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.

The procedure needs a starting routine that sends the first n_{CPU}-1 nodes on each slave CPU without receiving any result, and another ending routine that receives the last n_{CPU}-1 nodes from each slave CPU.

This way of centralizing the simulation on a master node has many implications. First, this strategy is impossible to apply on machines having a single processor. Second, it is not efficient if the total number of processors is not large enough. For example, using 2 processors is absurd as only one of them (the slave) is performing the simulation while the other one (the master) waits. But with an increasing number of processors, the communication time of each slave decreases, while it stays constant for the master, which can afford it because it practically does not perform any computation.

Another implication of the centralization is that it is easy to efficiently implement algorithms that come back on previously simulated nodes, such as for example the Gibbs sampler or the syn-processing. If changes must happen in the simulation path or in previously simulated nodes, the master node alone has to manage it and no additional communications have to take place.

A disturbing consequence of the strategy is that it does not allow reproducibility of the resulting simulations, although all of them will be consistent with the spatial law m. The random order in which nodes are sent to slaves and are returned in the simulation grid depends on the local characteristics and on the workload of each processor. Therefore, the order in which the nodes are simulated is continuously altered during the simulation process, which is equivalent to using a different random simulation path. As the simulated path is random anyway, this does not affect the faithfulness to the spatial model. But reproducibility of the simulations is compromised because the simulation path is not reproducible (it depends on complex hardware and software interactions that are difficult to control). Different runs of the same code yield different simulations, even if the random seed is identical.

It is also important to note that this strategy is inefficient when using an unilateral path (Daly, 2004) because the amount of conflicts generated would make it inefficient. It is also not directly applicable to techniques other than sequential simulations, such as Boolean simulation (Deutsch and Tran, 2002; Lantuéjoul, 2002) or turning bands (Matheron, 1973), for which other parallelization methods have been developed (Armstrong and Marciano, 1997; Kerry and Hawick, 1998; Ingam and Cornford, 2008).

4. Conflicts management

Let U_{sim} be the ensemble of all nodes currently simulated by all slave processors (ensemble containing n_{CPU}-1 elements). When a new node $\mathbf{x}$ has to be simulated, conflicts arise when $N(\mathbf{x}) \cap U_{sim} \neq \emptyset$, i.e. when at least one node currently simulated by any slave processor belongs to the neighborhood of node $\mathbf{x}$.

When a conflict arises, one can consider three ways of dealing with it. The first one is to ignore the conflict. This option is worth considering if the simulation grid is large and the number of processors reasonable, thus a limited number of conflicts will occur and it can be assumed that the resulting simulation will not be significantly biased. The second option is to try simulating another node $\mathbf{x}$ until there is no more conflict. Most of the time, the conflict will disappear after a certain number of tries, but there are cases where there is no suitable location in the entire simulation grid that generates no conflict. The conflict is then severe and only the third option remains: waiting until some of the nodes creating conflicts are simulated, and then try again.

The strategy adopted here combines the second and the third options. The idea is to try the second option n_{tries} times. After this maximum number of tries, one can consider that there is no conflict-free location in the entire simulation grid. At this point, the third option is used: waiting for slaves to return their simulated nodes.

In the context of node distribution, simply waiting for a returning node is not trivial. The procedure involves that for each returned node, another node has to be sent. The only way to receive a node is to send another node, even if it presents a conflict. The solution is to mark this node as "false", by setting the flag function $f(\mathbf{x})$ to 1 to signal that this node is causing a severe conflict, and send it to any slave processor. When a value for this "false" node is returned by the slave processor, it is discarded and simulated again with a different, conflict-free neighborhood. There are no more conflicts because in the meantime, the neighbors causing conflicts have been simulated. Fig. 1 schematically describes the entire strategy.

Note that nodes distribution and conflicts management involve that the random path defined at the initial stage of the simulation is continuously changed depending on neighborhood interactions and individual computational load of each processor. The path updating is done by the master.

5. Performance tests

The parallelization strategy described above has been tested on the Direct Sampling algorithm (Mariethoz, 2009; Mariethoz and Renard, 2010), a recent implementation of Multiple-points simulation (Guardiano and Srivastava, 1993; Strebelle, 2002; Hu and Chugunova, 2008). For each node $\mathbf{x}$ in the simulation grid, the algorithm scans a training image (TI) representing what the simulated field should look like independently of the data. As soon as a pattern matching the neighborhood of $\mathbf{x}$ in the simulation is found in the TI, the value of the central node of this pattern is assigned at location $\mathbf{x}$ of the simulation grid.

The performance tests consist in generating, with a varying number of processors, one unconditional realization on a relatively small grid ($100 \times 100 \times 5$ nodes). The neighborhoods are large (search radius size ranging from 30 nodes at the beginning of the simulation to 3 nodes at the end), thus likely to generate conflicts. With 14 million nodes, the training image is much larger than the simulation, and therefore scanning it for each simulated node represents a steep cost in terms of CPU time. The MPI libraries were used as a parallel interface. The machine available for the tests is a cluster of 56 AMD Opteron processors, grouped in 14 machines of 4 processors each, with InfiniBand interconnect.

Simulations were performed using a total number of processors ranging from 2 to 54. Two series of runs were performed. The first series addresses conflicts as described above (combination of the second and the third options for dealing with conflicts), with n_{tries} set to 1000. The second series ignores conflicts. Comparing both series allows evaluating the time spent for conflicts management.

Fig. 2 shows the decrease of CPU time as the number of slave processors becomes larger. The measure of CPU time is the user time of the slowest processor of each run (master and slaves), in seconds. Both series of runs are displayed. The series ignoring conflicts is expected to be faster than the one accounting for it, but such is not always the case. We explain this by fluctuations in the simulation time from one simulation to another. One should keep in mind that the simulations are not reproducible. Depending on the simulation path, more or less probable patterns may be generated, causing a different fraction of the TI to be scanned for each simulated node (this fraction has a large influence on the simulation time). It is an inherent characteristic

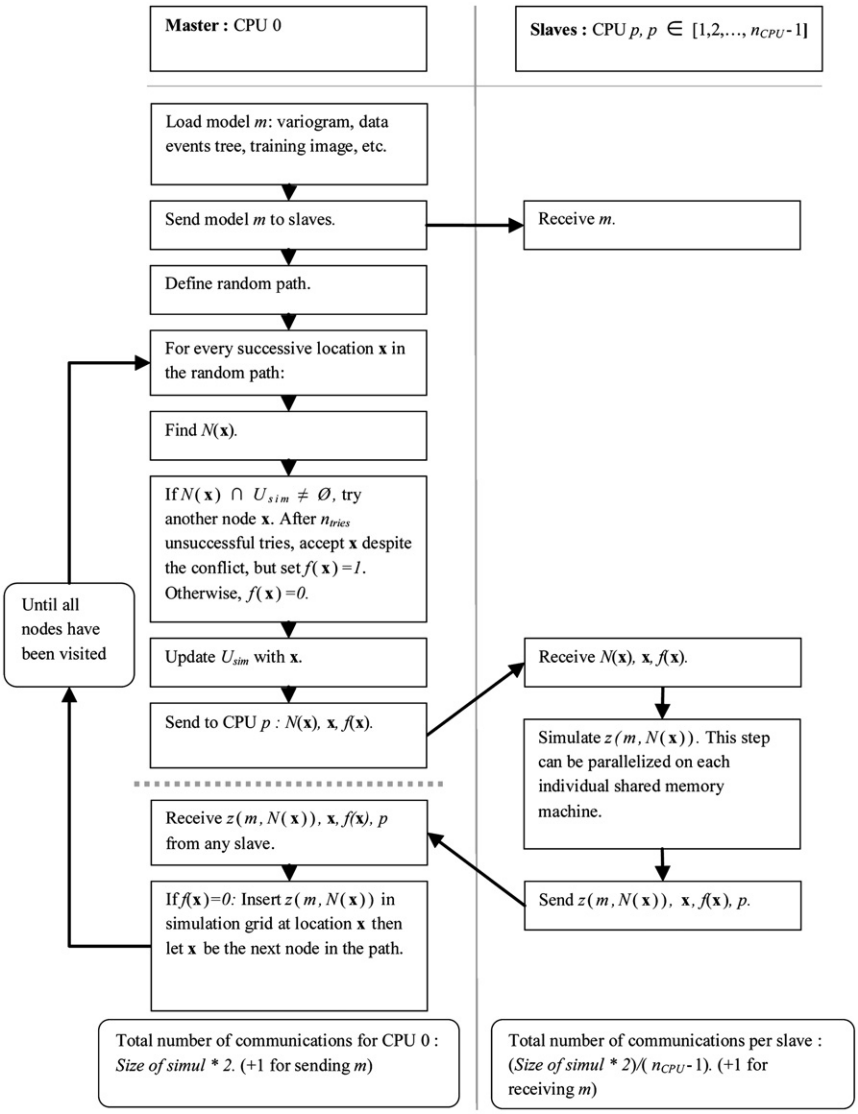

Fig. 1. Proposed parallelization strategy. Each column describes instructions executed by master and slaves. Variable are described in text. Dashed horizontal line: lapse when master waits for a result coming from any slave CPU.

of the parallelization strategy that the path is different for each realization. Therefore, computation times are subject to fluctuations that cannot be controlled.

A common performance test for parallel algorithms is to compute the speed-up function, which is the ratio between the time taken by the algorithm to perform some computations in a serial way and in parallel. It is defined as

$$sp(n_{slaves}) = \frac{t_{serial}}{t(n_{slaves})} = \frac{t_{serial}}{t(n_{CPU}-1)}, \quad (1)$$

where t is the CPU time needed to solve a given problem with a certain number of processors. In the best case, the speed-up should follow the ideal slope $sp(n_{slaves})=n_{slaves}$ (Amdahl, 1967). Fig. 3 shows the speed-up functions for both series of runs. In our case, the number of slave processors is considered instead of the total number of processors, because using a single processor is impossible. The reference time t_{serial} was obtained by running the serial version of the Direct Sampling code with the same parameters, on the same machine.

It is not straightforward to explain why the speed-up function (Fig. 3) exceeds the ideal curve when more than 16 processors are used. Moreover, the serial version is slower (7240 s) than using only one slave (6483 s), which is not logical because the overhead caused by the communications should degrade the performance. One possible reason could be related to the optimized use of the cache memory in the parallel version. Another reason could be that the reference time was computed using a different code (parallel and serial versions are fundamentally different, although it is the same simulation algorithm), which could induce a bias. Fine analysis of both codes may give clues on the cause for this speed-up increase. In any case, the speed-up in this example is good when the number of processors is large, which is a sign that the method is efficient.

For a parallel algorithm to be efficient, the workload must be balanced between all available processors. If large discrepancies are observed between the times of the different CPUs, it means that the workload is not evenly distributed Fig. 4 displays the load balancing of each processor in 9 of the performance tests with conflicts control. These curves show that there are no large CPU time differences between the processors of a same run. Moreover, when the number of processors becomes larger, fewer discrepancies appear. This result indicates that there are no major

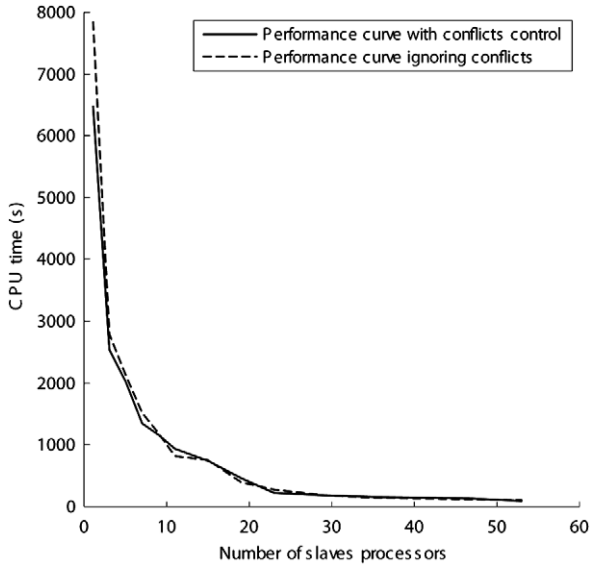

Fig. 2. Performance curve on a grid of 50,000 nodes ($100 \times 100 \times 5$), with and without conflicts control. Conflicts management consists in first trying n_{tries} times another location **x** to simulate. If no suitable location is found, wait for conflicts to disappear.

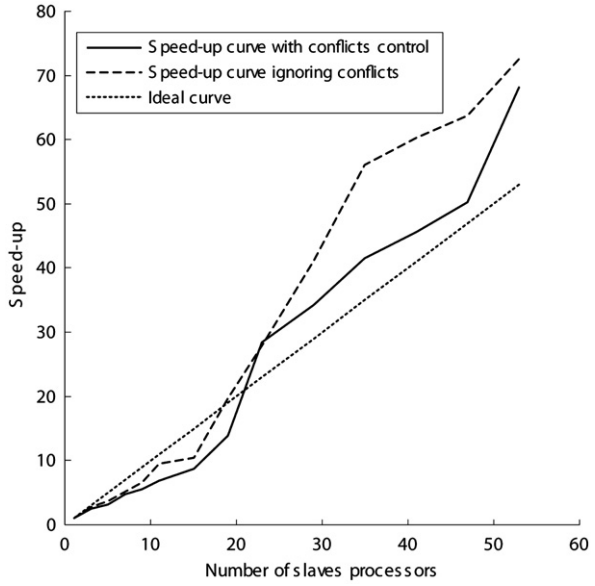

Fig. 3. Speed-up curves on a grid of 50,000 nodes ($100 \times 100 \times 5$), with and without conflicts control.

parallelization bottlenecks slowing down the entire simulation. Here again, one can observe that the CPU time is very well balanced when more than 16 processors are used.

In these performance tests, conflicts management does not significantly influence performance. Table 1 shows that even if the total number of tries (second option: the total number of times that the master node tried to solve a conflict by trying to simulate another node **x**) is very large, sometimes larger than the number of nodes in the simulation, it does not have a major influence on the global simulation time. This is because the computational cost of trying another node is small compared to the cost of simulating a node.

In case of severe conflicts (when no conflict-free location can be found in the entire simulation grid), the master has to send to a slave a node marked as "false", in order to wait other slaves return

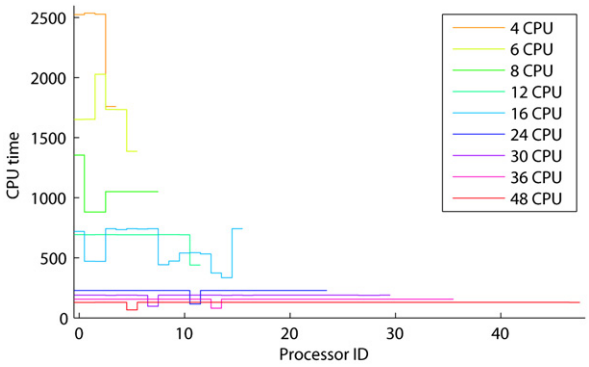

Fig. 4. Load balancing on a grid of 50,000 nodes, with conflicts control. In an ideal case, all processors would use identical CPU time.

Table 1
Numerical results of the performance tests with conflicts management. Max time is the CPU time of the slowest processor in the run. Total tries is the total number of times that another node was tried in case of conflicts (second option). Total wait is the total number of times that a "false" node had to be simulated (third option) because the second option failed. Logically, total wait is 1000 times smaller than total tries because n_{tries} was set to 1000.

n_{CPU}	**2**	**4**	**6**	**8**	**10**	**12**	**16**
Max time (s)	6483	2537	2028	1355	1158	934	743
Total tries	0	313	877	1491	54,055	2831	129,130
Total wait	0	0	0	1	54	2	129
n_{CPU}	**20**	**24**	**30**	**36**	**42**	**48**	**54**
Max time (s)	467	228	189	156	142	129	95
Total tries	31,032	44,045	86,088	147,268	251,421	746,746	272,272
Total wait	31	44	86	147	251	746	272

their result, causing the severe conflict to disappear. This way of solving severe conflicts (third option) is much more time-consuming than the second option, but fortunately it happens much less often (compare values of rows "total tries" and "total wait" in Table 1).

The computational burden associated to conflicts management only depends on the size of the simulation grid and on the number of processors. If the cost of simulating one node is small, the time devoted to conflicts management may be comparatively large. Performance is most affected by conflicts management in cases where the simulation grid is small, the number of processors is large and the cost per simulated node is low. Nevertheless, in these cases realizations should be quickly obtained with a serial algorithm, and parallelization is not needed.

If the number of processors is very large, the workload of the master node increases because of the extra cost of managing conflicts between all processors. Although we did not observe it during numerical tests with up to 54 processors, there should be a number of slaves where the master becomes the bottleneck because there are simply too many slaves to manage. This could affect performance on very large problems. A solution would be to split the work of the master in two levels by having not one but several master processors, and one super-master managing the masters. Following this idea, one could imagine systems with three or more levels of management.

6. Conclusion

Among the three simulation parallelization levels (realization, path and node), the path level allows using a large number of

processors without losing efficiency. It is applicable to all random path based geostatistical simulation methods, even if a small number of simulations have to be generated (contrarily to the realization level) In this paper, we propose a strategy that takes full profit of this parallelization level.

The overall performance of the strategy is good according to the speed-up criterion. This confirms that the communication and latency times are minimal and that most of the computational effort of all processors is devoted to the simulation and not to the parallelization overhead.

The issue of overlapping neighborhoods arising in the context of nodes distribution is addressed in a two-step process that tries to quickly solve the conflict by trying to find a conflict-free location to simulate, and in case of failure waits for the conflict to disappear. Conflicts management ensures that the quality of the realizations obtained with a parallelized simulation algorithm is the same as with the serial algorithm. In terms of performance, tests showed that conflicts management does not have a major influence.

The proposed parallelization strategy is flexible in many ways. It allows accelerating the stochastic image generation process independently of the simulation technique used. It can also accommodate different simulation techniques as well as various computer architectures such as grid computing or heterogeneous clusters.

The main drawback of the method is that it does not allow reproducibility of the resulting simulations.

Acknowledgments

We acknowledge the Swiss National Science Foundation (Grant PP002-1065557) and the Swiss Confederation's Innovation Promotion Agency (CTI Project 8836.1 PFES-ES) for funding this work. We also thank Philippe Renard and Julien Straubhaar for their constructive comments.

References

Amdahl, G., 1967. Validity of the single processor approach to achieving large-scale computing capabilities, Proceedings AFIPS Conference Proceedings. Thompson Books, Washington, DC, pp. 483–485.

Armstrong, M., Galli, A.G., Loc'h, G.L., Geoffroy, F., Eschard, R., 2003. Plurigaussian Simulations in Geosciences. Springer, Berlin 149 pp.

Armstrong, M., Marciano, R., 1997. Massively parallel strategies for local spatial interpolation. Computers & Geosciences 23 (8), 859–867.

Arpat, B., Caers, J., 2007. Conditional simulations with patterns. Mathematical Geology 39 (2), 177–203.

Caers, J., 2005. Petroleum Geostatistics, SPE Interdisciplinary Primer Series, Society of Petroleum Engineers, Richardson, TX, USA, 96 pp.

Carle, S.F., Fogg, G.E., 1997. Modeling spatial variability with one and multi-dimensional continuous Markov chains. Mathematical Geology 7 (29), 891–918.

Daly, C., 2004. Higher order models using entropy, Markov random fields and sequential simulation. In: Leuangthong, O., Deutsch, C.V. (Eds.), Proceedings of Geostatistics Banff 2004. Kluwer Academic Publisher, Banff, Alberta, pp. 215–224.

Deutsch, C., Journel, A., 1992. GSLIB: Geostatistical Software Library. Oxford University Press, New York 340 pp.

Deutsch, C., Tran, T., 2002. FLUVSIM: a program for object-based stochastic modeling of fluvial depositional systems. Computers & Geosciences 28 (4), 525–535.

Dimitrakopoulos, R., Luo, X., 2004. Generalized sequential Gaussian simulation on group size ν and screen-effect approximations for large field simulations. Mathematical Geology 36 (5), 567–591.

Geman, S., Geman, D., 1984. Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images. IEEE Transactions on Pattern Analysis and Matching Intelligence 6 (6), 721–741.

Guardiano, F., Srivastava, M., 1993. Multivariate geostatistics: beyond bivariate moments, Geostatistics-Troia. Kluwer Academic, Dordrecht, pp. 133–144.

Hu, L., Chugunova, T., 2008. Multiple-point geostatistics for modeling subsurface heterogeneity: a comprehensive review. Water Resources Research 44 (11), W11413.1–W11413.14.

Ingam, B., Cornford, D., 2008. Parallel geostatistics for sparse and dense datasets. In: Proceedings of geoENV VII 2008, Southampton, UK.

Isaaks, E., 1984. Indicator simulation: application to the simulation of a high grade uranium mineralization. In: Verly, G., David, M., Journel, A.G., Marechal, A.G. (Eds.), Geostatistics for Natural Resources Characterization, Part 2. D. Reidel Publishing Company, Dordrecht, Netherlands, pp. 1057–1069.

Journel, A., Alabert, F., 1990. New method for reservoir mapping. Journal of Petroleum Technology 42 (SPE paper 20781), 212–218.

Kerry, K., Hawick, K., 1998. Kriging interpolation on high-performance computers. In: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking, Amsterdam, The Netherlands, pp. 429–438.

Lantuéjoul, C., 2002. Geostatistical Simulation: Models and Algorithms. Springer, Berlin 232 pp.

Le Loc'h, G., Beucher, H., Galli, A.G., Doligez, B., 1994. Improvement in the truncated Gaussian method: combining several Gaussian functions. In: Proceedings of Ecmor 4, Fourth European Conference on the Mathematics of Oil Recovery, Roros, Norway, 13 pp.

Mariethoz, G., 2009. Geological stochastic imaging for aquifer characterization, Ph.D. Dissertation, University of Neuchâtel, Neuchâtel, Switzerland, 232 pp.

Mariethoz, G., Renard, P., Cornaton, F., Jaquet, O., 2009a. Truncated plurigaussian simulations to characterize aquifer heterogeneity. Ground Water 47 (1), 13–24.

Mariethoz, G., Renard, P., Froidevaux, R., 2009b. Integrating collocated auxiliary parameters in geostatistical simulations using joint probability distributions and probability aggregation. Water Resources Research 45 (W08421).

Mariethoz, G., Renard, P., 2010. Reconstruction of incomplete data sets or images using Direct Sampling. Mathematical Geosciences 42 (3), 245–268.

Matheron, G., 1973. The intrinsic random functions and their applications. Advances in Applied Probability 5, 439–468.

Ortiz, J.M., Peredo, O., 2008. Multiple point geostatistical simulation with simulated annealing: implementation using speculative parallel computing. In: Proceedings of geoENV VII—International Conference on Geostatistics for Environmental Applications, Southampton, UK, p. 384.

Pickard, D., 1980. Unilateral Markov fields. Advances in Applied Probability 12, 655–671.

Remy, N., Boucher, A., Wu, J., 2009. Applied Geostatistics with SGeMS: a User's Guide. Cambridge University Press, Cambridge 284 pp.

Straubhaar, J., Walgenwitz, A., Renard, P., Froidevaux, R., 2008. Optimization issues in 3D multipoint statistics simulation. In: Ortiz, J.M., Emery, X. (Eds.), Proceedings of Geostats 2008, Santiago, Chile, pp. 1035–1045.

Strebelle, S., 2002. Conditional simulation of complex geological structures using multiple point statistics. Mathematical Geology 34 (1), 1–22.

Vargas, H., Caetano, H., Filipe, M., 2007. Parallelization of sequential simulation procedures. In: Proceedings of Petroleum Geostatistics. EAGE (European Association of Geoscientists & Engineers), Cascais, Portugal.

Zhang, T., Switzer, P., Journel, A., 2006. Filter-based classification of training image patterns for spatial simulation. Mathematical Geology 38 (1), 63–80.