Computers & Geosciences 36 (2010) 953–958

Contents lists available at ScienceDirect

Computers & Geosciences

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

A general parallelization strategy for random path based geostatistical simulation methods

Grégoire Mariethoz*

Centre for Hydrogeology, University of Neuchâtel, 11 Rue Emile Argand, CP 158, CH-2000 Neuchâtel, Switzerland

ARTICLE INFO

Article history:
Received 16 July 2009
Received in revised form
17 November 2009
Accepted 21 November 2009

Keywords:
Geostatistics
Simulation
Parallelization
Sequential simulation
Multiple-point
Multiple-points
MPI
Speed-up
Parallel computing
Random path

ABSTRACT

The size of simulation grids used for numerical models has increased by many orders of magnitude in the past years, and this trend is likely to continue. Efficient pixel-based geostatistical simulation algorithms have been developed, but for very large grids and complex spatial models, the computational burden remains heavy. As cluster computers become widely available, using parallel strategies is a natural step for increasing the usable grid size and the complexity of the models. These strategies must profit from of the possibilities offered by machines with a large number of processors. On such machines, the bottleneck is often the communication time between processors. We present a strategy distributing grid nodes among all available processors while minimizing communication and latency times. It consists in centralizing the simulation on a master processor that calls other slave processors as if they were functions simulating one node every time. The key is to decouple the sending and the receiving operations to avoid synchronization. Centralization allows having a conflict management system ensuring that nodes being simulated simultaneously do not interfere in terms of neighborhood. The strategy is computationally efficient and is versatile enough to be applicable to all random path based simulation methods.

1. Introduction

The size of the simulation grids used for geological models (and more generally for spatial statistics) has increased by many orders of magnitude in the last years. This trend is likely to continue because the only way of modeling different scales together is to use high-resolution models. This is of utmost importance in applications such as hydrogeology, petroleum and mining, due to the critical influence of small scale heterogeneity on large scale processes (e.g. Mariethoz et al., 2009a).

Efficient pixel-based geostatistical simulation algorithms have been developed, but for very large grids and complex spatial models, the computational burden remains heavy. Furthermore, with increasingly sophisticated simulation techniques including complex spatial constraints, the computational cost for simulating one grid node has also raised. As multicore processors and clusters of computers become more and more available, using parallel strategies is necessary for increasing the usable grid size and hence allowing for models of higher complexity.

Parallel computers can be divided into two main categories: shared memory machines and distributed memory architectures. Shared memory machines have the advantage of ease and rapidity of the communications between the different computing units. Nevertheless, their price is extremely high and the total amount of memory as well as the total number of processors are limited. Therefore, most of the time it is distributed memory machines (or clusters computers) that are used in the industry or in the academic world. As such machines do not have a common shared memory space, the processors have to communicate by sending and receiving messages. The communication time between processors can be important and is often the bottleneck in a program execution.

In this paper, we propose a parallelization strategy applicable in the context of sequential simulation methods and based on the distribution of the grid nodes among all available processors. The method minimizes communication and latency times and can be applied using shared or distributed memory architectures, or a combination of both. It consists in centralizing the simulation on a master processor that calls other slave processors as if they were functions simulating one node each time. The key is to decouple the sending and the receiving operations to avoid waiting for synchronization. Centralization allows having a conflict management system making sure that nodes being simulated simultaneously do not interfere in terms of neighborhood.

The strategy is computationally efficient and is versatile enough to be applicable to all random path based simulation methods. It is illustrated with an example using the Direct Sampling approach (Mariethoz, 2009; Mariethoz and Renard,

* Tel.: +41 32 718 26 10.
E-mail addresses: gregoire.mariethoz@minds.ch, gregoire.mariethoz@unine.ch.

0098-3004/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cageo.2009.11.001

2010), which is a simulation algorithm using multiple-points (MP) statistics.

2. Parallelizing sequential simulations

Sequential simulation is a class of methods that is used to generate realizations of a random field (Deutsch and Journel, 1992; Caers, 2005; Remy et al., 2009). The general principle of the method is to discretize the random field on a grid and to draw successively (sequentially) for each node $\mathbf{x}$ of the grid an outcome of the random variable Z in a local cumulative conditional density function (ccdf). This local ccdf is conditional to the previously simulated nodes and to local data if those are available. Usually, a truncation is made and the ccdf is computed only from the values located in a neighborhood $N(\mathbf{x})$ of limited extension.

The local ccdf is determined using a spatial model, termed m, that describes the spatial structure of the random field. Grid nodes are often simulated in a random order, but alternative simulation paths can also be used (Pickard, 1980; Daly, 2004).

Depending on the sequential simulation technique used, m can be for example one or a set of variograms in the case of SGS/SIS simulations (Isaaks, 1984; Journel and Alabert, 1990), with possibly some auxiliary information (e.g. Mariethoz et al., 2009b), plus a lithotype rule for plurigaussian simulations (Le Loc'h et al., 1994; Armstrong et al., 2003), a training image or its associated data events catalogue for MP simulations (Strebelle, 2002; Zhang et al., 2006; Arpat and Caers, 2007; Straubhaar et al., 2008; Mariethoz and Renard, 2010), or a set of transition probabilities (Carle and Fogg, 1997).

Each sequential simulation method has its own way of computing the value $z(m,N(\mathbf{x}))$ that will sequentially be attributed to each node $\mathbf{x}$. Nodes are simulated in an order defined by a random path initialized at the beginning of the simulation. Once a value has been attributed to a node, this node becomes conditioning for the nodes that come later in the simulation process, i.e. it will be included in the ensemble $N(\mathbf{x})$ for the next nodes to simulate. This is the reason why these simulation techniques are termed sequential.

Parallelization of such simulations is possible at three levels. The realization level is the easiest to parallelize. It consists in having each realization of a Monte-Carlo analysis computed by a different processor. As every realization is, by definition, independent of the others, no communications between processors are needed. The maximum number of processors that can be used with this strategy is equal to the desired number of realizations. This strategy is widely used (e.g. Mariethoz et al., 2009a) and will not be discussed further in this paper.

Parallelizing a simulation at the path level means to divide the grid in zones and to attribute a different zone to each processor. By now, this strategy has been implemented by simulating groups of grid nodes at the same time (Dimitrakopoulos and Luo, 2004; Vargas et al., 2007).

The third level of parallelization is the node level. The simulation of each single node is parallelized. For example, the inversion of a large kriging matrix for SGS or the search for a data event in the multiple-points data events catalogue can be shared among many processors (Straubhaar et al., 2008). Speculative parallel computing can also be applied in the context of simulated annealing (Ortiz and Peredo, 2008). In all of these cases, the efficiency of the parallelization is limited when a large number of processors are available, because the size of the problem to solve for each individual processor becomes small compared to the communications time between processors.

These different strategies are not mutually exclusive. For example, the path can be distributed among different parallel machines, who themselves distribute the simulation of their individual nodes on local processors.

This paper focuses on parallelization strategies at the path level. The sequential character of the simulation process is a challenge for these strategies because of the dependence of the value of $z(\mathbf{x})$ with all previously simulated nodes. Another issue is that the time taken to simulate a node is not necessarily uniform, depending on the simulation method. Some nodes can be simulated much slower than others, which have for example a neighborhood less compatible with the spatial structure model m. In some cases, the number of neighbors can be different from one node to another, incurring variations in computational load. This problem becomes more acute when the simulation algorithm is run on heterogeneous architectures mixing processors of different performance.

Parallelizing the random path will inevitably lead to conflicts when a node has to be simulated by a processor while nodes of its neighborhood are being simulated by other processors. Moreover, certain algorithms, such as the Gibbs sampler (Geman and Geman, 1984) or the syn-processing (Mariethoz, 2009), require to re-simulate nodes that do not match certain conditions. This complicates the problem as it leads to changes in the simulation path, making it impossible to define in advance a conflicts-free path (a strategy adopted by Vargas et al., 2007). In certain cases, the simulation method can be adapted to be less sensitive to these conflicts (Dimitrakopoulos and Luo, 2004). But our goal is to find a general strategy that is applicable to all random path based simulation methods without generating conflicts.

3. Nodes distribution

The solution proposed in this paper is to have one processor, the master, managing the path, the search for neighbors and the conflicts, while all other processors, the slaves, devote their calculation power to the simulation itself. If n_{CPU} processors are available, the processor 0 is the master and processors 1 to n_{CPU}-1 are the slaves. The most obvious strategy would be to group n_{CPU}-1 nodes and distribute them among slave processors. Unfortunately, this strategy is not efficient with a large number of processors because it involves that all slaves must have returned their result to the master before the next group of nodes is sent to slaves for simulation. If one of the slaves uses more time than the others to simulate his node, all processors have to wait for it to finish. Moreover, the master does not perform any calculations while slaves are working, and the slaves also have to wait until the master has finished updating the simulation with the received group of nodes and has defined the neighborhoods for the next group.

Instead of groups, we propose that the master sends sequentially one node to each slave, and then waits for a result coming from any slave processor. Once this result is obtained, the master includes it in the simulated grid, finds the neighborhood of the next node and sends it to the same slave processor from which the result just came from. Then it waits again for a result coming from any slave processor. By avoiding synchronization, this strategy ensures that a minimum number of processors are waiting. The master practically does not wait when there are a lot of slaves. Moreover, while the master works on defining neighborhoods and attributing nodes to slave processors, all slaves except one are working. The slaves do not wait for each other as they perform their workload independently. The time devoted to communications is minimized because while the master sends or receives information from a slave, all other slaves carry on their work undisturbed.