Int J Coal Sci Technol (2014) 1(3):356–363
DOI 10.1007/s40789-014-0040-3

Analysis on intersections between fractures by parallel computation

Zhiyu Li · Mingyu Wang · Jianhui Zhao · Xiaohui Qiao

Received: 20 May 2014 / Revised: 6 August 2014 / Accepted: 6 August 2014 / Published online: 31 October 2014

Abstract The discrete fracture network model is a powerful tool for fractured rock mass fluid flow simulations and supports safety assessments of coal mine hazards such as water inrush. Intersection analysis, which identifies all pairs of intersected fractures (the basic components composing the connectivity of a network), is one of its crucial procedures. This paper attempts to improve intersection analysis through parallel computing. Considering a seamless interfacing with other procedures in modeling, two algorithms are designed and presented, of which one is a completely independent parallel procedure with some redundant computations and the other is an optimized version with reduced redundancy. A numerical study indicates that both of the algorithms are practical and can significantly improve the computational performance of intersection analysis for large-scale simulations. Moreover, the preferred application conditions for the two algorithms are also discussed.

Keywords Fracture intersections · Discrete fracture network · Intersection analysis · Parallel computing

1 Introduction

Water inrush is one of the most concerning safety problems in coal production. Fractures act as channels for fluid flows that are highly relevant to this type of disaster. The discrete fracture network (DFN) model is a powerful tool for fluid flow simulations in fractured rock mass and can provide invaluable information to assist safety assessments.

To create a DFN model, many independent yet related procedures are involved. Intersection analysis is one of these procedures and is used to identify all pairs of intersected fractures (PIF) in the region of study–a basic step to construct an interconnected fracture network. Because the DFN model is an application of Monte Carlo simulation technology, tens or hundreds of complete models runs are usually performed to obtain a stable result. Thus, it is necessary to require each of the procedures, including intersection analysis, to be computationally efficient and seamless interface with other procedures in modeling. For example, a mere 10-min performance gain in a single model run is of great practical significance because it can save nearly 17 h for 100 repeated simulations.

The essence of intersection analysis is quantifying the spatial relation between two individual fractures. Because a precise portrayal of the shape of a real fracture is often impractical, some assumptions have to be adopted for mathematical modeling. One of the most prevalent approaches is the Baecher's model (Baecher et al. 1977) in which a fracture is simplified to be a three-dimensional disc defined by its center point, radius, orientation and aperture. Thus, the task for intersection analysis is transformed into a geometrical computation on pairs of disc-shaped objects. Although the analytical solution for this problem is explicitly defined in solid geometry and has been well discussed in the field of fracture simulation (Gilmour et al. 1986; Song and Xu 2004; Li et al. 2007), its applications

Z. Li · M. Wang (✉) · J. Zhao · X. Qiao
College of Resources and Environment, University of Chinese Academy of Sciences, Beijing 100049, China
e-mail: mwang@ucas.ac.cn

J. Zhao
College of Computer and Information Engineering, Henan University, Kaifeng 475001, China

are often limited to theoretical analyses or small scale projects due to its computational expense.

A revised method was proposed in which the entire process of intersection analysis can be implemented in a stepwise manner (Yu et al. 2006). The concept of this approach is based on the theory of computer graphics, where each fracture is wrapped inside an individual bounding box, and the spatial relation between the boxes is analyzed in the first step, a procedure known as 'coarse analysis'. If the two bounding boxes intersect, a 'precise analysis' (the second step) on the real disc-shaped fractures is then performed. Because the computation cost for a single analysis on one pair of bounding boxes is far less than that of the fractures themselves, the stepwise procedure reduces the total execution time of a high-cost algorithm (precise analysis) by introducing a preprocessing technique such as a low-cost algorithm (coarse analysis) that acts as a filter. This approach can result in a considerable performance gain compared to the traditional method where the precise analysis is used alone.

Moreover, several modifications were proposed to optimize the procedure of intersection analysis. Liu et al. (2011) introduced the concept of spatial indexing from GIS and incorporated an R tree to accelerate the coarse analysis. Fadakar et al. (2011) designed a general framework for intersection analysis that handles polygon fractures, and more detailed algorithm can be found at Einstein and Locsin (2012). Liu et al. (2013) applied a spatial database to efficiently structure and manage fracture data. Li et al. (2014) performed an analysis on the relation between fractures' predominant orientations and the actual performance of the intersection analysis, presenting some practical suggestions on tuning the algorithm.

This paper is proposed to introduce parallel computing technology into the fracture intersection analysis procedure, extending its scope of application to large-scale problems. Section 2 describes the basic requirements for the parallel algorithm design, Sect. 3 presents an independent parallel algorithm with redundant computations, Sect. 4 presents an optimized version with reduced redundancy, Sect. 5 presents the numerical study, and Sect. 6 provides conclusions for this study.

2 Basic concepts

The basic idea behind parallel computing is breaking down a complete task into several sub-tasks and executing them concurrently. A parallel algorithm should address the potential side effects introduced by task segmentation and guarantee a result that is identical to the original algorithm (serial version). The actual performance of a specific parallel algorithm depends on the degree of parallelism of the underlying problem and the procedures used to implement the computational logic. Moreover, the solution to a real-world problem often requires a combination of different algorithms (such as in the DFN simulation). Thus, a practical scheme for parallel computing should be designed in view of the entire problem, not simply for a single algorithm alone.

The intersection analysis of fractures is a standalone procedure, but it is also one link of a complete simulation task. For a certain region of study, the number of fractures, N_f, is fixed. The workflow of the intersection analysis is essentially an enumeration process in which each pair of fractures is selected and successively tested against intersection. Thus, there are a total number of $N_f(N_f-1)/2$ pairs of fractures for the analysis. A simple method to parallelize this procedure is evenly dividing (as possible) the N_f fractures into N_{group} groups according to the fractures' ordinal numbers and selecting each fracture from the group to test it against every other fracture from the entire fracture pool. This approach exactly distributes the required analysis of all pairs of fractures into several groups. Thus, the complete task can be solved in parallel without redundancy with the total number of calculations equal to $N_f(N_f-1)/2$.

However, the following drawbacks exist in this approach:

(1) This approach is a pure parallel algorithm in view of the computation itself and ignores the physical reality of the underlying problem. Although the fractures have been divided into groups, each group still requires access to the entire fracture pool when performing the analysis. It is known that fractures are geographic objects with spatial features, and there are surely higher possibilities of intersections among fractures that are located in the same or nearby regions than those that are separated from each other. The scheme that groups the fractures by their ordinal numbers cannot make use of this prior knowledge.

(2) Although it is a parallel procedure, the analysis results of all the groups have to be centralized and subsequently redistributed. This is because the procedures before and after this step are often parallelized by partitioning the region of study according to some geological properties (e.g., parameter assignment for geologically homogeneous zones, zonal flow path identification and flow fluid simulations along hydraulic gradients). Thus, the inconsistency in parallel computing designs between the related procedures may influence the algorithm's performance as a result of excessive data transformations and inter-process communications.

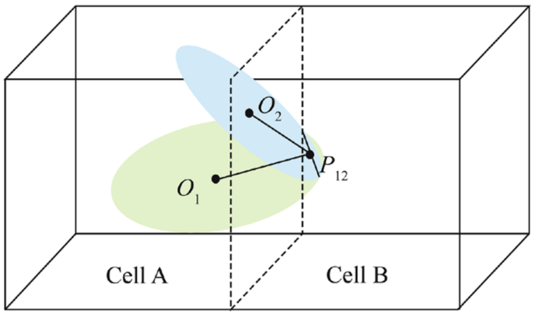

Fig. 1 Sketch of two boundary fractures in a two-cell partition study region

Therefore, it is necessary to design a parallel intersection analysis algorithm by partitioning the region of study, where each processor only requires a portion of the data during the analysis process. The next two sections present two parallel algorithms. The first algorithm is straightforward and easy to implement but contains a certain amount of redundant computations. The second algorithm is an optimized version with considerably less redundancy.

3 An independent approach with redundant computations

The study region can be partitioned in one, two, or three directions along the coordinate axis, yielding multiple adjacent boxes called grid cells. Each cell is assigned with a certain amount of fractures and is typically associated with one processor. Next, a parallel computation is performed such that each cell conducts an intersection analysis on its own fracture data.

Figure 1 shows a two-cell partition in the region of study where fractures O_1 and O_2 are crossing the boundary between cells A and B and intersecting with each other. The point P_{12} is the mid-point of the two fractures' intersected line.

Much attention is required for a fracture intersecting a cell boundary (called a boundary fracture) for two reasons: (1) it is these boundary fractures that establish the connections between different cells, which is of great hydrogeological interest for simulations, and (2) an important step in parallel intersection analysis is properly addressing these boundary fractures because they are related to more than one cell and may result in an overestimation or underestimation on the total number of PIFs.

As shown in Fig. 2, the fractures that intersect the center cell are grouped into a fracture set and assigned to this cell. An independent analysis procedure is then performed in this cell to identify all the PIFs it owns. At the same time, other cells analyze their own fracture sets in parallel.

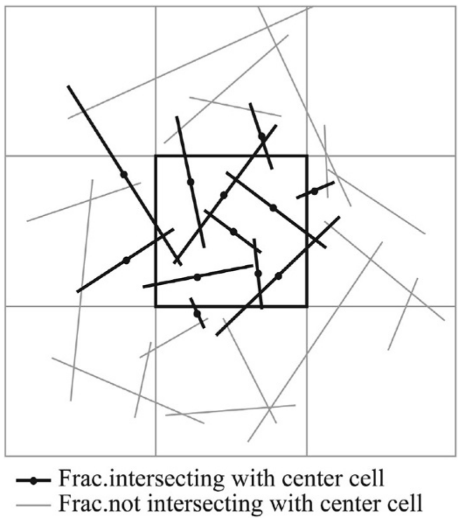
Frac.intersecting with center cell
Frac.not intersecting with center cell

Fig. 2 Sketch of fracture assignment in two dimensions

It is clear that some redundant analyses are performed, which means that an identical pair of fractures is analyzed in more than one cell. For example, the fractures O_1 and O_2 shown in Fig. 1 will be tested against intersection twice: once in cell A and once in cell B. Moreover, a pair of large fractures can introduce additional redundant analyses because they may simultaneously intersect many cells.

The first algorithm presented here is based on the concept that, under the requirement for achieving a full coverage and investigation of all possible PIFs, a certain number of redundant analyses is expected and acceptable because the extra computations can be averaged and accommodated by parallel computing.

However, if two fractures have been identified to be a PIF after performing an analysis in one cell, the record of this PIF should be unique throughout all cells, meaning that no duplication of the same record is allowed in other cells. This requires a criterion upon which each cell can decide to accept (record) or reject (ignore) a certain PIF that it has found. A straightforward approach is selecting a point P on every PIF that spatially associates it with a cell, establishing a 'one-to-one' relation. Figure 1 shows two obvious options for P: the center of either of the two fractures (O_1 or O_2) or the mid-point of a PIF's intersected line (P_{12}). The former two can be viewed as 'the priors' because they are known parameters, whereas the latter is a 'posterior' property requiring calculation. However, the point P_{12} is almost ready to use because its coordinate is merely the average of both ends of the intersected line that have been calculated in the process of precise intersection analysis. Moreover, compared to the edge effect of the fracture centers caused by fracture radii inside the boundary cells, the mid-points can distribute more uniformly under evenly spaced partitions, yielding similar numbers of PIFs among the different cells. This can benefit the loading balance of subsequent simulation procedures. Thus, for a certain cell, a PIF is recorded only if its mid-point of the intersected line is inside this cell.

Assume that there are a total number of N_f factures and N_p cells. Each cell is analyzed on a separate processor in parallel with other cells. The procedure for the k-th processor can be presented as follows:

(1) Assemble a subset from the fracture pool where each fracture intersects with cell k;
(2) Enumerate a pair of factures from the set; and.
(3) Perform an intersection analysis between the two fractures:
(3–1) If they intersect, calculate the position of the mid-point of their intersected line; otherwise, go to step 4;
(3–2) if the mid-point is inside cell k, record this PIF in cell k; otherwise, simply ignore this PIF (another cell will record it), and go to step 4.
(4) If there are pairs that have not been analyzed, return to step 2; otherwise, go to step 5;
(5) Terminate the intersection analysis procedure on cell k.

4 An optimized approach with reduced redundancy

The advantage of the first algorithm is that it is a completely independent parallel algorithm that requires no inter-process communication. However, its shortcoming is also obvious, as it lacks a method to reduce the redundant analysis among different cells, which may counterbalance some of the performance gains of parallel computing. Thus, an optimization is necessary to improve this technique, and this can begin with a further analysis of the sources of redundancy.

Figure 3 shows four possible states of a single fracture related to cell A:

(1) State 1: facture O_1 is fully contained in cell A, called a ‘contained fracture’ (CF);
(2) State 2: facture O_1 is separated from cell A, called a ‘disjoint fracture’ (DF);
(3) State 3: facture O_1 intersects cell A, with its center inside this cell, called an ‘inside-center boundary fracture’ (ICBF);
(4) State 4: facture O_1 intersects cell A, with its center outside this cell, called an ‘outside-center boundary fracture’ (OCBF).

For a certain cell, it is assigned to the fractures in states 1, 3 and 4. Thus, six types can be used when selecting a

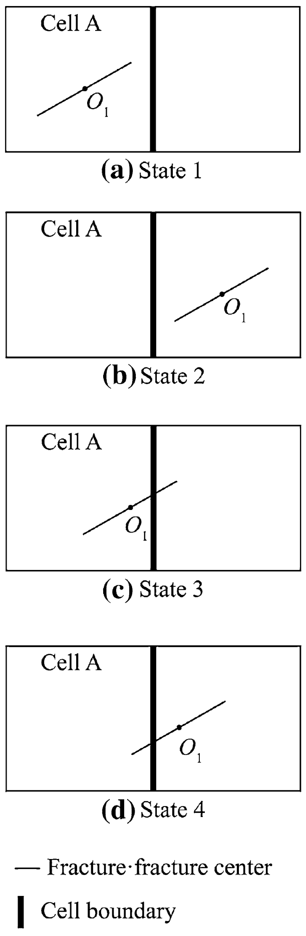

Fig. 3 Sketch of the possible states of a fracture related to a cell in two dimensions

pair of fractures: (CF, CF), (CF, ICBF), (CF, OCBF), (ICBF, ICBF), (ICBF, OCBF), and (OCBF, OCBF). The first three types of pairs can only appear in one cell. In other words, the two fractures appearing in cell A can never simultaneously appear in cell B or other cells because one of the fractures is fully contained by cell A. Thus, no redundant analysis occurs for these types of pairs.

However, the last three types, which are pairs of boundary fractures, may be involved in various types of redundancy:

(1) For a pair of fractures that are both ICBFs in cell A, if they simultaneously intersect another cell (or cells), say cell B, the two fractures must both be OCBFs in cell B.
(2) Similar to (1), for a pair of fractures where one is an ICBF and the other is an OCBF in cell A, for cell B, the types of the two fractures could be either (ICBF, OCBF) or (OCBF, OCBF). If they simultaneously intersect more than one cell, the possible type could be one (ICBF, OCBF) with multiple (OCBF, OCBF) or only multiple (OCBF, OCBF).

Table 1 Possible redundancy caused by a pair of boundary fractures ($n \geq 1$)

Cell A	Cell B (s)	Redundancy
(ICBF, ICBF)	Absent	0
	(OCBF, OCBF) × n	n
(ICBF, OCBF)	Absent	0
	(ICBF, OCBF) × 1	1
	(OCBF, OCBF) × n	n
	(ICBF, OCBF) × 1 + (OCBF, OCBF) × n	$1 + n$
(OCBF, OCBF)	Absent	0
	(OCBF, OCBF) × n	n
	(ICBF, ICBF) × 1	1
	(ICBF, ICBF) × 1 + (OCBF, OCBF) × n	$1 + n$
	(ICBF, OCBF) × 1	1
	(ICBF, OCBF) × 1 + (OCBF, OCBF) × n	$1 + n$
	(ICBF, OCBF) × 2	2
	(ICBF, OCBF) × 2 + (OCBF, OCBF) × n	$2 + n$

(3) For a pair of fractures that is in the form of (OCBF, OCBF) in cell A, one possible type for cell B is one or multiple (OCBF, OCBF). Other possible types can be obtained through reverse deduction of types (1) and (2).

Table 1 lists all possible redundancies in the presence of a pair of boundary fractures. Based on the possible states listed in Table 1, the number of redundant analyses on the same pairs of fractures in different cells can be reduced to some degree. This is accomplished by 'forecasting', e.g., if cell A has a pair of (ICBF, ICBF), the same pair can only appear in cell B (or in more cells) in the form of (OCBF, OCBF). To avoid a potential redundancy, one of the two cells should postpone its analysis. Table 1 indicates that the (OCBF, OCBF) pair type may be involved in more redundancies than that of the other two types. Thus, it is practical to require all cells to postpone their analysis on all pairs of (OCBF, OCBF) that they own while preferentially performing analyses of the (ICBF, ICBF) and (ICBF, OCBF) types. A special case is when a (ICBF, OCBF) pair in cell A may appear in the same type in cell B, introducing another type of redundancy. A workaround can be performed by comparing the global ordinal numbers (id) of the two fractures: if the ICBF-type fracture has a lower id than that of the other (OCBF), an analysis is performed; otherwise, it is simply postponed because another cell may meet this condition and thus perform the analysis. Note that all the postponed analyses will likely result in an underestimation of the number of actual PIFs. Thus, a double-check phase is required to re-examine all the postponed pairs. Provided that some of these postponed pairs have not been analyzed by other cells, a supplementary analysis is then performed.

The above procedure can reduce all types of redundancies except for some pairs in the form of (OCBF, OCBF). For a pair of large fractures that are crossing through the entire region of study and simultaneously intersecting many cells in the form of (OCBF, OCBF), it is possible that no analysis has been performed on them because they do not simultaneously intersect with a cell where any of their centers are located (being absent in the form of (ICBF, ICBF) or (ICBF, OCBF) in other cells). Moreover, these postponed (OCBF, OCBF) pairs will be discovered in the double-check phase. Thus, redundant supplementary analyses are concurrently performed in many cells. To avoid duplicated records of this pair if it is a PIF, only the cell that contains the mid-point of their intersected line records this PIF, whereas other cells simply ignore it.

Note that the double-check phase requires inner-communications among all the cells (processors) such that each cell tells other cells which pairs it has analyzed while gathering information on the work that others have performed. When every cell has determined the analyzed pairs of the entire fracture pool, each cell can decide whether a postponed pair it owns requires a supplementary analysis.

The complete workflow of this parallel algorithm is presented as follows, where steps 1–5 is the 'forecasting' phase, steps 6–8 is the 'postponing analysis' phase and the remainder (steps 9–15) is the 'double-check' procedure:

(1) Assemble a fracture set $\{F\}_k$ of which all fractures intersect cell k;
(2) Divide $\{F\}_k$ into $\{CF\}_k$, $\{ICBF\}_k$ and $\{OCBF\}_k$;
(3) Perform analyses on all pairs enumerated from $\{CF\}_k$, and record all found PIFs;
(4) Perform analyses on all pairs, of which one fracture is from $\{CF\}_k$ and the other is from $\{ICBF\}_k \cup \{OCBF\}_k$; and record all found PIFs;
(5) Perform analyses on all pairs enumerated from $\{ICBF\}_k$, save each pair's name to List A, and record all found PIFs;
(6) Enumerate a pair where one fracture is from $\{ICBF\}_k$ and the other is from $\{OCBF\}_k$; the two fractures are denoted as (f_i, f_j), where $i < j$ and where i and j are their global ordinal numbers, respectively;
(7) If f_i is a type of ICBF, save this pair name to List B1, and then perform an analysis and record all found PIFs; otherwise, save this pair name to List B2;
(8) Return to (6) if there are other pairs that have not been enumerated; otherwise, proceed with remaining steps;

Table 2 Properties of the synthesized fracture network

Group	Intensity (m^{-3})	Radius (Gaussian)		Orientation (Fisher)		
		Mean (m)	STD (m)	Strike (°)	Dip (°)	Coef. (K)
1	0.5	4	2	0	45	20
2	0.5	2	1	90	45	20
3	0.5	1	0.5	180	45	20

(9) Communications to other cells:
(9–1) Send List B1 to other cells;
(9–2) Receive all foreign List B1 sent from other cells;
(9–3) Merge these foreign List B1 into List B1';
(10) Enumerate each pair from List B2. If this pair does not exist in List B1', then save this pair name to List B3 and perform an analysis and record all found PIFs; otherwise, ignore this pair because it has been analyzed by another cell;
(11) Communications to other cells:
(11–1) Send List A and List B3 to other cells;
(11–2) Receive all foreign List A and List B3 sent from other cells;
(11–3) Merge all foreign List A into List A'; merge all foreign List B3 into List B3';
(12) Enumerate a pair from$\{\text{OCBF}\}_k$;
(13) If this pair does not exist in either List A', List B1' or List B3', perform an analysis on this pair:
(13–1) If the two fractures of this pair intersect and the mid-point of their intersected line is inside cell k, then record this PIF; otherwise, ignore it because it should be recorded in another cell that contains this mid-point;
(13–2) If the two fractures are separated, simply ignore this pair;
(14) Return to step (12) if there are other pairs from $\{\text{OCBF}\}_k$ that have not been enumerated; otherwise, proceed with the remaining step;
(15) Terminate the intersection analysis procedure for cell *k*.

5 Numerical study

All the testing codes are written in the Python and C++ languages, and the Massage Passing Interface (MPI) is used to provide a parallel programming environment. The MPI is a flexible framework that supports a wide scalability and various types of parallel computing, e.g., single processor with multiple cores, multiple-processor clusters and hybrid models (Gropp et al. 1999). Moreover, the stepwise intersection analysis approach is implemented in both of the two algorithms because it can reduce the number of precise analyses, which are computationally expensive.

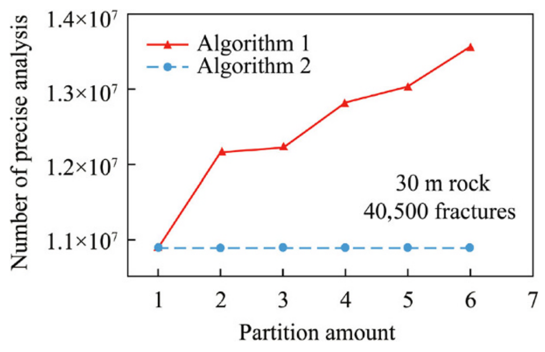

Fig. 4 Changes of the required number of precise intersection analyses using different numbers of partitions

Table 3 Configuration of the testing platform

CPU	Memory	OS	Software
AMD	4 GB*2	Windows 7	Python 2.6
A8-5600 k		64-bit	VC2010
4 Cores			MPICH2-1.4.1p1
3.60 GHz			mpi4py-1.3.1

The data for this test are several stochastic fracture networks synthesized from three groups of fractures. Table 2 shows the main properties.

The first test demonstrates the differences in the total number of computations resulting from the redundant analyses between the two algorithms. We use a realization of a 30 m^3 cubic rock that contains 40,500 fractures for this test. Figure 4 shows the changes of the total numbers of required precise intersection analyses for different numbers of partitions.

Figure 4 indicates that the number of calculations for algorithm 1 increases as more cells are involved, whereas algorithm 2 shows no evident fluctuation. This is because a denser cell partitioning introduces additional boundary fractures, yielding a substantially greater number of redundant analyses in algorithm 1. However, algorithm 2 can maintain a stable number of calculations because it is equipped with a procedure to reduce this type of redundancy to the greatest extent possible.

The second test compares the actual calculation times for the two parallel algorithms. Three rock sizes (10, 30, and 50 m^3) are used, yielding different numbers of fractures of 1,500; 40,500; and 187,500, respectively. Five realizations of stochastic fracture networks are generated for each rock size, and each realization undergoes ten independent micro-runs, where the averaged calculation time is used for comparison. The detailed configuration of the testing platform is presented in Table 3.

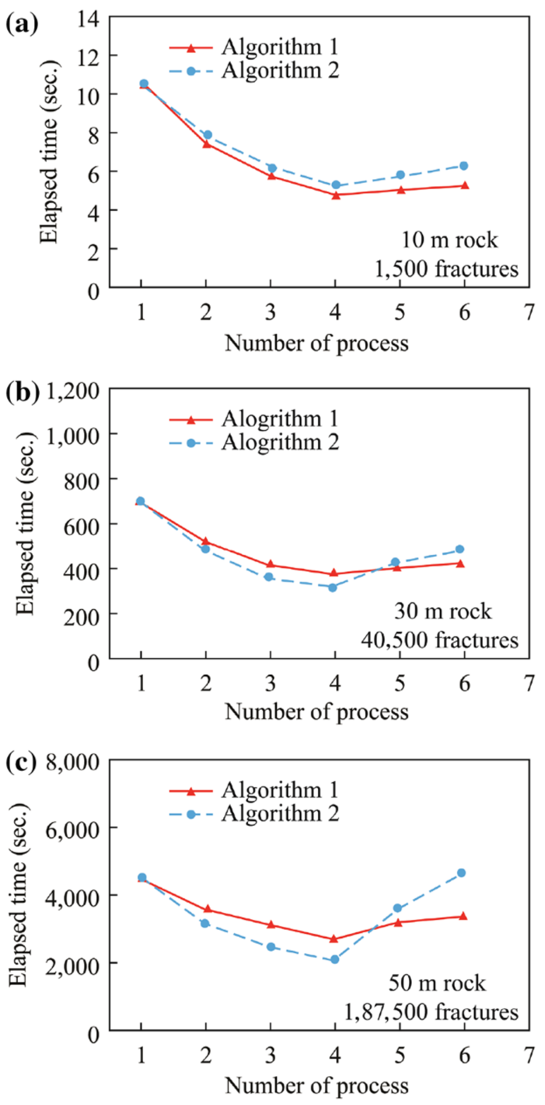

Fig. 5 Comparison of calculation times for the two algorithms for (a) 10 m, (b) 30 m and (c) 50 m rocks

Table 4 Averaged speedup ratios

Number of processes	Algorithm ID	
	1	2
1	1.0	1.0
2	1.3	1.4
3	1.6	1.8
4	1.9	2.2
5	1.7	1.6
6	1.7	1.4

Figure 5 shows the results for each rock size, and the averaged speedup ratios among all rock sizes are summarized in Table 4. A common tendency shown in Fig. 5 is that the calculation times of both algorithms decrease as more processes are launched, but a rebound occurs when more than four processes are used. This is due to the hardware limitations of the processor (four cores): although many processes can be launched, only four of them execute in parallel, while the others are running in serial mode, resulting in a degradation of the performance. Another tendency is that, within the four processes, algorithm 1 outperforms algorithm 2 in the low fracture amount test (10 m^3 rock), while the advantages of algorithm 2 become significant when the fracture amount increases (30 m^3 and 50 m^3 rocks). In contrast, when the number of processes is beyond the capacity of the processor, algorithm 2 shows a more severe performance loss than that of algorithm 1. Table 4 also shows this tendency in the measurement of the averaged speedup ratios.

This is because when there is a small amount of fracture data, the influence of the redundant analyses is completely covered by routine procedures, e.g., program initialization and I/O operations. Thus, algorithm 1 demonstrates its advantage in performance because it is quite simple in regard to computation logic. When more fractures are involved, the burden of the redundant analyses gradually dominates the total calculation time. Thus, algorithm 2 begins to outperform algorithm 1 because it can eliminate much of the redundancy. However, the performance of algorithm 2 is more sensitive to the limitations of hardware capacity compared to algorithm 1. This is because algorithm 1 is a completely independent parallel procedure, whereas algorithm 2 relies on inner-process communications (double-check phase). When excessive processes are launched, some of the processor cores are running processes in series mode, which delays the collective communication among all processes and thus reduces the overall performance.

Therefore, in the face of different numbers of fractures and parallel computing environments, a trade-off is necessary when choosing between a simple algorithm at the cost of redundant computations (algorithm 1) and another algorithm that can avoid these redundancies but utilizes a more complicated procedure (algorithm 2). Generally speaking, if the number of fractures is large (more than ten thousand) and a professional parallel computing environment where sufficient processors (or cores) and unobstructed inter-process communication channels are available, algorithm 2 is the best choice. Otherwise, algorithm 1 is the practical solution.

6 Conclusions

In this paper, two parallel algorithms are proposed to improve the computational performance of the intersection analysis procedure. They are designed considering the computational efficiency of the algorithms and a seamless interfacing between other procedures. The first algorithm is logically simple but contains redundant computations, whereas the second is an optimized version with reduced redundancy. The numerical study demonstrates that significant performance gains can be obtained in both of the algorithms in large-scale simulations. A further analysis indicates that the second algorithm is the best choice in a favorable parallel computing environment, but the first algorithm is still a practical approach because it is insensitive to hardware limitations and is easy to implement. Future studies will include tests of the scalability on a parallel cluster system, accommodating additional fracture geometric properties and applying the algorithm to real-world problems.

Acknowledgments The research was supported by the National Basic Research Program of China (973 Program) (2010CB428801, 2010CB428804); National High-tech R&D Program of China (863 Program) (2011AA050105); National Science Foundation of China (40972166); National Science and Technology Major Project of China (2011ZX 05060-005).

References

Baecher GB, Lanney NA, Einstein HH (1977) Statistical description of rock properties and sampling. In: Proceedings of the 18th U.S. symposium on rock mechanics. CO, 1–8

Einstein H, Locsin J (2012) Modeling rock fracture intersections and application to the Boston area. J Geotech Geoenviron Eng 138(11):1415–1421

Fadakar AY, Xu C, Dowd PA (2011) A general framework for fracture intersection analysis: algorithms and practical applications. In: Proceedings of the 4th Australian geothermal energy conference. Melbourne, 15–20

Gilmour HMP, Billaux D, Long JCS (1986) Models for calculating fluid flow in randomly generated three-dimensional networks of disc-shaped fractures: theory and design of FMG3D, DISCEL, and DIMES. Lawrence Berkeley Laboratory, CA

Gropp W, Lusk E, Skjellum A (1999) Using MPI: portable parallel programming with the message-passing interface. MIT Press, MA

Li XQ, Yang SQ, Wang XG (2007) Generation and visualization technologies of three-dimensional network of rockmass stochastic structural plane. Chin J Rock Mech Eng 26(12):2564–2569

Li ZY, Wang MY, Zhao JH, Wang HF (2014) Optimizing fracture intersection analysis procedure in 3D fracture network seepage simulation. J China Coal Soc. doi: 10.13225/j.cnki.jccs.2013.1385

Liu HM, Wang MY, Song XF (2011) A new approach for effectively determining fracture network connections in fractured rocks using R tree indexing. J Coal Sci Eng (China) 17(4):401–407

Liu HM, Wang MY, Song XF (2013) Stepwise approach for identifying pairwise fracture intersections in 3D fracture networks. J Grad Univ Chin Acad Sci 30(1):24–32

Song X, Xu W (2004) Numerical model of three-dimensional discrete fracture network for seepage in fractured rocks (I): generation of fracture network. Chin J Rock Mech Eng 23(12):2015–2020

Yu QC, Xue GF, Chen DJ (2006) Theory on general block method of fractured rock mass. China Waterpower Press, Beijing