetric when the sliding occurs. Therefore, FI-DDA requires a linear equation solver for asymmetric stiffness matrices.

Fig. 1 shows the calculation flow of the FI-DDA. As mentioned earlier, the introduction of the Newton-Raphson iteration in the FI-DDA increases the computational time per step because the linear equation is constructed and solved iteratively. Therefore, a parallel iterative linear equation solver for asymmetric matrix and an efficient matrix construction/assembling algorithm are needed for more efficient computation. In addition, the computational cost of the contact detection is high among other items as same as in the original DDA, that means development of an efficient parallelization scheme for the contact detection is also a key for the fast computation.

2.2 Parallelization Scheme for Fully Implicit DDA

2.2.1 Outline of Parallel Computing

Parallel computing is the process of dividing independent processes within a program's algorithm into multiple computing devices within a CPU (Central Processing Unit) or GPU (Graphics Processing Unit) and executing them simultaneously to speed up the process. In the original DDA, Yu *et al.* (2020) have succeeded in reducing analysis execution time using OpenMP, which generates processing units called threads on multiple CPU cores and executes them in parallel. Based on this, thread parallelization with OpenMP was determined to be effective for full implicit DDA in this study and was adopted. However, the parallelization scheme shown below is applicable also for other parallel programming models, e.g. MPI (Message Passing Interface) and GPGPU.

2.2.2 Introduction of Iterative Linear Equation Solver and Its Parallelization

Yu *et al.* (2020) used the Jacobi Preconditioned Conjugate Gradient (JPCG) method, an iterative solver for positive definite symmetric matrices for the parallelization of the original DDA. On the other hand, in the FI-DDA, the stiffness matrix of the linear equations is asymmetric, so the JPCG method cannot be used. For this reason, the FI-DDA has used Intel® oneAPI Math Kernel Library PARDISO, a direct solver for sparse matrices. However, the direct solver requires more memory and operations to solve the problem than the iterative method, and its applicability to large-scale calculations is relatively poor. Therefore, in this study, the Bi-Conjugate Gradient Stabilized (BiCGSTAB) method, which is an iterative solver for asymmetric matrices was newly implemented together with the Jacobi preconditioner.

The most part of the BiCGSTAB method consists of the sparse matrix-vector products (SpMV), and inner products of the vectors. While the SpMV can be easily parallelized, the inner product calculations are not straightforward. In parallel processing of the inner product computation, the inner product in each component of a vector is first computed in each thread, and finally, the results of each thread's computation are summed. Therefore, an exclusion process is required when summing, but the exclusion control tends to degrade parallel performance due to synchronous processing. In addition, the adding order of the computed results by each thread differs depending on the number of parallel threads, which causes the changes in the calculation results due to the round-off errors. In such cases, the analyses with different numbers of threads are required to check the validity of the simulated results, and the speed-up advantage of parallel computation will be lost.

Therefore, this study parallelizes the inner product calculation by the following procedure. First, the vector is divided into k sections in advance (k has the number of parallel threads as a common divisor). Then the inner product for each section is calculated in parallel. Finally, the computed results for each section are summed up in the order of the original vectors. Since the order of summing is the same when the number of threads is changed, the calculation results do not change depending on them.

2.2.3 Parallelization of Contact Detection Procedures

In the DDA, a contact between a block pair occurs when a vertex of a polygonal block penetrates into an edge of another block. Therefore, at the beginning of each time step, the contact detection procedures to search potential vertex-edge pairs (hereafter referred to as "contact pairs") that are likely to contact during the time step are conducted. Fig. 2 shows the pseudocode after parallelization.

```
Thread computation equalization operation by number of contacts in previous step
#pragma omp parallel
 #pragma omp for
  for each block Ω_i = 1:n
   for each block Ω_m = i + 1:n
    Part 1-1: Calculate distance between blocks to detect possible contact
     if (distance below threshold)
      Part 1-2: Calculate the distance between the vertex of one block and the edge of the other block to detect possible contact
       if (distance below threshold)
        Part 1-3: Record vertex/edge pairs
         Part 1-4: Count and store the number of potential contacts
 #pragma omp for
   for each contact potential cp_i = 1:cp_n
    Part 2: Detect contacts by block posture in detail
Determine contact pairs
```

Figure 2. Parallel pseudocode of the contact detection process.

```
Thread computation equalization operation by contact pairs
#pragma omp parallel for
 for each block Ω_i = 1:n
  Compute intermediate variables
   Add terms to [K_ii], [K_mm], [K_im], [K_mi], {F_i} and {F_m}
```

Figure 3. Parallel pseudocode of the constructing stiffness matrices and vectors for contact.

The contact detection s performed in two steps, the detections based on the inter-block distance (part 1) and the block posture (Part 2). The details for each step are shown below.

The contact detection based on the inter-block distance is processed by a round-robin method among all the blocks. Firstly, the block pairs located closer than the threshold is searched using the maximum and minimum coordinates in x- and y-directions of each block (Part 1-1). Then, if the blocks are sufficiently close, the distance for all vertex-edge pairs between the blocks is computed (Part 1-2), and the vertex-edge pairs whose distance is less than the threshold are stored as the potential contact pairs (Part 1-3). The detection process for each block pair is independent and parallelizable. However, since the processes in Part 1-2 and 1-3 are conducted only for the block pairs that satisfied the threshold in Part 1-1, simple parallelization for Ω_i causes an imbalance in computational load between the threads, resulting in a significant loss of parallelization efficiency. Therefore, we added a balancing process of the computational load in this study. Based on the number of potential contact pairs for each block at the previous time step, the total computational load is estimated. Then, the iterations for Ω_i are divided and assigned to the parallel threads so that the computational load is equalized among the threads.

The posture-based detection is easily parallelized because it can be performed independently for the potential contact pairs that have been already detected by the inter-block distance.

2.2.4 Parallel Construction of Contact Stiffness Matrices and Contact Force Vectors

In the FI- DDA, the contact stiffness matrix, and contact force vector are calculated for each contact pair detected by the contact detection, and then assembled to the global stiffness equation (Eq. (1)). The parallelization of the matrices and vectors computation is straightforward; however, the assembling process needs a special treatment. If a block has multiple contact points assigned to different threads, concurrent access from the several threads to the same contact stiffness term (or memory address) may occur. This means that an exclusion procedure is required for the assembling process. However, as mentioned earlier, the exclusive control may degrade parallelization efficiency.

In contrast, this study proposes a new algorithm for the computation and assemblage of contact terms (Fig. 3). First, for each block Ω_i, the indices to the contact pairs involving Ω_i in the contact pair list obtained by the contact detection are made. Then, the contact stiffness matrices and contact force vectors are calculated and assembled by an iteration with respect to the blocks. Since the memory address of the matrices and vectors are independent for each block, this process can be easily

parallelized without exclusion control. For efficient parallelization, the iteration over Ω_i is divided and assigned to the threads considering the number of contact pairs for each block so that the computational load is equalized among the threads.

3 VALIDATION OF THE DEVELOPED METHOD

Simulations of a series of centrifugal model shaking table experiments of a rock slope model were conducted to verify the applicability and parallel performance of the developed method (Naya *et al.*, 2022). In the experiment, a jointed rock slope is modeled by stacking 2081 stainless steel hexagonal bars, and the model was shaken in 14 steps with varying maximum acceleration amplitude at a centrifugal acceleration of 25 G. In the simulations, all hexagonal bars and the shaking frame were modeled as DDA blocks (Fig. 4), and the acceleration in Steps 3-14 (excluding white noise shaking) was input to the frame in a series of steps (Fig. 5). The physical properties and analytical parameters of the block were determined referring to Hashimoto & Koyama (2022) as follows: density: 8.0 cm^3, Young's modulus: 200 GPa, Poisson's ratio: 0.3, time increment: 0.00005 s, tolerable residual (energy norm) for Newton-Raphson: 1.0×10^{-15}, and allowable residual norm for BiCGSTAB: 1.0×10^{-15}. In addition, to verify the performance of the developed method, we performed five analyses under the same conditions, one using the conventional FI-DDA using a direct solver PARDISO (hereinafter referred to as the previous method) without parallelization and the other using the developed method with the number of threads varied from 1 to 4. All the computation is performed on a Supermicro computer, powered by the Intel® Core Xeon® CPU E5-2687W with a clock rate of 3.10 GHz.

Fig. 6 shows the slope profiles after shaking steps 11-14. The results of the developed method and the previous method are not identical exactly due to the error of the iterative solver. However, the validity of the developed method was confirmed by the fact that in both the experiment and analysis, the collapse began at the top of the slope from step 12 and then progressed to the right.

Fig. 7 shows the execution time for the previous method with PARDISO and the developed method with BiCGSTAB method without parallelization (i.e., single thread). The figure shows that the introduction of the iterative BiCGSTAB method resulted in a significant reduction of execution time for the equation solving part compared to the previous method with the direct solver. Therefore, the speedup by the introduction of BiCGSTAB was confirmed.

Fig. 8 shows the speed up ratio by the parallelization in the developed method. The figure shows that the developed method speeds up as the number of threads increases. Especially for the equation solver, the contact detection, and the construction of the contact stiffness and contact force terms, speed up over three times was achieved with 4 threads. The reason why the speed up ratio of others is relatively low is that it includes procedures that are difficult to parallelize due to their high information dependence, such as the output of the computed results.

Fig. 9 compares the total execution time by the previous and developed methods (1~4 threads) and the speed up ratio by parallelization. The developed method completed the analysis in 13.6 hours with 1 thread, more than twice as fast as the previous method, which took about 31.1 hours to complete the analysis. In addition, the parallel execution of the developed method successfully accelerated the computation. Specifically, the analysis was completed in approximately 4.5 hours with 4 threads, which is 6.9 times faster than the previous method.

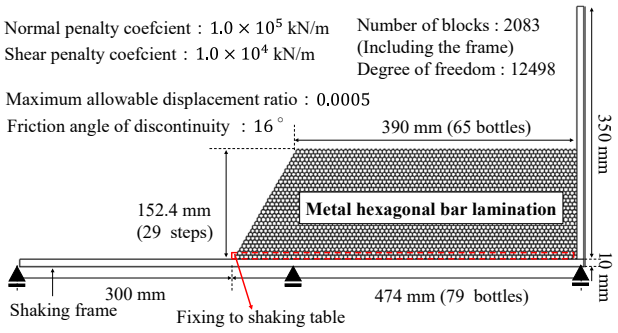

Figure 4. Analysis model and contact parameters.

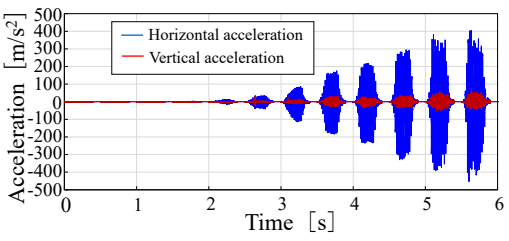

Figure 5. Input acceleration.

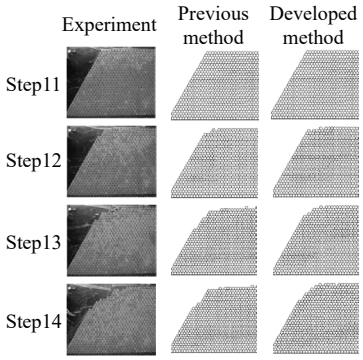

Figure 6. Changes in slope profile.

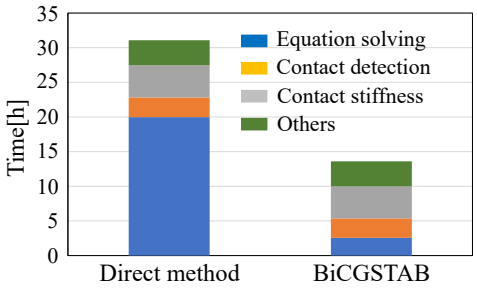

Figure 7. Comparison of execution time between the linear solvers.

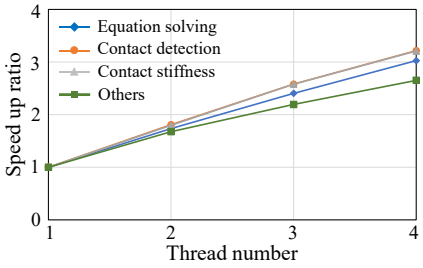

Figure 8. Speed up ratio for each part.

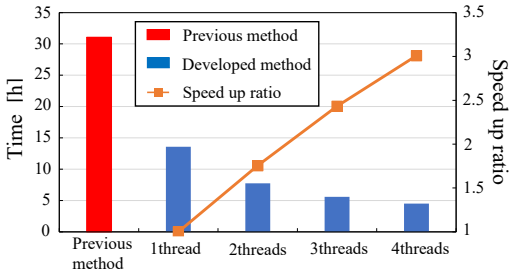

Figure 9. Comparison of total execution time.

4 CONCLUSION

In this study, we proposed a new parallelization scheme for the FI-DDA and developed the parallelized code using OpenMP. When the developed method was applied to the simulation of the centrifugal shaking table experiment, the execution time was significantly reduced compared to the previous method while reproducing the experimental results. Therefore, it was successfully shown that the developed method can analyze a large-scale rock slope with high accuracy and efficiency.

REFERENCES

Hashimoto, R., Sueoka, T., Koyama, T., Kikumoto, M. 2021: Improvement of discontinuous deformation analysis incorporating implicit updating scheme of friction and joint strength degradation, *Rock Mech. Rock Eng.*, Vol. 54, pp. 4239-4263.

Hashimoto, R. & Koyama, T. 2022: Centrifuge testing to dynamic behavior of slope model piled up steel hexagonal bar simulating discontinuous rock mass (part 13) -Evaluation by improved discontinuous deformation analysis-, *Proc. of the 48th Japanese Symposium on Rock Mechanics*, No. 25 (in Japanese).

Koyama, T., Akao, S., Nishiyama, S., Ohnishi, Y. 2009: Earthquake response analysis for rock slope using Discontinuous Deformation Analysis (DDA), *Journal of Japan Society of Civil Engineers C (geotechnical engineering)*, Vol. 65 No. 3, pp. 644–662 (in Japanese).

Naya, T., Okada, T. and Sekiguchi, A. 2022: Centrifuge testing to dynamic behavior of slope model piled up steel hexagonal bar simulating discontinuous rock mass (part 3) -Centrifugal force vibration test of slope model-, *Proc. of the 48th Japanese Symposium on Rock Mechanics*, No. 15 (in Japanese).

Shi, G. H. & Goodman, R. E. 1989: Generalization of two-dimensional discontinuous deformation analysis for forward modeling, *Int. J. Numer. Anal. Methods Geomech.,* Vol. 13, pp. 359–380.

The Japan Electric Association Nuclear Standards Committee. 2016: Technical Guidelines for Seismic Design of Nuclear Power Plants

Yu, P., Peng, X., Chen, G., Guo, L. and Zhang, Y. 2020: OpenMP-based parallel two-dimensional discontinuous deformation analysis for large-scale simulation, *Int. J. Geomech.*, Vol. 20(7), pp. 04020083-1–04020083-14.