Computers and Geotechnics 106 (2019) 217–227

Contents lists available at ScienceDirect

Computers and Geotechnics

journal homepage: www.elsevier.com/locate/compgeo

Research Paper

A hybrid MPI/OpenMP parallel computing model for spherical discontinuous deformation analysis

Yu-Yong Jiao[a,b], Qiang Zhao[b,c], Long Wang[b], Gang-Hai Huang[d], Fei Tan[a,*]

[a] Faculty of Engineering, China University of Geosciences, Wuhan, Hubei 430074, PR China
[b] State Key Laboratory of Geomechanics and Geotechnical Engineering, Institute of Rock and Soil Mechanics, Chinese Academy of Sciences, Wuhan 430071, PR China
[c] University of Chinese Academy of Sciences, Beijing 100049, PR China
[d] School of Civil Engineering, Hunan University of Science and Technology, Xiangtan 411201, PR China

ARTICLE INFO

Keywords:
SDDA
Conjugate gradient method
Message passing interface
OpenMP
Parallelization

ABSTRACT

Large-scale numerical simulation using spherical discontinuous deformation analysis (SDDA) is time-consuming, and this obstructs the application of the SDDA method for simulating the large-scale discrete spherical system. A key factor that affects the computational efficiency of the SDDA program is the solver of linear equations. To solve this problem, this paper replaces the original equilibrium equations solvers in SDDA by the conjugate gradient iterative method which is applicable for parallel computing, and three parallel computing models, i.e. pure MPI (Massage Passing Interface), pure OpenMP and hybrid MPI + OpenMP, are proposed to investigate which is better for SDDA. The parallelization algorithms are incorporated into the original SDDA source code, and the modified program is implemented on the high-performance computing clusters of Sugon TC4600. Three numerical examples are simulated for verification, and the results demonstrate that the proposed hybrid parallelization model is correct and effective.

1. Introduction

Discontinuous deformation analysis (DDA) proposed by Shi [1] is one of the most commonly used discrete element method in rock mechanics, and it is useful for investigating the behavior of mechanical systems dominated by discontinuities. This method will not encounter mathematical problem in solving any large displacement because of the relatively independence of individual block [2–4].

So far, great progress has been achieved in two-dimensional (2D) DDA [5–8], and this method has also been applied widely in many practical engineering. Hatzor et al. [9] used the DDA method to analyze the dynamic stability of jointed rock slopes, and found that some energy dissipation must be introduced to the otherwise un-damped DDA formulation. Jiao et al. employed this approach to model the stress wave propagation in jointed rock [10] and the process of reservoir-impoundment-induced landslide [11]. Wu et al. proposed a practical approach for earthquake-induced landslide simulations using DDA [12–14]. In order to investigate the effects of near-fault seismic force on landslide run-out, kinematic behavior of sliding mass was simulated by DDA, and results of the horizontal and vertical situation are in good agreement with those obtained from post-earthquake field investigation [15]. In order to enable the DDA method to be used to study the seismic dynamic response of underground caverns, Zhang et al. [16] made two modifications, which involved setting viscous boundary conditions and inputting seismic waves from the bottom in stress way. Jiang et al. [17] presented an accurate interface shear strength model, and applied the modified DDA to analyze the failure mechanisms and stability of the ancient masonry seawall along Qiantang River in China. Rizzi et al. [18] presented new analytical solutions of the classical Couplet-Heyman problem in the statics of circular masonry arches, and confirmed them by the numerical DDA computations. Ma et al. [19] replaced the original Mohr-Coulumb joint model with the displacement-dependent Barton-Bandis rock joint model, and applied the DDA code to predict the dynamic motion behavior of sliding blocks. Based on a viscous boundary and the free-field theory, Peng et al. [20] modified the DDA, and applied it to accurately simulate earthquake-induced landslide with high velocity and long runout.

To analyze actual rock engineering problems, it is not appropriate to directly apply the 2D calculation model because of typical three-dimensional (3D) effects. However, 3D DDA has focused on basis theory development until now [21–23]. The key of DDA method is the contact theory including the contact detection and open-close iteration. To reduce the complexity of the contact detection and improve the computational efficiency of the DDA program in 3D, fundamental studies of

* Corresponding author.
E-mail address: tanfei@cug.edu.cn (F. Tan).

https://doi.org/10.1016/j.compgeo.2018.11.004
Received 21 December 2017; Received in revised form 17 September 2018; Accepted 8 November 2018
Available online 14 November 2018
0266-352X/

the spherical discontinuous deformation analysis (SDDA) have commenced [24–27]. However, with the increasing element number of the numerical model to solve, the simulation tends to be very time consuming, and this obstructs the application of the SDDA method for simulating large-scale spherical system. Thus, to improve the computational efficiency of the SDDA is important.

In the DDA or SDDA program, solving simultaneous equations accounts for the highest percentage of computing time. Koo and Chern [28] adopted rigid body displacement function and preconditioned conjugate gradient solver, and reduced the computational time for DDA. Mohammad [29] compared several equation solvers in the DDA method and suggested three different ranges of solver in terms of degrees of freedom. Recently, parallel computing has developed rapidly. Based on Graphics processing unit(GPU), Xiao et al. [30] developed the parallel method accelerating DDA computing, and the total performance of DDA was improved about 2 to 10 times with different cases. Song et al. [31] proposed to accelerate the DDA using parallel Jacobi Preconditioned Conjugate Gradient (JPCG) technique on GPUs, and modeled earthquake-induced landslide. To improve the efficiency of the DDA method, Fu [32] introduced the block Jacobi (BJ) iterative method and the block conjugate gradient with Jacobi pre-processing (Jacobi-PCG) iterative method, and developed the parallel computing of the DDA linear equations based on the multi-thread and CPU-GPU heterogeneous platforms with OpenMP and CUDA, respectively.

Nowadays, shared memory storage with several multi-core CPUs is widely used in parallel numerical computation. OpenMP is one of the dominant parallel programming languages for the shared memory parallel architecture, but it is not suitable for the cluster system which adopts distributed memory parallel architecture. Meanwhile, supercomputing involving multiprocessors has become interesting to use. Cluster based computing system has emerged as a mainstream method for parallel computing in many application domains, with Linux leading the pack as the most popular operating system for clusters. These systems are typically built from symmetric multi-processor (SMP) nodes connected with high speed local area networks (LANs) or system area networks (SANs). A majority of these scientific applications are written on top of Message Passing Interface (MPI) [33]. Whether OpenMP or MPI is faster, the results are not directly comparable because MPI applies best to coarser-grained parallelism, which has less overhead, whereas OpenMP applies best to fine-grained parallelism [34]. Against this background, this paper presents three parallel computing programs, i.e. pure MPI, pure OpenMP and hybrid MPI + OpenMP, and makes a comparison of these programs in computational efficiency to find a more appropriate parallel computing way for the SDDA. The original SDDA program is not suitable for parallel computing because of the solutions are not completely independent in the existing SOR iterative method and Cholesky decomposition method. Therefore, the conjugate gradient iterative method is introduced to provide a feasible parallelizable implementation for SDDA. The parallel programming of the SDDA is operated on the high-performance computing clusters of Sugon TC4600.

2. Fundamental of the SDDA

2.1. Displacement pattern

Each sphere in SDDA is rigid and has six degrees of freedom, the displacement unknowns of the i-th element can be expressed as

$$[\mathbf{D}_i] = [\,d_x \quad d_y \quad d_z \quad r_x \quad r_y \quad r_z\,]^T, \tag{1}$$

where (d_x, d_y, d_z) indicate rigid body translations of the sphere center; (r_x, r_y, r_z) represent the rotation angles around the sphere center.

The displacement function of an arbitrary point (x, y, z) of sphere $i$can be written as

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = [\mathbf{T}_i(x, y, z)][\mathbf{D}_i], \tag{2}$$

where $[\mathbf{T}_i(x, y, z)]$ is the displacement transformation matrix of sphere i,

$$[\mathbf{T}_i(x, y, z)] = \begin{bmatrix} 1 & 0 & 0 & 0 & \bar{z} & -\bar{y} \\ 0 & 1 & 0 & -\bar{z} & 0 & \bar{x} \\ 0 & 0 & 1 & \bar{y} & -\bar{x} & 0 \end{bmatrix}, \tag{3}$$

where $\bar{x} = x\text{-}x_c$, $\bar{y} = y\text{-}y_c$, $\bar{z} = z\text{-}z_c$ and (x_c, y_c, z_c) are the coordinates of sphere i's center.

2.2. Governing equations

According to the principle of minimum potential energy, the governing equation of sphere system can be obtained from the extremum condition of the total potential energy (including different kinds of deformation potential energy, potential energy of external forces). Assuming that there are n spherical elements in the computational model, the simultaneous equilibrium equations can be expressed as follows

$$\begin{bmatrix} [K_{11}] & [K_{12}] & \cdots & [K_{1n}] \\ [K_{21}] & [K_{22}] & \cdots & [K_{2n}] \\ \vdots & \vdots & \ddots & \vdots \\ [K_{n1}] & [K_{n2}] & \cdots & [K_{nn}] \end{bmatrix} \begin{bmatrix} [D_1] \\ [D_2] \\ \vdots \\ [D_n] \end{bmatrix} = \begin{bmatrix} [F_1] \\ [F_2] \\ \vdots \\ [F_n] \end{bmatrix}, \tag{4}$$

where $[K_{ij}]$ $(i, j = 1,2, \cdots, n)$ is a 6×6 submatrix, $[K_{ii}]$ depends on the material properties of sphere i, and $[K_{ij}]$ $(i \neq j)$ depends on the interactions between sphere i and sphere j; $[D_i]$are the 6×1 submatrices of the basic unknowns of sphere i; $[F_i]$are the 6×1 submatrices of the generalized forces acting on sphere i.

In the sphere system, all the contact forces and displacement constraints of spheres are cause by the neighboring spheres, and all contact issues ultimately come down to the establishment of the simultaneous equilibrium equations. By using the minimum total potential energy principle, the effects of external forces and internal interaction among contacting spheres are both transferred into submatrices which are added to the simultaneous equations. By the same token, the volume loading, the inertia forces, the submatrices of the point loading, the displacement constraint as well as the directional constraint can be obtained easily.

2.3. Kinetic condition

The solution of Eq. (4) must satisfy no-penetration and no-tension contact constrains between spheres. If penetration happens, stiffness springs are applied at the contacts, the number, the direction as well as the location of the contact springs depend on the contact type and the detected contact status. This algorithm is termed as the penalty function (also known as Open Close Iteration, OCI) method.

After the possible contact springs are added to the two contact spheres, the submatrix of the contact springs are computed and added to the matrices of the governing equations at the corresponding positions, and then, Eq. (4) is solved. According to the computed results, the conditions of no-penetration and no-tension are checked. If the conditions of first time OCI iteration are satisfied, the computation of this time step is finished, or else, the contact springs are appended where penetration occurs, or removed where tension occurs, and the corresponding stiffness matrix is adjusted, then the next round OCI iteration begins with corrected stiffness matrix and the process continues until the conditions are satisfied. If the conditions are not satisfied within six OCI iterations, the time step will be further reduced, and the OCI procedure goes on. The flow chart of OCI is shown in Fig. 1. This repeatable process makes the DDA computation process relatively lengthy.

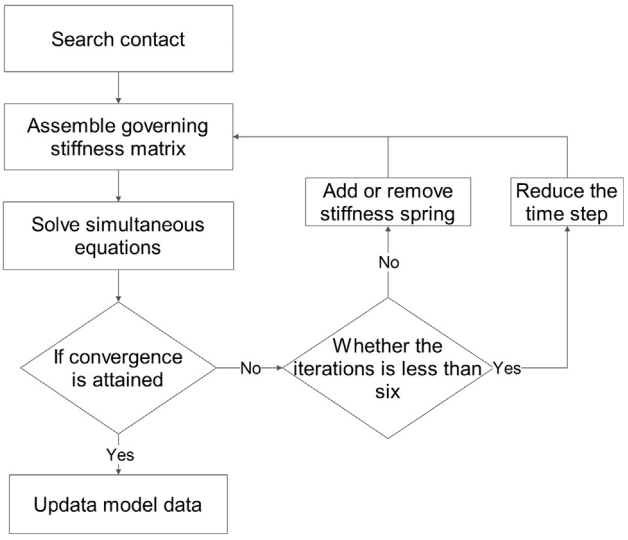


Fig. 1. Flowchart of open close iteration.

2.4. Governing equations solvers

The governing equation which possesses symmetric positive matrix is solved repeatedly in one time-step. The original SDDA program uses two governing equation solvers, i.e. the direct solver based on Cholesky decomposition method and the Gauss-Seidel method with successive over relaxation (SOR). SOR is an iterative solver for solving a linear system of equations based on an extrapolation of the Gauss-Seidel method. In the original SDDA program, the governing equation is solved by SOR method when the number of spheres is larger than 300, otherwise it is solved by Cholesky decomposition method. These two solvers work well for the SDDA program involving a few hundred spheres but for a large number of spheres they lose efficiency, and they are also not suitable for parallelization. To overcome this difficulty a more efficient solver based on the conjugate gradient (CG) iterative method is adopted. Since the global stiffness matrix, $[K]$in SDDA is symmetric and positive definite, and the CG iterative solver can be used to efficiently solve for system involving a large number of spheres. An iteration of the CG method is of the form

$$x(t) = x(t-1) + s(t)d(t) \tag{5}$$

where the new value of vector $x$is a function of the old value of vector x, a scalar step size s, and a direction vector d.

Fig. 2 shows the pseudo-code step of the sequential CG algorithm. It demonstrates that the CG method consists of matrix-vector multiplications and dot products, which are applicable for parallel computing. Before the first iteration, values of $x(0)$ and $d(0)$ are both initialized to the zero vector and $g(0)$ is initialized to $-[F]$. The tolerance threshold used by the stopping criteria is taken as $1.0e-5$, i.e. epsilon $= 1e-5$.

In order to assess the computational efficiency of the solvers, a series of simulations with different number of spheres have been analyzed. The solver speed comparison between the original SDDA program and the modified one is shown in Fig. 3. In the original SDDA program, the Gauss-Seidel method with successive over relaxation (SOR) is employed, and the solver of the modified SDDA is based on the conjugate gradient (CG) iterative method. It is clear from this figure that with increase of number of unknowns of equations, the computational time increases, and the CG method solver is found to be more efficiency for the SDDA.

3. Computer implementation

Sugon TC4600 under Linux Redhat 6.2 is used for developing the

```
for i ← 0 to n-1 do
    d[i] ← 0
    x[i] ← 0
    g[i] ← -F[i]
end for
for iter ← 1 to n do
    d1 ← Inner Product(g, g)
    g ← Matrix Vector Product(A, x)
    for i ← 0 to n-1 do
        g[i] ←g[i] - F[i]
    end for
    n1 ← Inner Product(g, g)
    if n1< epsilon
        break
    end if
    for i ← 0 to n-1 do
        d[i] ←g[i] + (n1/d1) × d[i]
    end for
    n2 ← Inner Product(d, g)
    t ←Matrix Vector Product(A, d)
    d2 ← Inner Product(g, t)
    s ←n1/d1
    for i ← 0 to n-1 do
        x[i] ←x[i] + s × d[i]
    end for
end for
```

Fig. 2. Pseudo-code step of sequential CG algorithm.

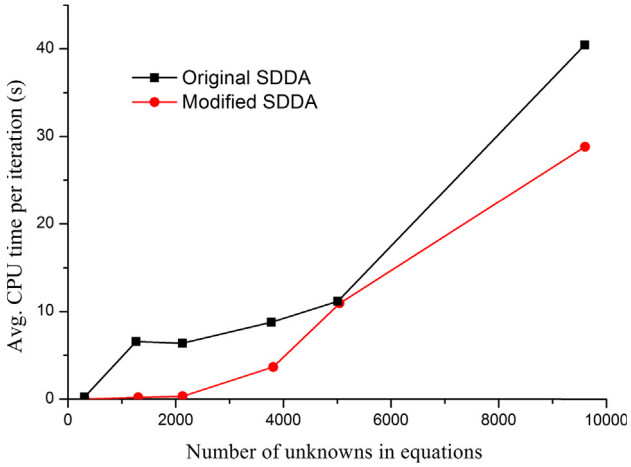


Fig. 3. Comparison of computation speed.

performances of the parallel SDDA code. It is equipped with one hundred and thirty-eight Intel E5-2670 processors and two 64T of Openstor-IB storage nodes. Because graphical display of the original SDDA code is implemented on the Windows platform, and it is not compatible with the Linux system, so, file exchange protocol (FXP) is used to implement the data file exchange across platforms, as shown in Fig. 4.

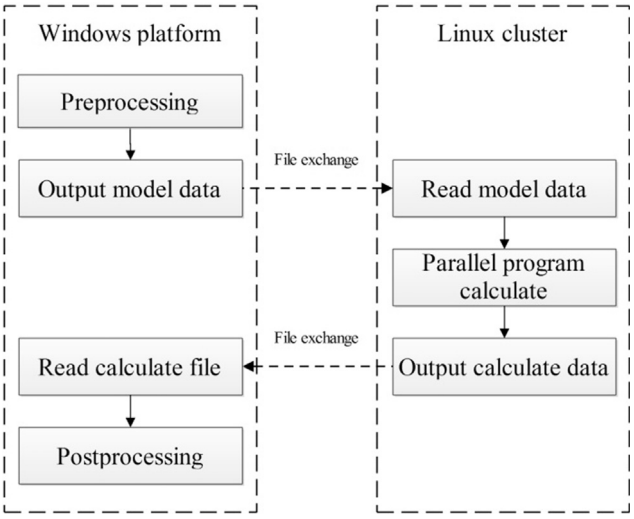


Fig. 4. Data exchange form of modified program.

3.1. Pure MPI parallel computing model

MPI runs well on a wide variety of distributed storage architectures, and it is a natural fit for multi-computers. However, MPI relies on an explicit communication among the parallel processes which requires mesh decomposition in advance due to data decomposition. Therefore, MPI may cause load balance and introduce additional time consumption.

3.1.1. Data decomposition in parallel CG method

Since the CG method consists of matrix-vector multiplications and dot products, a parallel matrix-vector multiplication algorithm based on a domain decomposition that associates a task with each row of the matrix is used. To reduce the communication time among processors, vectors are duplicated among the subtasks. Fig. 5 shows the parallel matrix-vector multiplication algorithm resulting from this domain decomposition. It demonstrates that each processor has a row and a column vector. Processor i has row i of matrix and a copy of vector, so it has all the data it needs to perform the inner product. After the inner product computation, processor i has element i of vector c. An all-gather step communicates each task's element of c to all the other processors, and the algorithm terminates. It should be noticed that the number of matrix rows must be divided by the number of processors, or the processor will receive incorrect number of data.

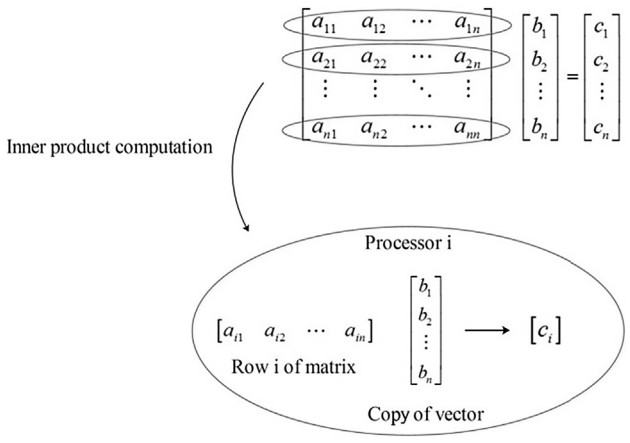


Fig. 5. Parallelization of matrix-vector multiplication.

3.1.2. MPI parallelization for SDDA

According to the data decomposition format mentioned above, Fig. 6 demonstrates the pure MPI parallelization which is designed on the basis of serial algorithm by using peer-to-peer mode and standard communication mode. In another word, the parallel SDDA program is executed on a multiprocessor cluster by creating one MPI process for each CPU on the system, and each process deals with different data obtained from the matrix. In this case all process interactions will happen via message-passing.

The flowchart of parallel SDDA on each MPI process is shown in Fig. 7. The first MPI function call made by every MPI process is the call to "MPI_INIT", which allows the system to do any setup needed to handle further calls to the MPI library. The step of "Allocate matrix to processors" makes the processors hold corresponding row elements of the matrix. Then the CG iterative solver solves the linear equations with the corresponding row elements of the matrix, as shown in Fig. 8. Finally, after a process has completed all of its MPI library calls, it calls function "MPI_FINALIZE", allowing the system to free up resources (such as memory) that have been allocated to MPI. The number of process is specified by user.

Fig. 8 presents the parallel pseudo-code of the CG iterative solver. Inner products and matrix-vector multiplications are accomplished through function calls, as shown in Fig. 9. The function "Inner_Product", when passed two vectors, returns a double precision scalar value that is the inner product of the two vectors. Because all vectors are duplicated, every process has all the values it needs to compute any inner product, and it needs no communications. The function "Matrix_Vector_Product", when passed the row elements of the matrix and a vector, returns a vector that is the product of the matrix and the vector. Vectors are duplicated, while each process is assigned a contiguous group of rows of the matrix. Hence multiplying these rows times a vector results in the solution being distributed in blocks across the set of processes. In order to gather the solution vector obtained on each processes, the function "Matrix_Vector_Product" calls the function "MPI_Gather".

3.2. Pure OpenMP parallel computing model

OpenMP is an implementation of multithreading, a method of parallelizing whereby a master thread forks a specified number of slave threads and the system divides a task among them. Fig. 10 demonstrates the parallel principle of OpenMP. It runs well on a shared storage system and is easier to implement parallelization than MPI. Fig. 11 presents the parallel pseudo-code of the matrix-vector multiplier and inner product in SDDA.

3.3. Hybrid MPI + OpenMP parallel computing model

The pure MPI parallel computing model can scale beyond one node, and has no data placement problem. But it is difficult to develop and debug, the communication is explicit, and it is very difficult to achieve the load balancing.

The pure OpenMP parallel computing model is very easy to implement parallelism, and it has low latency, high bandwidth. However, it runs only on shared memory machines, and scales within one node.

Since most architecture in high performance computing is distributed shared memory, this study presents a hybrid MPI + OpenMP parallel computing program for SDDA to further enhance calculation ability on single node, as shown in Fig. 12. The parallel pseudo-code of the matrix-vector multiplier and inner product are the same as in Fig. 11.

4. Verification examples

Based on the parallel algorithms mentioned above, the parallel CG solvers, written in C + +, have been developed and incorporated into

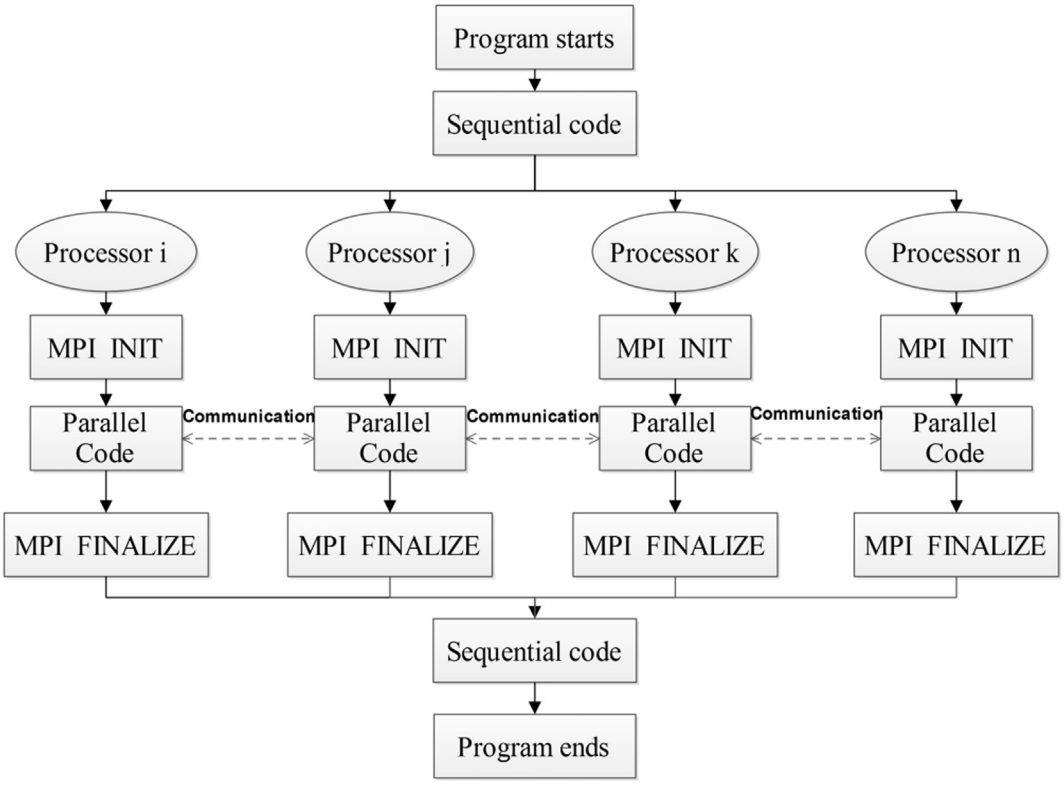


Fig. 6. Parallel model of MPI.

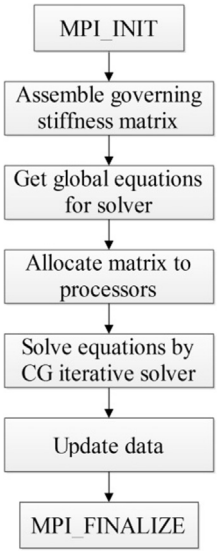


Fig. 7. Flowchart of parallel SDDA.

the original SDDA source code proposed by Jiao et al. [28]. This section presents three numerical examples to verify the correctness of the proposed parallel models and to make a comparison of pure MPI, pure OpenMP and hybrid MPI + OpenMP parallelization in computational efficiency. The common computational properties used in these examples are listed in Table 1.

The speedup ratio as one of the key indexes of parallel processing is used to analyze the efficiency of the parallel CG solver. Speedup is the ratio between sequential execution time and parallel execution time [34]:

$$Speedup = \frac{Sequential\text{execution time}}{Parallel\text{execution time}}$$

```
void CG (int id, int p, double *A, double *b, double *x, long n)
{
for i ←0 to n-1 do
      d[i] ←0
      x[i] ←0
      g[i] ←F[i]
end for
for iter ← 1 to n do
      d1 ← Inner_Product(g, g, n)
      Matrix_Vector_Product(id, p, A, x, g, n)
      MPI_Bcast(g, n, MPI_DOUBLE, 0, MPI_COMM_WORLD)
      for i ← 0 to n-1 do
            g[i] ← g[i] - F[i]
       end for
      n1 ← Inner_Product(g, g, n)
      if n1< epsilon
            break
      end if
      for i ← 0 to n-1 do
            d[i] ← g[i] + (n1/d1) ×d[i]
      end for
      n2 ← Inner_Product(d,g)
      Matrix_Vector_Product(id, p, A, d, tmpvec)
      MPI_Bcast(tmpvec, n, MPI_DOUBLE, 0, MPI_COMM_WORLD)
      d2 ← Inner Product(d,tmpvec)
      s ← n2/d2
      for i ← 0 to n-1 do
            x[i] ← x[i] + s×d[i]
      end for
end for
}
```

Fig. 8. Pseudo-code of the parallel CG iterative method.

```
void Matrix_Vector_Product(int id, int p, double *m, double *v, double *g, long n)
{
    int size ← n/p;
    double *local_sum ← (double *) malloc(size*sizeof(double));
    for i ← 0 to size-1 do
      local_sum[i] ← 0
        for j ← 0 to n-1 do
          local_sum[i] ← local_sum[i] + v[j] * m[i*n+j]
        end for
    end for
    MPI_Gather(&local_sum[0], size, MPI_DOUBLE, g, size, MPI_DOUBLE, 0,
                    MPI_COMM_WORLD)
}

double Inner_Product(double *a, double *b, long n)
{
    double rs ← 0
    for i ← 0 to n-1 do
        rs ← a[i] * b[i]
    endfor
    return rs
}
```

Fig. 9. Pseudo-code of the function called by CG solver.

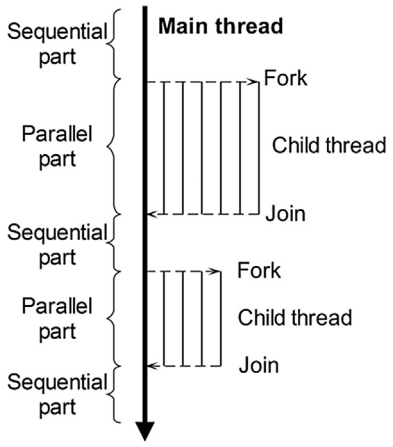


Fig. 10. Parallel principle of OpenMP.

4.1. Numerical tests of free falling body

This example intends to verify the correctness of the modified solver. The spherical element, with a mass of 10 kg and radius of 0.5 m, respectively, is falling under gravity, as shown in Fig. 13. Time step $t = 0.001$. Pure MPI parallel computing model is employed.

Fig. 14 compares the obtained velocities by the original solver (with the method of Cholesky and SOR), sequential CG solver and parallel CG solver. It can be seen that the results of the two CG solvers converge to a stable state at time $t = 6.8$ s, and the original one converges to a stable state at $t = 7.1$ s. The motion curves obtained by the three solvers are consistent, indicating that the two CG solvers are correct and faster.

4.2. Simulation of landslide

Landslide is one of the serious natural hazards in mountainous areas and represents a major threat to infrastructure, transportation lines, and people. In this section, a simple landslide is simulated. The landslide plan is shown in Fig. 15. The sliding surface is about 103 m in length, and the shearing opening is about 57 m in width. The computational model is shown in Fig. 16, and the slide mass consists of 1526 sphere

```
int size ← n/p
double *local_sum ← (double *)malloc(size*sizeof(double))
#pragram omp parallel for private (j, tmp)
for i ← 0 to size-1 do
    local_sum[i] ← 0
    tmp ← 0
    for j ← 0 to n-1 do
        local_sum[i] ← local_sum[i] + v[j]*m[i*n+j]
    end for
end for
```

(a) matrix-vector multiplier

```
double rs← 0
#pragram omp parallel for
for i ← 0 to n-1 do
    rs ← a[i]*b[i]
end for
return rs
```

(b) Inner product

Fig. 11. Pseudo-code of the call function.

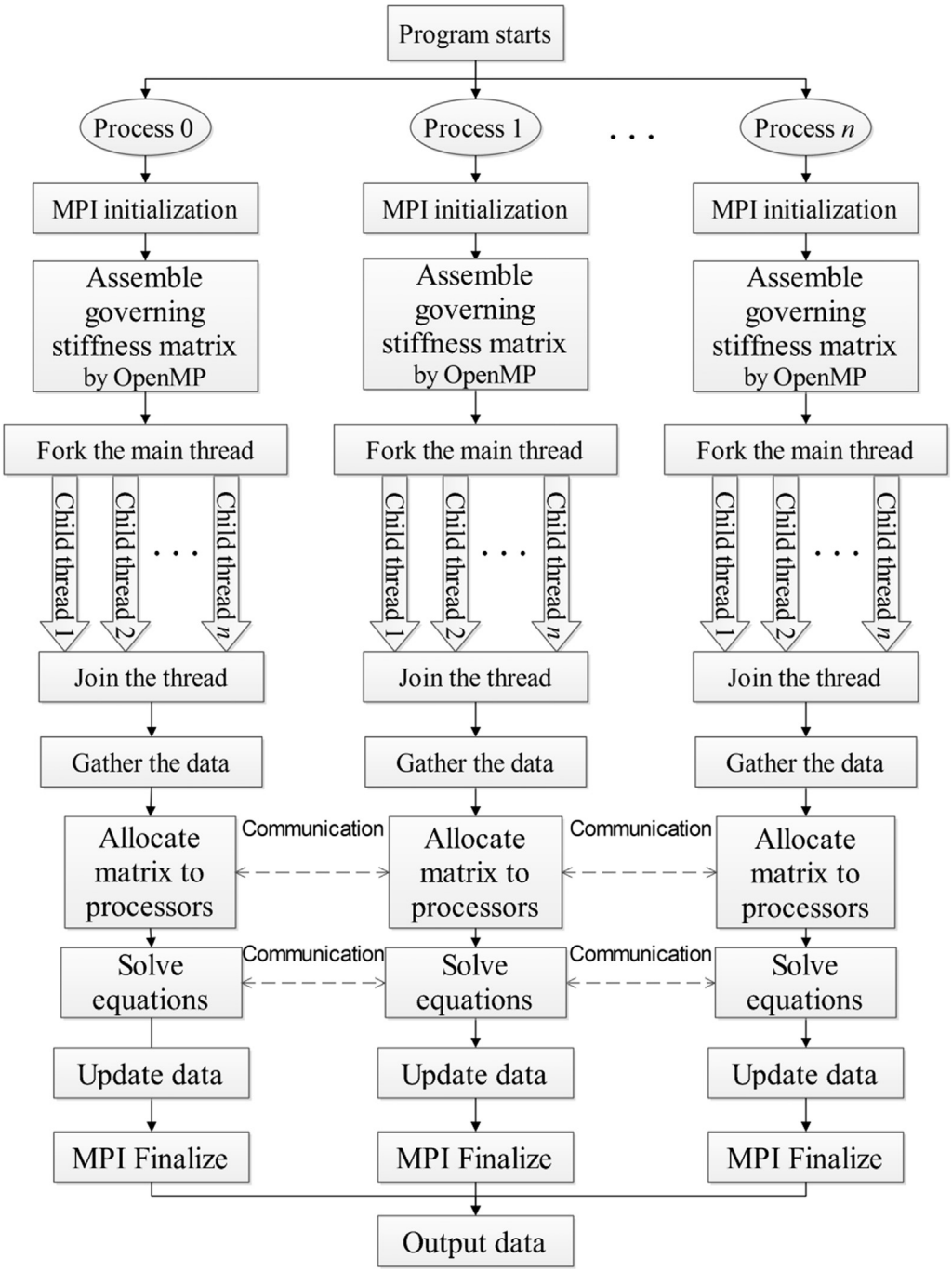


Fig. 12. Parallel model of hybrid MPI + OpenMP.

Table 1
Computational properties.

Parameters	Values
Gravitational acceleration (m/s^2)	9.8
Normal spring stiffness (GN/m)	1
Shear spring stiffness (GN/m)	0.25

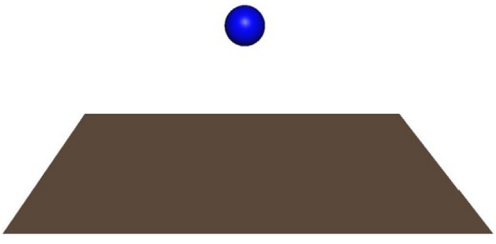


Fig. 13. Model of free falling body.

elements without bonding strength. The spherical elements, with mass of 2700 kg and radius of 1 m, respectively, are sliding under gravity. Time step $t = 1 \times 10^{-5}$ s. Fig. 17 shows the simulated results at different calculating time steps. The results show that the parallel SDDA model can simulate the motion of landslide.

In order to analyze the parallel efficiency, different numbers of

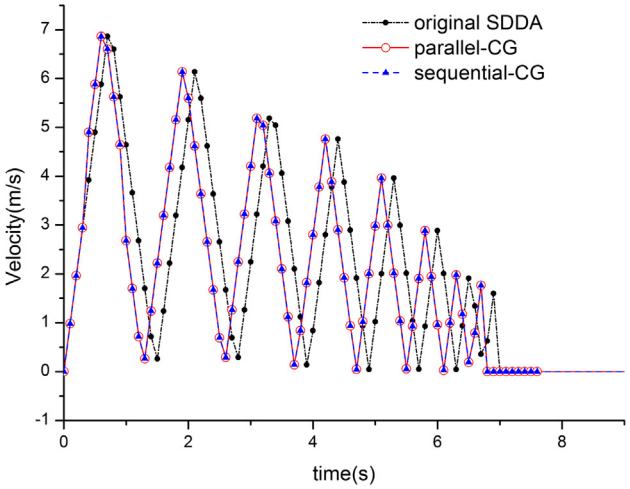
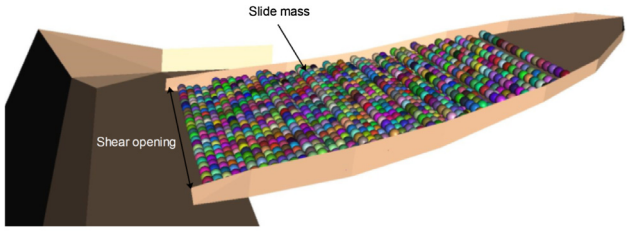
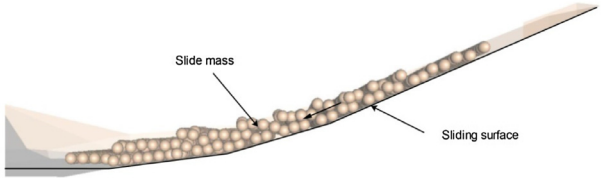


Fig. 14. Comparison of velocity-time curves in Example 1.

Fig. 15. Plan of landslide.

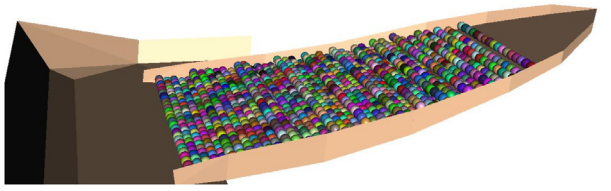


(a) Model of landslide

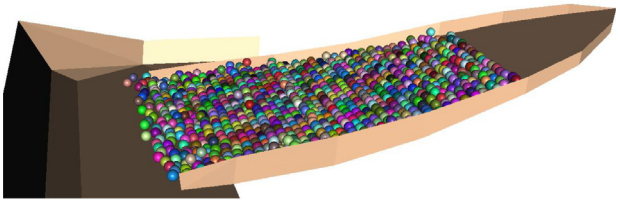


(b) Side view of landslide model

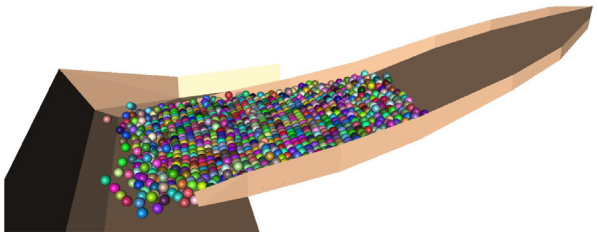
Fig. 16. Model of landslide.



(a) Step = 61002



(b) Step = 201302



(c) Step = 400802

Fig. 17. Simulated results of landslide.

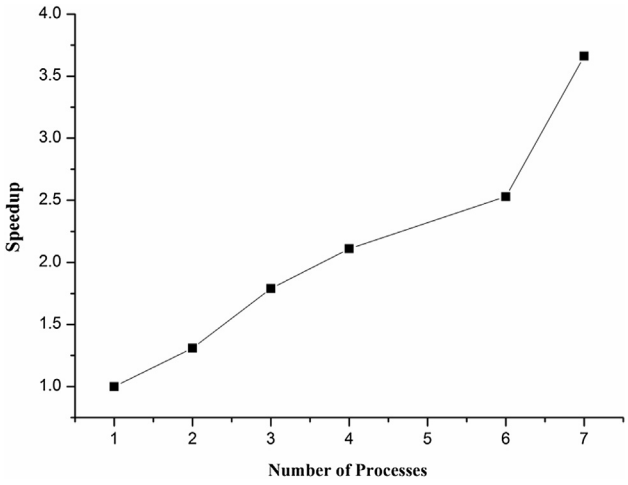


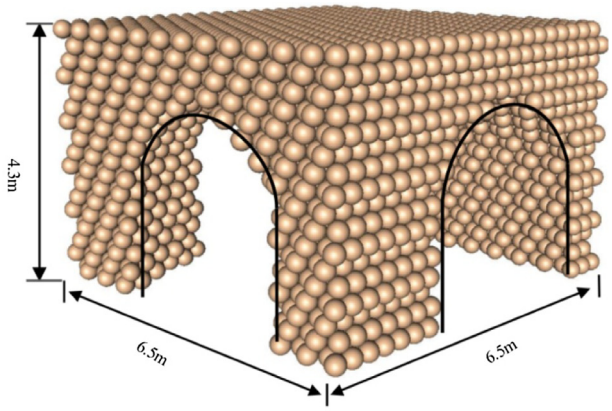
Fig. 18. Speed-up ratio results of the parallel CG solver with pure MPI computing model.

processors are employed. The speedup ratio is shown in Fig. 18. It clearly highlights that the parallel speedup ratio increases with the number of processes, and the effect of speedup is very obvious with pure MPI computing model.

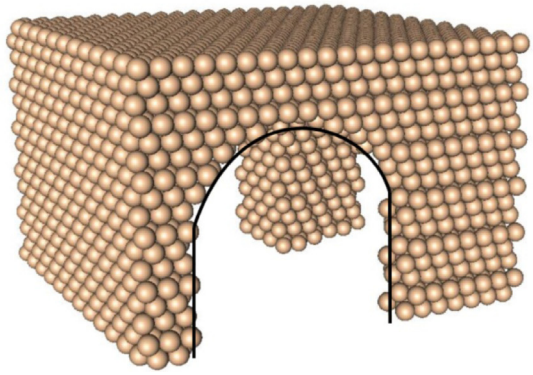
4.3. Simulation of tunnel collapse

To further verify the ability of the proposed parallel SDDA in modeling large scale problem, a model with two cross tunnels was constructed and simulated. The tunnel, with a shape of straight wall and semi-circle, is 2.8 m in width and 3.2 m in height. The calculating model, with 4161 spherical elements, is 6.5 × 6.5 × 4.3 m, as shown in Fig. 19. The properties of the spherical element are: the density $\rho = 2700\ \mathrm{kg/m^3}$, the normal stiffness $k_n = 5 \times 10^8\ \mathrm{N/m}$, the shear stiffness $k_s = 1.25 \times 10^8\ \mathrm{N/m}$, and the radius $r = 0.2\ \mathrm{m}$.

The mechanical properties are: the tensile strength $T_0 = 10\ \mathrm{MPa}$,



(a) Front view



(b) Back view

Fig. 19. Model of tunnel collapse.

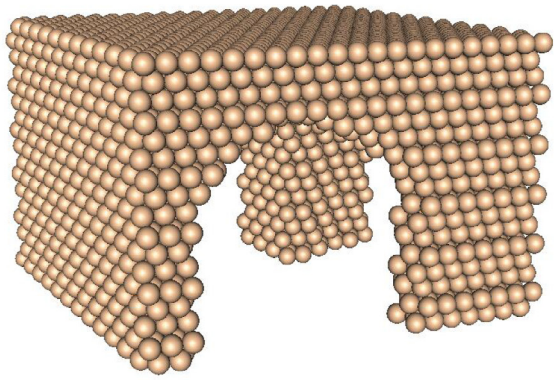
the cohesion $c = 12$ MPa, and the internal friction angle $\varphi = 30^{\circ}$. Time step $t = 1 \times 10^{-5}$. The boundary conditions are: the vertical load $F = 100$ MPa, the outer faces of the tunnel are fixed.

The simulated results at different calculating time steps are shown in Fig. 20. It can be seen that under the compression forces, the modified SDDA program with parallel CG solver can simulate the problem of tunnel collapse, effectively.

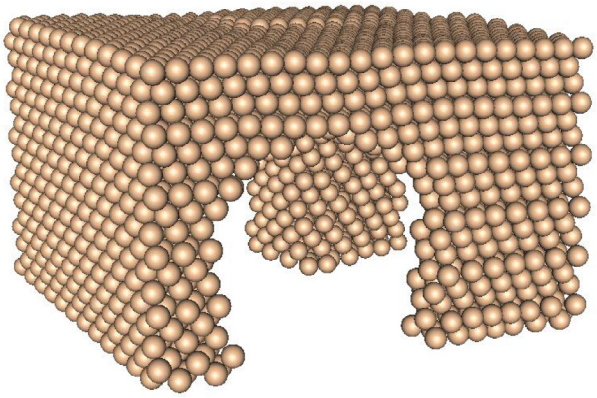
Fig. 21 shows the runtimes of solving linear equations with pure MPI parallel computing model and pure OpenMP parallel computing model. As can be seen in the figure, For this example, the computation speed of pure OpenMP parallel computing model is faster, in which 2, 4, 6 threads are employed.

In order to analyze the parallel efficiency, different numbers of processors are employed, and different numbers of threads are also used in hybrid MPI + OpenMP parallel computing model, the runtime of linear equations solving are demonstrated in Fig. 22. Table 2 lists the speedup of pure MPI and hybrid MPI + OpenMP with different numbers of processors and threads. In this case, the effect of hybrid parallel model is better when the number of processes is less than 6. When the number of processes comes up to 12, the effect of hybrid parallel model with more than 2 threads is worse than the pure MPI parallel mode. Besides, it can be seen that the maximum speedups of pure MPI, pure OpenMP and hybrid MPI + OpenMP are 3.91, 2.85 and 6.73, respectively. Therefore hybrid parallel programming is better than the other two pure parallel models, and it can further enhance the computational efficiency for the SDDA.

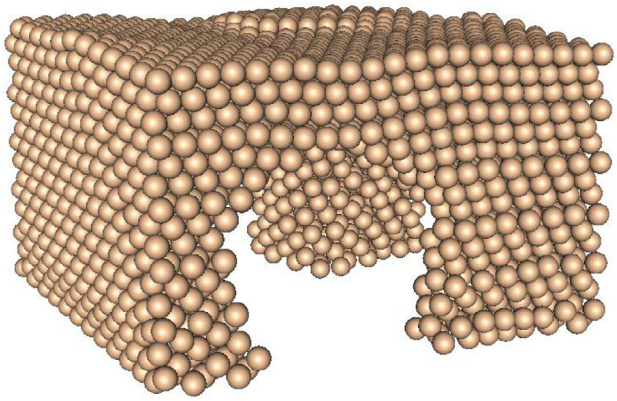
Data communication has also a major effect on the parallel efficiency. The time-consuming of getting data in parallel computing is shown in Fig. 23. It can be seen that time-consuming is more as the number of processes is more, and the time-consuming do not increasing



(a) Step = 0



(b) Step = 1790



(c) Step = 3734

Fig. 20. Simulated results of tunnel collapse.

when the number of processes comes up to 4, the possible reason is that the load balancing is implemented. Moreover, the parallel efficiency of hybrid MPI + OpenMP model is higher than the pure MPI model.

To illustrate the implementation and performance of the present method for large scale problem, we employ more spherical elements, up to 10,660 elements, to simulate the tunnel collapse. The result is shown in Fig. 24. It can be seen that the computational efficiency of the parallel SDDA is higher, the speedup is about 7.

5. Concluding remarks

Since SDDA uses an implicit solution scheme which is unconditionally stable for any time-step size, larger time-step size can be

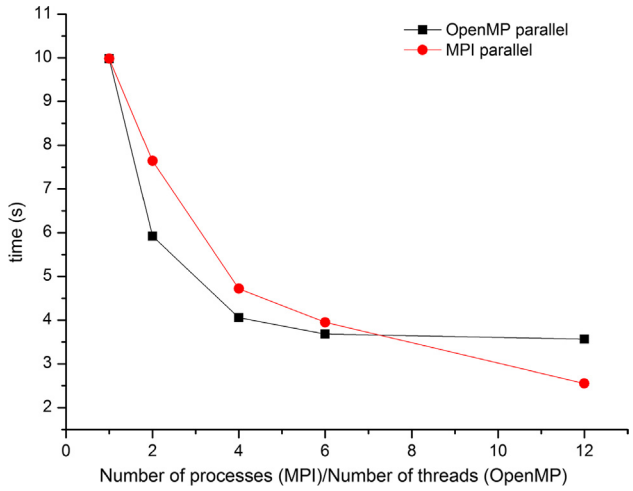


Fig. 21. Comparison of efficiency between pure MPI model and pure OpenMP model.

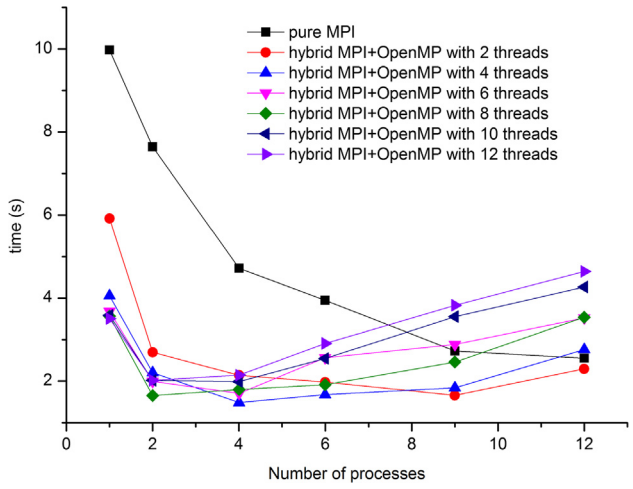


Fig. 22. Comparison of efficiency between pure MPI model and hybrid parallel model.

Table 2
Speedup results of hybrid MPI + OpenMP.

Processes	Threads						
	1	2	4	6	8	10	12
1	1.00	1.69	2.46	2.71	2.80	2.78	2.85
2	1.31	3.7	4.52	5.00	6.03	4.95	4.95
4	2.11	4.66	6.73	5.84	1.54	5.01	4.65
6	2.53	5.05	5.94	3.9	5.21	3.92	3.43
9	3.66	6.02	5.42	3.46	4.05	2.81	2.61
12	3.91	4.34	3.61	2.83	2.82	2.34	2.15

used. However, the open-close iteration restricts the time-step size, and it requires a number of iterations to satisfy contact constrains and a large number of linear systems of equations in each time-step to be solved. These imply that the calculate process in SDDA will require more time. Therefore, to improve the computational efficiency for SDDA, this paper presents three parallel computing models which run on Linux cluster platform, and the following conclusions are determined.

The CG iterative method is used to overcome the shortcomings of the original SDDA (with the solver of Cholesky decomposition and SOR iterative) which is unsuitable for parallel computing. The efficiency

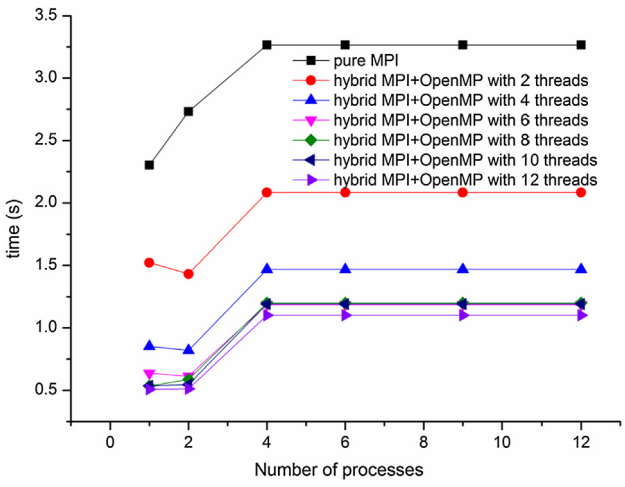


Fig. 23. Time-consuming of getting data in parallel computing.

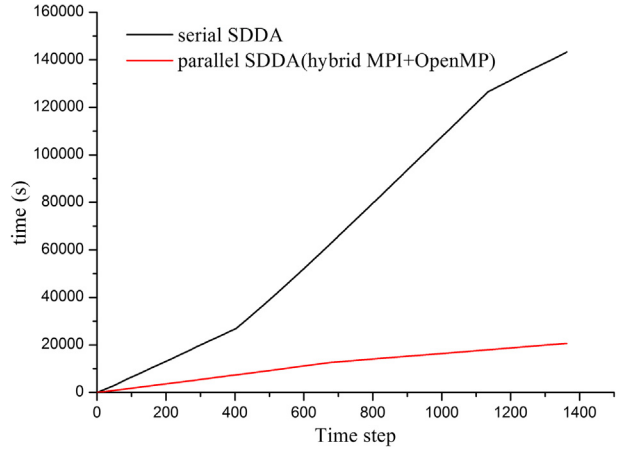


Fig. 24. Comparison of time-consuming between serial SDDA and parallel SDDA using 10,660 elements.

analyses of the solvers are investigated, and the results demonstrate that the CG iterative solver is a better choice for SDDA to solve the linear equations.

Parallel computing model for solving the linear equations in SDDA is implemented by pure MPI, pure OpenMP and hybrid MPI + OpenMP, respectively. Through the simulations of three verification examples, the correctness of the proposed parallel computing models is proved, and the parallel efficiency analyses demonstrate that the multiple parallel programming can further enhances the computing efficiency for SDDA.

However, the pure MPI and the hybrid MPI + OpenMP programs proposed for SDDA remain problematic because of the problems of load balance and communication between processes. Further studies are needed to improve the number of parallel process for achieving higher speedup.

Acknowledgement

This work was supported by the National Basic Research Program of China (“973” Project) (Grant Nos. 2014CB046904 & 2014CB047101), and the National Natural Science Foundation of China (Grant Nos. 41731284, 51479191, 11672360 & 11402280). It was also funded by the Fundamental Research Funds for the Central Universities, China University of Geosciences (Wuhan) (No. CUG170645).

References

[1] Shi GH. Discontinuous deformation analysis-a new numerical model for the static and dynamic of block systems (PhD.thesis). Berkeley: University of California; 1988.
[2] Hatzor YH, Feintuch A. The validity of dynamic block displacement prediction using DDA. Int J Rock Mech Min 2001;38:599–606.
[3] Yagoda-Biran G, Hatzor YH. Benchmarking the numerical discontinuous deformation analysis method. Comput Geotech 2016;71:30–46.
[4] Doolin DM, Sitar N. Time interation in discontinuous deforamtion analysis. J Eng Mesh 2004;130(3):249–58.
[5] Bao H, Zhao Z. The vertex-to-vertex contact analysis in the two-dimensional discontinuous deformation analysis. Adv Eng Softw 2012;45(1):1–10.
[6] Wang LZ, Jiang HY, Yang ZX, Xu YC, Zhu XB. Development of discontinuous deformation analysis with displacement-dependent interface shear strength. Comput Geotech 2013;47(1):91–101.
[7] Zheng H, Li X. Mixed linear complementarity formulation of discontinuous deformation analysis. Int J Rock Mech Min Sci 2015;75:23–32.
[8] Zheng H, Zhang P, Du X. Dual form of discontinuous deformation analysis. Comput Methods Appl Mech Engrgy 2016;305:196–216.
[9] Hatzor YH, Arzi AA, Zaslavsky Y, Shapira A. Dynamic stability analysis of jointed rock slopes using the DDA method: king Herod's Palace, Masada, Israel. Int J Rock Mech Min Sci 2004;41(5):813–32.
[10] Jiao YY, Zhang XL, Zhao J, Liu QS. Viscous boundary of DDA for modeling stress wave propagation in jointed rock. Int J Rock Mech Min 2007;44:1070–6.
[11] Jiao YY, Zhang HQ, Tang HM, Zhang XL, Adoko AC, Tian HN. Simulating the process of reservoir-impoundment-induced landslide using the extended DDA method. Eng Geol 2014;182:37–48.
[12] Wu JH, Lin JS, Chen CS. Dynamic discrete analysis of an earthquake-induced large-scale landslide. Int J Rock Mech Min Sci 2009;46(2):397–407.
[13] Wu JH. Seismic landslide simulations in discontinuous deformation analysis. Comput Geotech 2010;37(5):594–601.
[14] Wu JH, Chen CH. Application of DDA to simulate characteristics of the Tsaoling landslide. Comput Geotech 2011;38(5):741–50.
[15] Zhang Y, Chen G, Zheng L, Li Y, Wu J. Effects of near-fault seismic loadings on run-out of large-scale landslide: a case study. Eng Geol 2013;166(8):216–36.
[16] Zhang Y, Fu X, Sheng Q. Modification of the discontinuous deformation analysis method and its application to seismic response analysis of large underground caverns. Tunn Undergr Space Technol 2014;40:241–50.
[17] Jiang H, Wang L, Li L, Guo Z. Safety evaluation of an ancient masonry seawall structure with modified DDA method. Comput Geotech 2014;55(55):277–89.
[18] Rizzi E, Rusconi F, Cocchetti G. Analytical and numerical DDA analysis on the collapse mode of circular masonry arches. Eng Struct 2014;60(347):241–57.
[19] Ma S, Zhao Z, Nie W, Nemcik J, Zhang Z, Zhu X. Implementation of displacement-dependent barton-bandis rock joint model into discontinuous deformation analysis. Comput Geotech 2017;86:1–8.
[20] Peng Y, Yu P, Zhang Y, Chen G. Applying modified discontinuous deformation analysis to assess the dynamic response of sites containing discontinuities. Eng Geol 2018;246:349–60.
[21] Wu W, Zhu H, Zhuang X, Ma G, Cai Y. A multi-shell cover algorithm for contact detection in the three dimensional discontinuous deformation analysis. Theor Appl Fract Mech 2014;72:136–49.
[22] Zhang H, Liu SG, Zheng L, Zhong GH, Lou S, Wu YQ, et al. Extensions of edge-to-edge contact model in three-dimensional discontinuous deformation analysis for friction analysis. Comput Geotech 2016;71:261–75.
[23] Zheng F, Jiao YY, Zhang XL, Tan F, Wang L, Zhao Q. Object-oriented contact detection approach for three-dimensional discontinuous deformation analysis based on entrance block theory. Int J Geomech 2016;17(5):E4016009.
[24] Zhao SL. Development of three dimensonal spherical discontinuous deformation analysis for granular materials (PhD.thesis). Raleigh: North Carolina State University; 2000.
[25] Beyabanaki SAR, Bagtzoglou AC. Three-dimensonal discontinous deformation analysis (3-D DDA) method for particulate media application. Geomech Geoeng 2012;7(4):239–53.
[26] Jiao YY, Huang GH, Zhao ZY, Zheng F, Wang L. An improved three-dimensional spherical DDA model for simulating rock failure. Sci China Tech Sci 2015;58:1533–41.
[27] Wang L, Jiao YY, Huang GH, Zheng F, Zhao ZY, Tan F. Improvement of contact calculation in spherical discontinuous deformation analysis. Sci China Tech Sci 2017;59:1–7.
[28] Koo CY, Chern JC. Modification of the DDA method for rigid block problem. Int J Rock Mech Min Sci 1998;35(6):683–93.
[29] Khan SM. Investigation of discontinous deformation analysis for application in jointed rock masses (PhD.thesis). Toronto: University of Toronto; 2010.
[30] Xiao Y, Miao Q, Huang M, Wang Y, Xue J. Parallel computing of discontinuous deformation analysis based on graphics processing unit. Int J Geomech 2016;17(5):E4016010.
[31] Song Y, Huang D, Zeng B. GPU-based parallel computation for discontinuous deformation analysis (DDA) method and its application to modelling earthquake-induced landslide. Comput Geotech 2017;86:80–94.
[32] Fu XD, Sheng Q, Zhang YH, Chen J. Investigation of highly efficiency algoritm for solving linear equations in the discontinous deformation analysis method. Int J Numer Anal Meth Geomech 2016;40(4):469–86.
[33] Quinn MJ. Parallel programming in C with MPI and OpenMP. New York: McGraw Hill; 2003.
[34] Bova SW, Breshears CP, Gabb H, Kuhn B, Magro B, Eigenmann R, et al. Parallel programming with message passing and directives. Comput Sci Eng 2001;3(5):22–37.