Computers and Geotechnics 89 (2017) 168–178

Contents lists available at ScienceDirect

Computers and Geotechnics

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

Research Paper

Parallel and scalable block system generation

Michael Gardner [a,*], John Kolb [b], Nicholas Sitar [a]

[a] University of California, Department of Civil and Environmental Engineering, Berkeley, CA 94720, United States
[b] University of California, Computer Science Division, Berkeley, CA 94720, United States

ARTICLE INFO

Article history:
Received 27 October 2016
Received in revised form 22 March 2017
Accepted 1 May 2017

Keywords:
Block generation
Fractured rock mass
Parallel computing
Cloud Computing
Open-source software
Linear programming
Apache Spark

ABSTRACT

Generating a realistic representation of a fractured rock mass is a first step in many different analyses. Field observations need to be translated into a 3-D model that will serve as the input for these analyses. The block systems can contain hundreds of thousands to millions of blocks of varying sizes and shapes; generating these large models is very computationally expensive and requires significant computing resources.

By taking advantage of the advances made in big data analytics and Cloud Computing, we have a developed an open-source program—`SparkRocks`—that generates block systems in parallel. The application runs on Apache Spark which enables it to run locally, on a compute cluster or the Cloud. The block generation is based on a subdivision and linear programming optimization as introduced by Boon et al. (2015). `SparkRocks` automatically maintains load balance among parallel processes and can be scaled up on the Cloud without having to make any changes to the underlying implementation, enabling it to generate real-world scale block systems containing millions of blocks in minutes.

1. Introduction

Generating a realistic three dimensional model of a fractured rock mass is the first step in many analyses. Discrete block based methods such as discontinuous deformation analysis (DDA) [1] and the distinct element method (DEM) [2] require a full geometric description of the particles in their initial configuration as a starting point for the computations. Identifying removable blocks in a larger rock mass also requires a complete representation of the orientation of the blocks and discontinuities within the rock mass—as blocks are removed, the stability of newly exposed blocks must also be considered. Similarly, analysis of seepage through fractured rock relies on a complete description of the fractures and how they are connected within the rock.

Fundamentally, this is not a new topic and many researchers have developed algorithms to address this problem. Warburton [3,4] presents a methodology for generating a blocky rock mass based on sequential introduction of discontinuities and stores the generated blocks using a three-level data structure (vertices, edges and faces). Heliot [5] proposes a scheme for generating a blocky rock mass that additionally deals with non-convex blocks by representing a rock block as an assemblage of convex blocks. In Heliot's approach, the blocks are stored using a two-level data structure (vertices and faces). Ikegawa and Hudson [6] developed the so-called directed body concept in which all the discontinuities are introduced simultaneously. The blocky mass is then systematically extracted from the vertices, edges and faces. Additionally, several researchers [7,8] have developed algorithms based on principles from combinational topology. These techniques are able to deal with complex geometry, but require a significant amount of "bookkeeping" when implemented. Recently, Boon et al. [9] presented a block cutting algorithm that is based entirely on linear programming. Instead of explicitly calculating the vertices where the discontinuities intersect, the problem is cast as a linear programming optimization. This makes it possible to represent the rock blocks by a single-level data structure since only information about the faces is needed. The simplicity and the efficiency of the method makes it an attractive candidate for large-scale computations. The algorithm itself is entirely decoupled—once a block has been subdivided into two new blocks, the further subdivision of these new blocks can proceed independently. Consequently, this algorithm is naturally parallel and multiple cuts can be made simultaneously without the need to share information among processes.

2. Apache Spark

The subdivision of the rock mass is an iterative process on the same set of data, making the parallel, open-source framework

* Corresponding author.
E-mail address: mhgardner@berkeley.edu (M. Gardner).

http://dx.doi.org/10.1016/j.compgeo.2017.05.001
0266-352X/

Apache Spark [10] an ideal platform for a blocky rock mass generator using a subdivision-type approach. Spark can run on any platform ranging from laptops and personal workstations with multicore processors to Cloud based computing platforms, such as Amazon Elastic Cloud Compute (EC2). This scalability in the power of the computing environment without having to make any changes to the application code allows for the analysis of extremely large problems requiring large amounts of memory and computing power.

The fundamental abstraction in Spark is Resilient Distributed Datasets (RDDs) [10] that allow it to keep large data sets in memory and perform computations in a fault tolerant manner. By keeping the dataset in memory, Spark is able to do iterative transformations on the data extremely quickly since it avoids writing to disk. Fault tolerance is achieved by tracking the lineage of RDDs—all operations applied to the RDD are represented through a lineage graph. When a new operation is applied to the RDD, a new link is added to the graph. Additionally, RDDs are evaluated "lazily": only when a result is requested does Spark execute the transformations described by the lineage graph to actually materialize the current RDD. In this way, if a process unexpectedly fails the current state can be quickly reconstructed from the lineage contained in the graph.

For large scale problems, Spark clusters can be deployed on EC2 using Amazon's Elastic MapReduce (EMR) framework, which automatically allocates and configures a cluster of EC2 instances to execute Spark tasks submitted by the user. Amazon EC2 falls under the greater umbrella of Cloud Computing—applications delivered as services over the Internet and the associated software and hardware that provide those services [11]. Running large scale computations on the Cloud offers users several advantages. First, resources can be scaled on demand to meet the computing requirements of the problem at hand. Second, it is no longer necessary to invest large amounts of capital in computational hardware and the associated management and maintenance. Lastly, usage can be scaled up or down as needed so users only pay for what they use and only use what they need. This makes it possible for anyone to run large scale computations since it is no longer necessary to physically own a computer cluster. Hence, Cloud Computing essentially opens the door to High Performance Computing (HPC) for anyone willing to step through it.

2.1. SparkRocks

By taking advantage of Spark's ability to run on any computer system and the scalability of Cloud Computing, we developed a parallel block cutting program, `SparkRocks`[1], that is capable of generating large numbers of blocks very quickly. The code is open source and the necessary inputs to generate a fractured rock mass are based on parameters that are obtained from field observations, allowing users to quickly translate field measurements into a three-dimensional model.

The program was tested on different systems—a laptop, desktop workstation, and Amazon EC2—to illustrate its ability run on different platforms and to verify its scalability. Results show that we can generate approximately 8 million blocks in roughly 9 minutes.

3. Block cutting algorithm

The block cutting algorithm uses a sequential subdivision approach based on linear programming optimization introduced by Boon et al. [9]. Each discontinuity is introduced individually and checked for intersection. If it intersects the block, two new blocks are generated. The process continues sequentially until all discontinuities have been introduced, yielding a representation of the fractured rock mass. Many block cutting algorithms require a significant amount of bookkeeping in terms of vertices, edges, faces and how all of these elements are connected. From an implementation perspective, this can be extremely tedious and may not be as robust in terms of floating point error. The linear programming optimization approach introduced by Boon et al. greatly simplifies how block cutting is implemented and how each block is represented in terms of data structure. We give only a brief overview of this rock cutting algorithm since the details are presented in [9].

The orientations of joints in a fractured rock mass are described by strike and dip, as shown in Fig. 1. The block cutting algorithm uses the normal vector of the plane containing the joint and the distance of that plane to some origin. The strike and dip define the normal vector of the joint. The distance of the joint plane from the origin is determined by projecting a vector connecting the origin to a point in the joint plane onto the normal vector. The global $+x$, $+y$ and $+z$ axes are oriented North, West and upward.

Using only the joint normal and distance, it is possible to completely describe a polyhedral block, as shown in Fig. 2. A block bounded by N planes is described by the following equation:

$$a_i x + b_i y + c_i z \leqslant d_i, \quad i = 1, \ldots, N \tag{1}$$

The coefficients (a_i, b_i, c_i) represent the normal vector to the ith plane bounding the block and d_i is the distance of that plane from some local origin. In vector notation this becomes:

$$\boldsymbol{a}_i^T \boldsymbol{x} - d_i \leqslant 0, \quad i = 1, \ldots, N \tag{2}$$

In order to subdivide a block, it is necessary to establish whether the block is intersected by the discontinuity being considered. The novelty in the algorithm presented in [9] is recasting this problem as a linear program:

$$\begin{aligned} &\text{minimize} \quad s \\ &\boldsymbol{a}_i^T \boldsymbol{x} - d_i \leqslant s, \quad i = 1, \ldots, N \\ &\boldsymbol{a}_{\text{new}}^T \boldsymbol{x} - d_{\text{new}} = 0 \end{aligned} \tag{3}$$

Here N represents the number of planes that define the block and the discontinuity being considered is represented by the equality. If $s < 0$, there is an intersection and the parent block is split into two child blocks. The child blocks inherit all the parent block's planes as well as the intersecting discontinuity with opposite signs for the discontinuity normal vector for each child block.

As the subdivision continues, some of the discontinuities may become geometrically redundant. It is not necessary to remove these redundancies after each intersection check; instead they can be removed at a later time as discussed in Section 4.2.3. Again, this can be done by solving a linear program:

$$\begin{aligned} &\text{maximize} \quad \boldsymbol{c}^T \boldsymbol{x} \\ &\boldsymbol{a}_i^T \boldsymbol{x} \leqslant d_i, \quad i = 1, \ldots, N \end{aligned} \tag{4}$$

Here, $\boldsymbol{c}$ is the normal vector specific to the discontinuity being checked for redundancy and with associated distance d. If $| \boldsymbol{c}^T \boldsymbol{x} - d | < \varepsilon$ the discontinuity is not redundant, where ε represents a numerical tolerance close to zero.

Additionally, we take advantage of two major optimizations to the block cutting process that are presented in [9]. The first optimization draws on an idea common to contact detection in particle methods (for example, see [12]): the complex geometry of the polyhedral blocks is enclosed in a simpler shape, in this case a bounding sphere. This enables a simple and fast check for intersection to determine if a more thorough but computationally expensive check is necessary. The second optimization is to control the size and aspect ratio of the blocks that are generated during the

[1] Available at https://github.com/cb-geo/spark-rocks.