Math Geosci (2011) 43: 305–328
DOI 10.1007/s11004-011-9328-7

An Improved Parallel Multiple-point Algorithm Using a List Approach

Julien Straubhaar · Philippe Renard · Grégoire Mariethoz · Roland Froidevaux · Olivier Besson

Received: 16 December 2009 / Accepted: 16 August 2010 / Published online: 16 March 2011

Abstract Among the techniques used to simulate categorical variables, multiple-point statistics is becoming very popular because it allows the user to provide an explicit conceptual model via a training image. In classic implementations, the multiple-point statistics are inferred from the training image by storing all the observed patterns of a certain size in a tree structure. This type of algorithm has the advantage of being fast to apply, but it presents some critical limitations. In particular, a tree is extremely RAM demanding. For three-dimensional problems with numerous facies, large templates cannot be used. Complex structures are then difficult to simulate.

In this paper, we propose to replace the tree by a list. This structure requires much less RAM. It has three main advantages. First, it allows for the use of larger templates. Second, the list structure being parsimonious, it can be extended to include additional information. Here, we show how this can be used to develop a new approach for dealing with non-stationary training images. Finally, an interesting aspect of the list is that it allows one to parallelize the part of the algorithm in which the conditional probability density function is computed. This is especially important for large problems that can be solved on clusters of PCs with distributed memory or on multicore machines with shared memory.

Keywords Geostatistical simulation · Multiple-point statistics · Facies · Non-stationarity · Reservoir modeling · Parallel computing

J. Straubhaar (✉) · P. Renard · G. Mariethoz
Centre of Hydrogeology and Geothermics (CHYN), University of Neuchâtel, rue Emile–Argand 11, 2009 Neuchâtel, Switzerland
e-mail: julien.straubhaar@unine.ch

R. Froidevaux
Ephesia Consult SA, 9, rue Boissonnas, 1227 Geneva, Switzerland

O. Besson
Institute of Mathematics, University of Neuchâtel, rue Emile–Argand 11, 2009 Neuchâtel, Switzerland

1 Introduction

Since the work of Strebelle (2002), multiple-point statistics has emerged as a prominent field of research. Multiple-point statistics allows the integration of a complex conceptual geological model into a stochastic framework and is therefore among the most flexible method available. In particular, the user can provide crucial information about the shape of the structures, their degree of connectivity, or their spatial relations by simply defining an adequate training image. This is extremely important when one is interested in modeling flow and transport processes in reservoirs (Journel and Zhang 2006; Renard 2007).

While the principle of the method was proposed in the early 1990s by Guardiano and Strivastava (1993), the first efficient implementation, called *snesim*, was carried out by Strebelle (2002). In the original algorithm (Guardiano and Strivastava 1993), a scan of the entire training image was required for every simulation node. The main idea of Strebelle (2002) consists in storing the multiple-point statistics inferred from the training image in a catalog. The training image is scanned with a search template and the multiple-point statistics are stored in a dynamically allocated search tree (Strebelle 2002). During the simulation, the catalog is used to rapidly retrieve the information, thereby allowing the conditional probability density function to be computed for the current pixel. In addition, in order to ensure a good reproduction of the patterns at different scales while keeping the size of the search template small, Strebelle (2002) used a multigrid technique. A detailed analysis of the possibilities offered by *snesim* is given by Liu (2006). Several case studies have also shown the power of the method in real applications (Caers et al. 2003; Liu et al. 2004; Okabe and Blunt 2007; Ronayne et al. 2008).

In parallel, several improvements have been proposed. For example, a classic issue in multiple-point statistics is that structures that are connected in the training image may be disconnected in the simulation to the limited size of the search template and due to the random path in the sequential simulation. This problem was addressed by, for example, developing post-processing techniques (Strebelle and Remy 2005; Stien et al. 2007) and real-time post-processing (Suzuki and Strebelle 2007; Mariethoz et al. 2009), and by using a unilateral path (Daly and Knudby 2007) or a patched path (Zhang et al. 2008). Daly and Caers (2010) discuss some implementation and performance issues; for example, they suggest the use of a compact tree to reduce the RAM requirements of *snesim*.

Multiple-point statistics techniques require a stationary training image; otherwise, it is not possible to derive meaningful statistics. This may be a problem if one uses an image derived from field observations as a training image. To overcome that limitation, two techniques have been proposed. One is to split the training image in regions that are analyzed separately and for which different probability trees are constructed (de Vries et al. 2009). Another one is to use one or several secondary variables whose ranges of values are stored in the search tree in order to distinguish regions and patterns that should not be mixed (Chugunova and Hu 2008). Other authors have proposed to maintain the use of a training image but to completely change the simulation method. One of these approaches is the *simpat* method (Arpat and Caers 2007), which analyzes the patterns in the training image and reproduces them in the simulation.

Another technique, also based on pattern classification and simulation, is *filtersim* (Zhang et al. 2006; Wu et al. 2008). Compared to the previous methods, it has the advantage of being applicable to continuous variables. A survey and a review of these techniques can be found in Caers (2005), Hu and Chugunova (2008), or Remy et al. (2009).

In this paper, we propose a new implementation of the original multiple-point technique. As in the work of Strebelle (2002), the simulation is based on the computation of a conditional probability distribution derived from the training image; however, instead of using search trees, we use a simpler structure of lists. This has three key advantages. First, using lists allows to significantly reduce the amount of RAM required by the algorithm. This is particularly important for large three-dimensional training images for which the size of the search tree can quickly become prohibitive, adversely affecting the simulation of complex structures (Arpat and Caers 2007). Second, because the memory load is significantly reduced, one can easily use many different lists or store additional information within the lists. This is used for example to develop a new technique to deal with non-stationarity both in the training image and in the simulation grid. For that purpose, we use a couple of training images, including a standard categorical field plus a continuous auxiliary variable in a manner similar to Chugunova and Hu (2008). Third, the proposed implementation with lists is simpler to parallelize than the implementation with a search tree. This allows us to obtain fast simulations on single machines that have multicore processors, or on clusters that have multiple processors with distributed memory. The originality of the proposed parallelization approach is that instead of simultaneously simulating several realizations or several nodes of the same realization (Vargas et al. 2008; Mariethoz et al. 2009), we parallelize the most time-consuming step, the computation of the conditional probability density function at each node. The resulting improved parallel multiple-point algorithm using a list approach is called *impala* and is implemented in ANSI C.

2 Using Lists for Multiple-point Statistics

2.1 Notations and Basic Multiple-point Statistics Concepts

The objective is to simulate a categorical variable (facies) on a regular grid such that the multiple-point statistics and the simulated patterns are similar to the ones of a training image. The simulation of a facies at a grid node proceeds as follows. A search template is predefined in the beginning of the simulation as a set of N relative node locations (offsets) $h_1, \ldots, h_N$, where h_i is a two-dimensional or three-dimensional vector, $1 \leq i \leq N$. For a given reference (central) node u, the search template τ at u is the set of nodes

$$\tau(u) = \{u + h_1, \ldots, u + h_N\}, \tag{1}$$

and, if $s(v)$ denotes the facies at node v, the vector, such that

$$d(u) = \{s(u + h_1), \ldots, s(u + h_N)\} \tag{2}$$

defines the data event d at u.

To attribute a facies at a node u in the simulation grid, the nodes v in the training image (TI) where the data event $d(v)$ has the same components as those of $d(u)$ are retained. Then the occurrences of all the facies at the nodes v are counted. This provides a conditional probability distribution function (CPDF) that can be used to randomly draw a facies at the node u. More precisely, if there are n defined components in $d(u)$, in positions $i_1 < \cdots < i_n$, with $0 \leq n \leq N$, then the probability to draw the facies k at the node u is

$$\mathbb{P}\big(s(u) = k \mid d(u)\big) = \frac{\#\{v \in TI : s(v + h_{i_j}) = s(u + h_{i_j}),\ 1 \leq j \leq n \text{ and } s(v) = k\}}{\#\{v \in TI : s(v + h_{i_j}) = s(u + h_{i_j}),\ 1 \leq j \leq n\}}, \tag{3}$$

for $0 \leq k < M$, where M is the number of facies. In (3), the denominator represents the number of matching data events in the training image, and the numerator represents the number of these data events having facies k at the central node. If the denominator is too small, the last informed node in $d(u)$ is dropped until the number of replicates is acceptable, (i.e., larger than a predefined threshold).

In addition, the multigrid approach (Tran 1994; Strebelle 2002) is used to capture structures at different scales. Let us recall that the simulation grid is divided in m multigrids (Strebelle 2002). The simulation proceeds successively with the simulation of all the nodes whose coordinates are a multiple of 2^{m-1}. The lag vectors in the search template are then scaled by 2^{m-1}. The process continues with the unsimulated nodes whose coordinates are a multiple of 2^{m-2}, and repeats itself similarly for all the multigrid levels.

2.2 Storing Multiple-point Statistics in Lists

In this paper, instead of using the tree structure proposed by Strebelle (2002), we store the multiple-point statistics inferred from the training image in a list. An element of the list is a pair of vectors (d, c) where $d = (s_1, \ldots, s_N)$ defines a data event and $c = (c_0, \ldots, c_{M-1})$ is a list of occurrences counters for each facies: c_i is the number of data events $d(v)$ equal to d found in the training image with facies i at the reference node v. To illustrate this point, the training image and the search template of Fig. 1 are considered. The list in Fig. 2 is obtained by scanning the training image in such a way that the search template is always entirely inside the training image. The facies code 0 (resp. 1) is used for white (resp. gray) nodes. Note that the training image can be scanned with the search template in such a way that its reference node covers all the nodes of the training image. This allows one to take into account the patterns around the boundaries of the training image. In this case, some nodes of the search template can be outside the training image, and a new value for lacking facies (e.g., -1) then has to be introduced for corresponding components of data events stored in lists.

This type of list is a catalog that allows to compute the CPDF (3) for performing simulations. The training image is used only to build the lists. For each multigrid

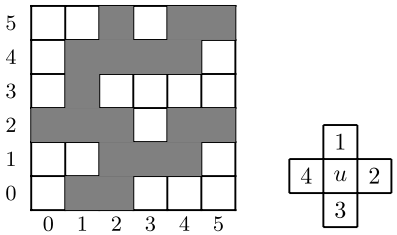

Fig. 1 Training image of dimensions 6×6 and search template of size $N = 4$. In the training image, the *white* nodes have facies code 0 and the *gray* nodes facies code 1. In the search template, u represents the reference (central) node and the numbers the indices of the relative locations

Elements of the list	Corresponding ref. nodes
$L_0 = ((0,0,1,1),(0,1))$	$\{\emptyset, \{(2,2)\}\}$
$L_1 = ((0,1,0,1),(0,2))$	$\{\emptyset, \{(3,1),(3,4)\}\}$
$L_2 = ((0,1,1,0),(0,2))$	$\{\emptyset, \{(1,4),(4,2)\}\}$
$L_3 = ((0,1,1,1),(1,0))$	$\{\{(3,2)\}, \emptyset\}$
$L_4 = ((1,0,0,0),(1,0))$	$\{\{(3,3)\}, \emptyset\}$
$L_5 = ((1,0,0,1),(0,2))$	$\{\emptyset, \{(4,1),(4,4)\}\}$
$L_6 = ((1,0,1,0),(1,1))$	$\{\{(4,3)\}, \{(1,3)\}\}$
$L_7 = ((1,0,1,1),(1,0))$	$\{\{(2,3)\}, \emptyset\}$
$L_8 = ((1,1,0,1),(0,2))$	$\{\emptyset, \{(1,2),(2,4)\}\}$
$L_9 = ((1,1,1,0),(1,1))$	$\{\{(1,1)\}, \{(2,1)\}\}$

Fig. 2 List for the training image and the search template of Fig. 1 (scanning in a way that the search template is always entirely inside the training image). The elements of the list (*first column*) are pairs of vectors (d, c) where $d = (s_1, \ldots, s_4)$ defines a data event found in the training image, and the components of $c = (c_0, c_1)$ are the number of replicates of the data event d with facies 0 and 1, respectively, at the reference node. The reference nodes (*second column*) are the locations of the central nodes in the training image where d is found; two sets of size c_0 and c_1 are given for the replicates having facies 0 and 1, respectively, at the reference node

level, the corresponding list is obtained by scanning the training image with the appropriate search template once. Then, the training image can be removed from the memory.

2.3 Computing the CPDF from a List

The CPDF used for drawing a facies at a node u of the simulation grid is computed based on the list. Equation (3) is used and a minimal number of replicates, C_{min}, is given. Assume that the simulated nodes in the data event $d(u)$ centered at u are $u + h_{i_1}, \ldots, u + h_{i_n}$. For $1 \leq j \leq n$, let $d^{(j)}(u)$ be the data event whose defined components are $s(u + h_{i_1}), \ldots, s(u + h_{i_j})$, (i.e., the data event corresponding to the first j simulated nodes in $d(u)$). For $1 \leq j \leq n$ and $0 \leq k < M$, let $C_k^{(j)}$ be the number of replicates of $d^{(j)}(u)$ found in the training image with facies k at the central node (i.e., the number of nodes v in the training image verifying $s(v + h_{i_l}) = s(u + h_{i_l})$, $1 \leq l \leq j$ and $s(v) = k$). Then, the greatest index j such that the total number of

replicates of $d^{(j)}(u)$, $C^{(j)} = \sum_{k=0}^{M-1} C_k^{(j)}$ is greater or equal to C_{min} is selected and the corresponding CPDF

$$\mathbb{P}\big(s(u) = k \mid d^{(j)}(u)\big) = \frac{C_k^{(j)}}{C^{(j)}}, \quad 0 \le k < M \tag{4}$$

is used to draw the facies at the node u.

The replicates of the data events are counted in the list. First, all the counters $C_k^{(j)}$ are set to 0. Then each element (d, c) in the list is scanned, and all data events $d^{(j)}(u)$ are compared to d: if they are compatible (i.e., $d_{i_l} = s(u + h_{i_l})$, $1 \le l \le j$), the counter $C_k^{(j)}$ is increased by c_k for $0 \le k < M$. Note that if $C^{(1)} < C_{min}$, or if no node in $d(u)$ is informed ($n = 0$), the marginal PDF (i.e., the proportion of each facies in the training image) is used.

Let us illustrate the computation of the CPDF on an example with the training image and the search template of Fig. 1. Assume that the node u to be simulated has a gray node on both sides (right and left) and that neither nodes below nor above u are informed ($n = 2$, $i_1 = 2$, $i_2 = 4$, $d(u) = \{?, 1, ?, 1\}$). In this situation, the counters

$$C_0^{(2)} = 1, \qquad C_1^{(2)} = 4 \tag{5}$$

are obtained by summing the counters of elements L_1, L_3, L_8 of the list of Fig. 2. If the node at the left of u (node $u + h_4$) is ignored, the contribution of the elements L_2 and L_9 is added and the corresponding counters are

$$C_0^{(1)} = 2, \qquad C_1^{(1)} = 7. \tag{6}$$

Thus, if C_{min} is lower or equal to 5, the counters $C_0^{(2)}$ and $C_1^{(2)}$ are used to retrieve the CPDF $(1/5, 4/5)$ for the facies code $(0, 1)$ at u.

2.4 Sorting Lists

Computing the CPDF (4) based on a list requires scanning all elements of the list, which can be CPU-expensive. In this section, we propose to alleviate the CPU burden by scanning only a portion of a sorted list. Let $N_{\mathcal{L}}$ be the number of elements in the list and

$$\mathcal{L} = \big\{L_0 = (d_0, c_0), \ldots, L_{N_{\mathcal{L}}-1} = (d_{N_{\mathcal{L}}-1}, c_{N_{\mathcal{L}}-1})\big\} \tag{7}$$

the list itself. Let s_0 be a reference facies and, for a data event d, let $N_{s_0}(d)$ be the number of nodes having a facies equal to s_0 in d. The idea is to group the elements whose data event has the same number of times the facies s_0. The list is sorted such that all couples of elements $L_i = (d_i, c_i)$ and $L_j = (d_j, c_j)$ verify

$$i \le j \quad \Longleftrightarrow \quad N_{s_0}(d_i) \le N_{s_0}(d_j), \quad 0 \le i, j < N_{\mathcal{L}}. \tag{8}$$

The sorted list is divided in $N + 1$ classes, the elements whose data event has i times the facies s_0 form the ith class,

$$K_{s_0}(i) = \big\{L = (d, c) \in \mathcal{L} : N_{s_0}(d) = i\big\}, \quad 0 \le i \le N. \tag{9}$$

Elements of the list			class indices (N_0)	Corresp. ref. nodes
L_0	=	$((0,1,1,1),(1,0))$	1	$\{\{(3,2)\},\emptyset\}$
L_1	=	$((1,0,1,1),(1,0))$	1	$\{\{(2,3)\},\emptyset\}$
L_2	=	$((1,1,0,1),(0,2))$	1	$\{\emptyset,\{(1,2),(2,4)\}\}$
L_3	=	$((1,1,1,0),(1,1))$	1	$\{\{(1,1)\},\{(2,1)\}\}$
L_4	=	$((0,0,1,1),(0,1))$	2	$\{\emptyset,\{(2,2)\}\}$
L_5	=	$((0,1,0,1),(0,2))$	2	$\{\emptyset,\{(3,1),(3,4)\}\}$
L_6	=	$((0,1,1,0),(0,2))$	2	$\{\emptyset,\{(1,4),(4,2)\}\}$
L_7	=	$((1,0,0,1),(0,2))$	2	$\{\emptyset,\{(4,1),(4,4)\}\}$
L_8	=	$((1,0,1,0),(1,1))$	2	$\{\{(4,3)\},\{(1,3)\}\}$
L_9	=	$((1,0,0,0),(1,0))$	3	$\{\{(3,3)\},\emptyset\}$
Associated class indices vector: $I_0 = I = (I(0) = 0, I(1) = 0, I(2) = 4, I(3) = 9, I(4) = 10, I(5) = 10)$				

Fig. 3 List sorted by classes for the facies $s_0 = 0$ for the training image and the search template of Fig. 1 (scanning in a way that the search template is always entirely inside the training image). The elements of the list (*first column*) are pairs of vectors (d, c) where $d = (s_1, \ldots, s_4)$ defines a data event found in the training image, and the components of $c = (c_0, c_1)$ are the number of replicates of the data event d with facies 0 and 1, respectively, at the reference node. The class indices (*second column*) represent the number of times that the facies $s_0 = 0$ appears in d. The reference nodes (*third column*) are the locations of the central nodes in the training image where d is found; two sets of size c_0 and c_1 are given for the replicates having facies 0 and 1, respectively, at the reference node

The sorting criterion described by (8) only gives the positions of the classes (9) with respect to each other. Each class can be sorted using the lexicographical order on the data events. For locating each class, pointers to the beginning of each class are used. More precisely, if the list (7) is sorted according to the relation (8), a vector $I_{s_0} = (I_{s_0}(0), \ldots, I_{s_0}(N+1))$ of length $N+2$ is defined by

$$I_{s_0}(i) = \min\left(\left\{j : N_{s_0}(d_j) \geqslant i\right\} \cup \{N_{\mathcal{L}}\}\right) \tag{10}$$

for $0 \leq i \leq N+1$. Hence, the elements of the ith class, $K_{s_0}(i)$, are the elements L_j, $I_{s_0}(i) \leq j < I_{s_0}(i+1)$. The list for the training image and the search template of Fig. 1, sorted by classes for the facies $s_0 = 0$, is given in Fig. 3.

2.4.1 Retrieving the CPDF from the List Sorted by Classes

Assume that for a node u to be simulated, the data event $d(u)$ has n informed nodes $u + h_{i_1}, \ldots, u + h_{i_n}$, and that the list is sorted by classes for the reference facies s_0. In this situation, the data events that are compatible with $d(u)$ necessarily have at least $N_{s_0}(d(u))$ and at most $N - (n - N_{s_0}(d(u)))$ times the facies s_0, where N is the size of the search template. To compute the counters in the case where some nodes in $d(u)$ have to be dropped, we introduce for $1 \leq j \leq n$ the set

$$\begin{aligned}\mathcal{L}_{s_0}\left(d^{(j)}(u)\right) = \big\{L_i : I_{s_0}\left(N_{s_0}\left(d^{(j)}(u)\right)\right) \\ \leq i < I_{s_0}\left(N - \left(j - N_{s_0}\left(d^{(j)}(u)\right)\right) + 1\right)\big\}\end{aligned} \tag{11}$$

of the elements $L = (d, c)$ of the list whose data event d can be compatible with $d^{(j)}(u)$. These sets verify

$$\mathcal{L}_{s_0}\big(d(u)\big) = \mathcal{L}_{s_0}\big(d^{(n)}(u)\big) \subset \mathcal{L}_{s_0}\big(d^{(n-1)}(u)\big) \subset \cdots \subset \mathcal{L}_{s_0}\big(d^{(1)}(u)\big). \tag{12}$$

If $s(u + h_{i_{j+1}}) = s_0$ (resp. $s(u + h_{i_{j+1}}) \neq s_0$), the set $\mathcal{L}_{s_0}(d^{(j)}(u))$ is obtained by appending one class (maybe empty) of elements of the list to $\mathcal{L}_{s_0}(d^{(j+1)}(u))$ at the beginning (resp. at the end).

For computing the counters $C_k^{(j)}$, $0 \leq k < M$ in (4), we need to scan the elements of the list in $\mathcal{L}_{s_0}(d^{(j)}(u))$ only. The range of the list that is scanned is enlarged following (12) until the minimal number of replicates is reached; the corresponding CPDF is then chosen. More specifically, retrieving the CPDF and drawing the facies at u is done in the following way:

(1) The elements of the list in $\mathcal{L}_{s_0}(d(u))$ are scanned and the corresponding counters $C_k^{(j)}$, $0 \leq k < M$, $1 \leq j \leq n$ are computed. At this step, the final values of $C_k^{(n)}$ are obtained, while for $C_k^{(j)}$, $1 \leq j < n$, the final values are not reached yet.
(2) If the criterion of the minimal number of replicates is satisfied for the counters $C_k^{(n)}$, (i.e., $C^{(n)} = \sum_{k=0}^{M-1} C_k^{(n)} \geqslant C_{min}$), the CPDF $\mathbb{P}(. \mid d^{(n)}(u))$ $(= \mathbb{P}(. \mid d(u)))$ is computed and used for drawing the facies at u. Otherwise, the last node in the data event is dropped until this criterion is satisfied. The index j is set to $n - 1$ and the following steps are then implemented:
   (a) If $j = 0$, the marginal PDF is used for drawing the facies at u. Otherwise, the elements of the list in $\mathcal{L}_{s_0}(d^{(j)}(u)) \setminus \mathcal{L}_{s_0}(d^{(j+1)}(u))$ are scanned and the counters $C_k^{(l)}$, $0 \leq k < M$, $1 \leq l \leq j$ are updated. At this step, the final values of $C_k^{(j)}$ are obtained, while for $C_k^{(l)}$, $1 \leq l < j$, the final values are not reached yet.
   (b) If the criterion of the minimal number of replicates is satisfied for the counters $C_k^{(j)}$ (i.e., $C^{(j)} = \sum_{k=0}^{M-1} C_k^{(j)} \geqslant C_{min}$), the CPDF $\mathbb{P}(. \mid d^{(j)}(u))$ is computed and used for drawing the facies at u. Otherwise, the index j is decreased by 1, $j = j - 1$, and the process continues in Step (a).

Also note that the vector I_{s_0}, associated to the sort of the list, is used for determining the part of the list to be scanned in the steps above. Sorting lists by classes requires a reference facies s_0. Any facies s_0 could be chosen. Practically, the dominant facies in the training image is used as reference facies s_0, because in most of cases it results in classes of relatively homogeneous sizes, thereby maximizing CPU efficiency.

3 RAM Usage

In this section, the RAM usage is compared for implementations of a multiple-point algorithm based on search trees and lists.

3.1 Estimation of RAM Requirements

Let us recall the structure of the search tree for estimating its RAM requirements. In a situation with a search template of size N and M facies, the search tree has a depth of N and is made up of cells divided into M subcells. Each subcell can store a counter and can have a child cell (in the next level of the tree); the tree is an M-ary tree. If the levels in the tree are numbered from 1 to $N+1$ and the subcells in a cell from 0 to $M-1$, the counter in a subcell is defined as follows. Let $\{i(1), i(2), \ldots, i(k)\}$ be a path in the tree where $i(j)$ is the identification number of a subcell in a cell of level j. The counter in the last sub-cell of the path is the number of data events found in the training image with facies $i(j)$ at node $v + h_j$ of the search template $\tau(v)$ for $1 \leq j \leq k-1$, and with facies k at the reference node v. Figure 4 shows the search tree corresponding to the training image and the search template of Fig. 1, scanning with the search template always entirely inside the training image. The statistics stored in the list shown in Figs. 2 and 3 correspond to the leaves of the search tree of Fig. 4; an important point to emphasize is that the list makes it possible to reconstruct the search tree with precision.

The size of the search tree and the list depends on the size N of the search template, the number M of facies and the entropy of the training image. Moreover, the size of the search tree can also depend on the numbering of the nodes in the search template. The previous descriptions make it possible to compute the ratio between the size of the search tree and the size of the list. Let s_{tree} be the size in octets of a tree cell, s_{list} the size in octets of an element of the list and $r = s_{tree}/s_{list}$ their ratio. Let K_{tree} and K_{list} be the number of cells in the tree and the number of elements in the list, respectively. Then, the number

$$R = \frac{K_{tree}}{K_{list}} \cdot r \tag{13}$$

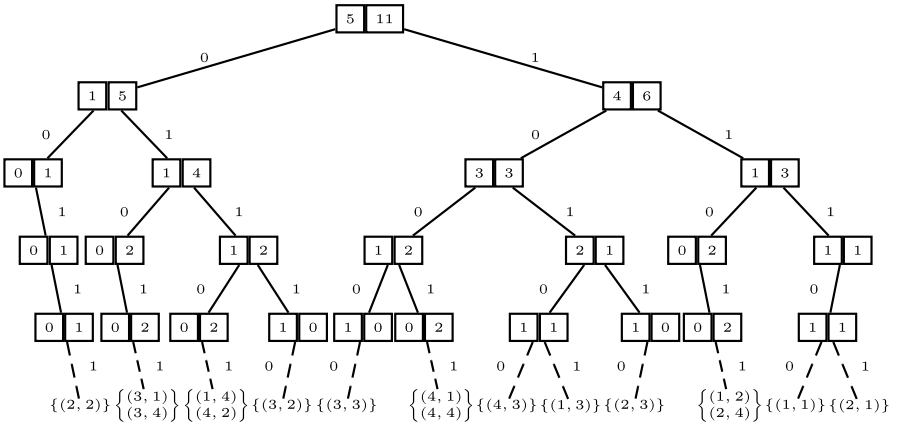

Fig. 4 Search tree for the training image and the search template of Fig. 1 (scanning in a way that the search template is always entirely inside the training image). The corresponding reference nodes (location in the training image) are given below the leaves of the tree

denotes the gain factor in terms of RAM usage for the list in comparison with the tree. In the lowest possible entropy case, there is only one data event in the training image and we have

$$K_{tree} = N + 1 \quad \text{(degenerated tree)}, \qquad K_{list} = 1 \quad \text{and} \quad R = (N+1) \cdot r. \tag{14}$$

In the highest possible entropy case, all possible data events appear in the training image and we have

$$K_{tree} = 1 + M + M^2 + \cdots + M^N = \frac{M^{N+1} - 1}{M - 1} \quad \text{(full } M\text{-ary tree)}, \qquad K_{list} = M^N \tag{15}$$

and

$$R = \frac{1 - M^{-(N+1)}}{1 - M^{-1}} \cdot r. \tag{16}$$

The degenerated tree corresponds to the worst case in terms of memory usage, whereas the full M-ary tree is the most favorable case. The gain factor is always within the interval given by these two extreme values of R

$$R \in \left[\frac{1 - M^{-(N+1)}}{1 - M^{-1}} \cdot r, (N+1) \cdot r \right]. \tag{17}$$

An estimation for r is calculated as follows. A tree cell is divided in M subcells which each contain a counter and the address of its child cell, whereas an element of the list contains a data event made up of N facies codes and M counters. Assume that a counter is coded in a long integer of size 8, an address is of size 8, and a facies code is stored in a character of size one (the sizes are in octets, if nothing is mentioned). Then, $s_{tree} = (8+8) \cdot M = 16 \cdot M$, $s_{list} = N + 8 \cdot M$ and

$$r = \frac{16 \cdot M}{N + 8 \cdot M}. \tag{18}$$

For the example in Fig. 1, $N = 4$, $M = 2$, $r = 1.6$ and $R = 3.84 \in [3.1, 8]$. For a three-dimensional application (Sect. 3.2) with a search template of size $N = 124$ ($5 \times 5 \times 5$ box around the reference node) and $M = 4$ facies, we have $r = 0.410256$ and $R \in [0.5470, 51.2821]$. Note that the extreme cases are never reached in practice.

3.2 RAM Limitation

We show that the RAM limit (*i.e.,* the total amount of available RAM on a given machine) for storing multiple-point statistics is reached faster with the search tree approach than with the list based approach. Consider a three-dimensional training image and a search template that contains the N closest nodes to the reference node and let the size N be increased. For each search template, the corresponding search tree and list are built and their respective sizes in gigabytes (GB) are calculated. Figure 5(a) shows the results for a three-dimensional training image of dimensions $100 \times 100 \times 60$ with 4 facies and representing a boolean model of fluvial deposit.

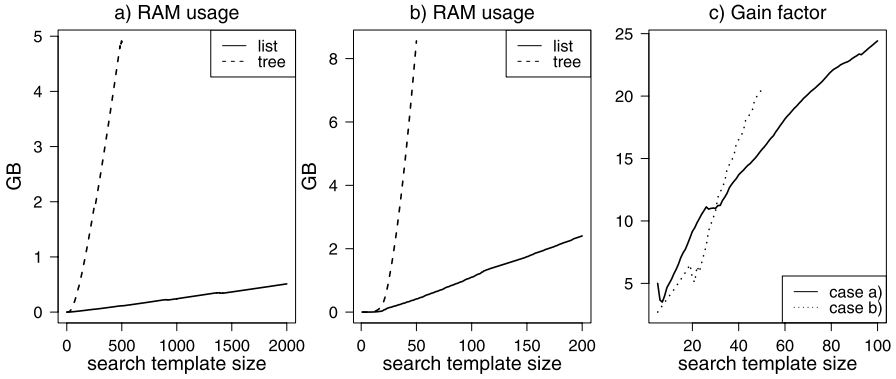

Fig. 5 (**a**) RAM usage for a three-dimensional training image having 600,000 nodes and 4 facies. (**b**) RAM usage for a three-dimensional training image having 14,128,385 nodes and 6 facies. (**c**) Gain factor with the list in comparison with the search tree (ratio: size of tree over size of list)

Figure 5(b) shows the results for a three-dimensional training image of dimensions $101 \times 101 \times 1385$ with 6 facies and representing an alluvial plain simulated with a pseudo-genetic technique (Rivoirard et al. 2008). The gain factor R for these two cases is drawn in Fig. 5(c).

These experimental values show that the list-based approach quickly results in the reduction of RAM requirements by a factor of 10 or more. Moreover, the search templates size can be significantly limited by the RAM capacity when using the search tree approach. Indeed, in the case shown in Fig. 5(b), the RAM required by the tree reaches 2 GB with already 30 nodes. However, such a limited number of nodes is often insufficient to characterize complex three-dimensional structures.

4 CPU Performance for Serial Implementations

In this section, serial implementations of a multiple-point algorithm using lists sorted lexicographically, and lists sorted by classes (based on the dominant facies) and search trees, are compared in terms of CPU performance. Two three-dimensional training images are used in the following tests. The number of multigrids is set to 3, and the same search template is used for each multigrid. Several symmetric search templates of increasing sizes ($N = 6$, 18, 26, 32, 44, 56, 80, ...) are considered, and for each one, a single realization is generated. Each method leads to the same simulated images, provided that the path and the random numbers used in the simulations are identical. In Fig. 6(a), the real time spent for each method is shown as a function of the search template size for a three-dimensional training image with dimensions of $100 \times 100 \times 60$ and with 4 facies (same image as that one used for Fig. 5(a). Figure 6(b) shows the times for a three-dimensional training image with dimensions of $101 \times 101 \times 536$ and with 6 facies (part of the image used for Fig. 5(b)).

The algorithm based on trees is often faster than the algorithm using lists, provided that the RAM capacity is not exceeded. This is because in a search tree, statistics for

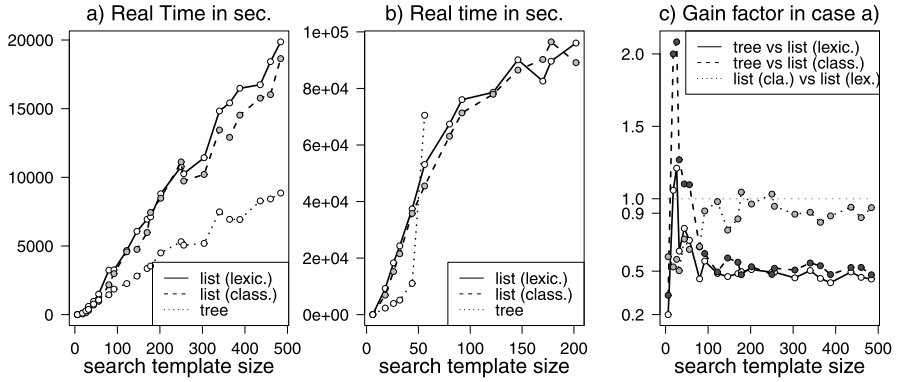

Fig. 6 Real time for the algorithm using lists sorted lexicographically, lists sorted by classes and trees with (**a**) a three-dimensional training image having 600,000 nodes and 4 facies (3 multigrids), (**b**) a three-dimensional training image having 5,467,736 nodes and 6 facies (3 multigrids), and (**c**) a gain factor using lists in comparison with search trees (ratio: time using trees over times using lists), and using lists sorted lexicographically in comparison with lists sorted by classes

data event partially informed are stored in the levels above the leaves, whereas if these statistics are needed, they have to be computed based on the list. For large training images, however, the simulations become unmanageable due to the size of the search trees. This situation is already met for a search template of size $N = 80$ in Fig. 6(b), which is why the times for the algorithm using search trees are not given for larger values of N. Figure 6(c) shows that for large search templates, the classical approach using search trees is approximately two times faster than the algorithm based on lists. This trend is reversed for small search templates, which is probably due to the recursion process used for retrieving statistics from the tree. This figure also shows that the algorithm using lists sorted by classes is more efficient than the one using the lexicographical sort: This is clear for small search templates and remains true for large ones (excepted for two cases) with a gain of approximately 10% of the CPU time.

It is important to emphasize that the use of lists makes it possible to parallelize the algorithm easily, thus allowing one to catch up with the tree version when it comes to CPU time. This is discussed in the next section.

5 Parallelization

Using lists instead of trees allows a straightforward parallelization of the algorithm. The parallelization is based on Message Passing Interface (MPI) technology (Dongarra et al. 1998; Gropp et al. 1998). In the next sections, p denotes the number of processors that are used: they are numbered from 0 to $p - 1$. Note that an OpenMP version of the code was also developed. It shows similar performances but is dedicated to smaller machines with shared memory.

Table 1 Parallelization for the list storage $\mathcal{L} = \{L(0), \ldots, L(N_{\mathcal{L}} - 1)\}$ using p processors; $N_{\mathcal{L}} = q \cdot p + r$ is the Euclidean division of $N_{\mathcal{L}}$ by p

Process	Elements of the partial list				
0	$L(0)$	$L(p)$	$\ldots$	$L((q-1)\cdot p)$	$L(q\cdot p)$
1	$L(1)$	$L(p+1)$	$\ldots$	$L((q-1)\cdot p+1)$	$L(q\cdot p+1)$
$\vdots$					
$r-1$	$L(r-1)$	$L(p+r-1)$	$\ldots$	$L((q-1)\cdot p+r-1)$	$L(q\cdot p+r-1)$
r	$L(r)$	$L(p+r)$	$\ldots$	$L((q-1)\cdot p+r)$	
$r+1$	$L(r+1)$	$L(p+r+1)$	$\ldots$	$L((q-1)\cdot p+r+1)$	
$\vdots$					
$p-1$	$L(p-1)$	$L(p+p-1)$	$\ldots$	$L((q-1)\cdot p+p-1)$	

5.1 Distributing Lists

Each list is distributed over all p processors. It is divided into p parts of balanced size and each one of these p partial lists is stored in the memory space dedicated to a single processor. More precisely, if $N_{\mathcal{L}}$ is the number of elements in the list and $N_{\mathcal{L}} = q \cdot p + r$ is the Euclidean division of $N_{\mathcal{L}}$ by p, there are r partial lists with $q+1$ elements and $p-r$ partial lists with q elements. Let

$$\mathcal{L} = \{L(0), \ldots, L(N_{\mathcal{L}} - 1)\} \tag{19}$$

be the entire list, sorted by classes, as explained in Sect. 2.4. Then the partial lists stored in the memory attached to the different processors are given in Table 1, and each processor computes the vector I containing the indices of the beginning of each class, associated to its partial list (10). In this situation, two consecutive elements of the entire list are stored on two consecutive processors according to their identification number, assuming that the processor 0 follows the processor $p-1$. The computational load is well balanced between all processors, provided that each class of elements in the entire list is distributed over all available processors.

Note that every list required by the simulation (one per multigrid and zone) is built by scanning a training image. This step is carried out ahead of the distribution above, and can also be parallelized. Parallelization of the lists' construction is accomplished by first dividing the training image in p regions. Then each processor builds the list corresponding to one region; finally, all of these temporary lists are grouped together into one list. Construction of the lists is not a heavy task.

5.2 Simulation with Distributed Lists

For computing the CPDF (3), each processor retrieves the required occurrences counters by scanning its local list. The information retrieved by all processors is then merged using a parallel computing communication, and the CPDF is computed. Therefore, the simulation of each successive node is parallelized.

More precisely, the main steps of the parallelized algorithm are the following. The simulation grid is stored and updated at each simulated node on all processors. In order to simulate the facies at one node u, each processor retrieves the data event centered at u and retrieves from its local list the occurrences counters relative to the data event, as described in Sect. 2.4. In particular, the scan of the partial list is enlarged until the sum of the local occurrences counters reaches the criterion of the minimal number of replicates (or until the partial list is entirely scanned). This guarantees that the criterion of minimal number of replicates is met with only one parallel communication. The CPDF is computed based on the gathered information and is used for drawing a facies at u. Only one of the processors (typically processor 0) actually draws the facies value. This processor sends the simulated value to the others, and the simulation proceeds to the next node. Therefore, two parallel communications are required for each simulated node.

5.3 Performance Measurements: CPU Time and Speed-up

The performance of a parallel algorithm in terms of CPU time is experimentally evaluated using following measurements. A simulation with the same input parameters is launched on $p = 1, 2, 4, 6, 8, \ldots$ processors. For each test, the CPU time spent on each processor is retrieved. Let $T_p(i)$ be the time spent by processor i, $0 \leq i \leq p-1$.

5.3.1 Load Balancing

The times $T_p(i)$, $0 \leq i \leq p-1$, allow to evaluate how the computational load is balanced. If large discrepancies appear in the workload of the different processors, it means that the load balancing is not optimal and that the parallelization is poor. Conversely, if all processors spend similar CPU time, it means that the load balancing is optimal.

5.3.2 CPU Time Max

We observe how the CPU time max $T_p = \max_{0 \leq i \leq p-1} T_p(i)$ decreases as a function of p.

5.3.3 Speed-up and Efficiency

The speed-up and the efficiency are respectively the ratios

$$S_p = \frac{T_1}{T_p} \quad \text{and} \quad E_p = \frac{S_p}{p}. \tag{20}$$

They can be compared to the ideal curves $S_p = p$ and $E_p = 1$; that is, to the situation where the time is divided by p when the number of used processors is multiplied by p. Note that the efficiency gives a normalized rate.

5.4 Numerical Tests

Three sets of tests are considered for evaluating the simulation part of the proposed algorithm, using the sort of the list by classes according to the dominant facies. The main parameters of these tests are given in Table 2 and the sizes of the lists for each multigrid are given in Table 3. Note that for test 2, 20 zones in the simulation grid are used for rotations, and that twenty lists are then used for each multigrid. Moreover, an auxiliary variable is considered for this test and the extended lists are then used (Sect. 6).

The evaluation of load balancing, maximum CPU time and speed-up, and efficiency curves for the three sets of tests are given in Figs. 7 and 8. Figures 7(a), 7(c), and 7(e) show that the computational load is well balanced, while Figs. 7(b), 7(d), and 7(f) show that the maximum CPU time is strongly decreasing. This results in good speed-up (and efficiency) curves, especially for small numbers of processors (Fig. 8). Table 3 and Fig. 8 show that the speed-up curves are closer to the ideal curve for the largest examples. This is typical when using parallel codes: the number of processors that allow for a significant decrease in CPU time depends on the computational burden of the problem to be solved. For small problems, the time spent on communications between processors quickly exceeds the time used for the actual computations.

Table 2 Main parameters of the tests for evaluating parallel performances

Main parameters	Test 1	Test 2	Test 3
Dim. of TI	$100 \times 100 \times 60$	1501×551	$200 \times 160 \times 200$
Dim. of sim. grid	$100 \times 100 \times 60$	1500×900	$120 \times 96 \times 120$
Number of facies	4	3	4
Number of aux. variables	0	1	0
Number of zones	1	20	1
Number of multigrids	3	3	3
Template sizes (by multig.)	202, 122, 56	120, 96, 56	170, 146, 92
Number of realizations	1	10	1

Table 3 Approximative number of elements in each entire list for the tests of evaluation of the parallelization (Table 2)

	Test 1	Test 2	Test 3
Coarse multigrid	474,000	720,000	6,252,000
Middle multigrid	265,000	489,000	5,113,000
Fine multigrid	91,000	170,000	2,122,000

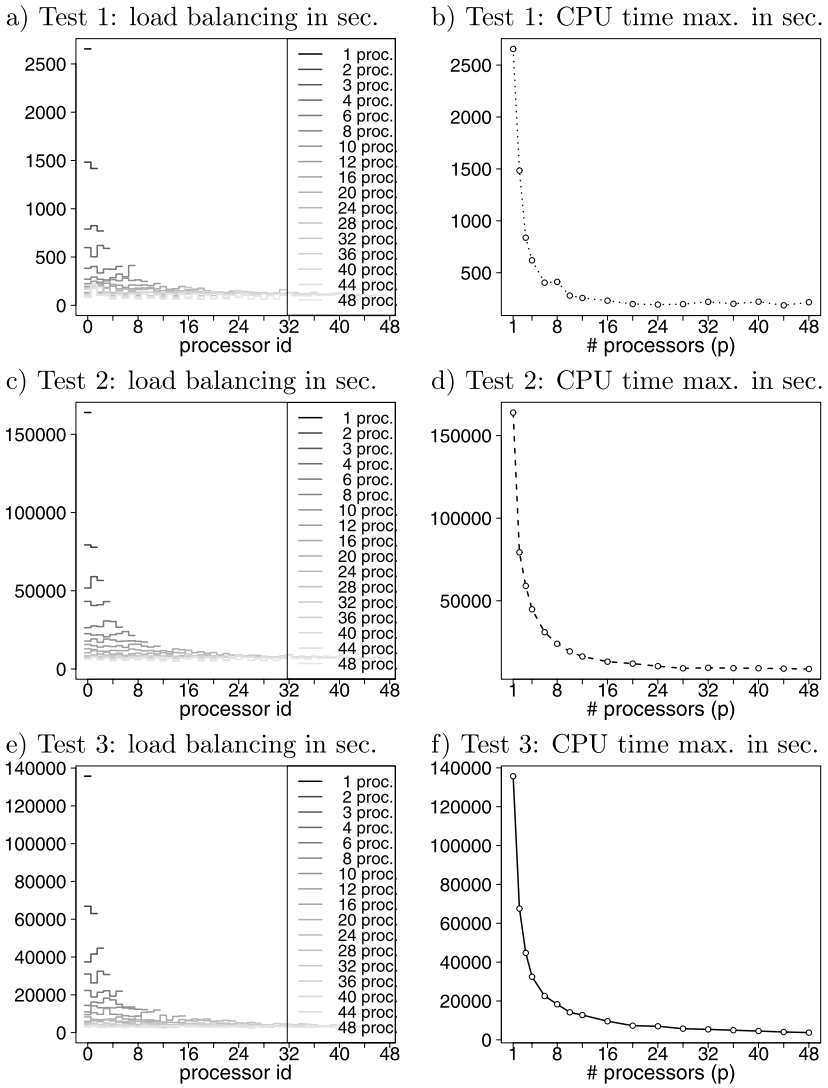

Fig. 7 Load balancing (LB) and CPU time max. (CPU) for tests of Table 2: (**a**) LB for test 1; (**b**) CPU for test 1; (**c**) LB for test 2; (**d**) CPU for test 2; (**e**) LB for test 3; (**f**) CPU for test 3

6 Extending the List to Account for Non-stationarity

6.1 Non-stationary Training Images

The multiple-point simulation method described above is valid for stationary training images because spatial patterns (pixels configurations) according to a search template are stored in the list regardless of their location in the training image. In this section,

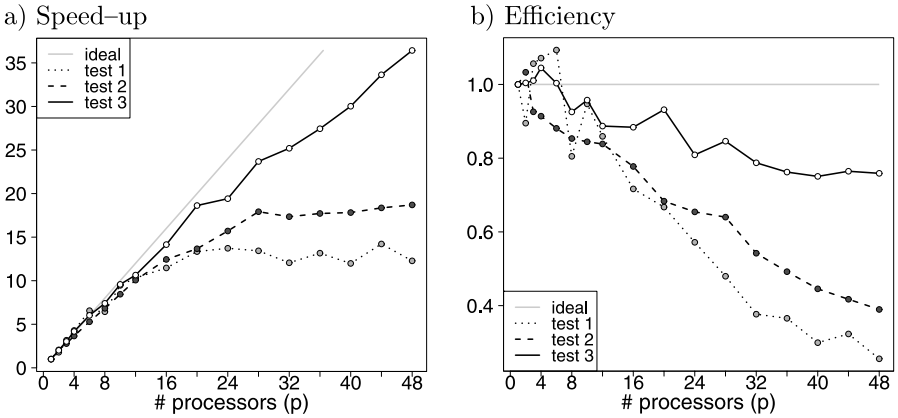

Fig. 8 Performance for tests of Table 2: (**a**) speed-up curves and (**b**) efficiency curves

a method for performing simulations with nonstationary training images using lists is presented.

Our approach is inspired from the work of Chugunova and Hu (2008) which consists in using continuous auxiliary variable(s) for describing nonstationarity. For a nonstationary training image (TI) (containing facies, which constitutes the primary variable), one or several auxiliary map(s) is (are) associated, and each one contains an auxiliary variable describing some form of nonstationarity within the TI. Each of these auxiliary variables must be exhaustively known on the simulation domain for guiding the simulation. One or several auxiliary maps associated to the simulation grid (SG) are also required. Note that the method described in the next sections can be easily integrated in a parallel implementation, following the steps presented in Sect. 5.

6.2 Storing an Auxiliary Variable in Lists

In the presence of an auxiliary variable, a vector m is appended to each element of the list. An element of the list is then a triplet of vectors (d, c, m) where $d = (s_1, \ldots, s_N)$ defines a data event, $c = (c_0, \ldots, c_{M-1})$ is a list of occurrence counters for each facies and $m = (m_0, \ldots, m_{M-1})$ is a list of mean values for the auxiliary variable associated to the occurrences of the different facies at the central node. More precisely, c_i is the number of data events $d(v)$ equal to d found in the training image with facies i at the reference node v, and m_i is the mean of the auxiliary variable at these nodes v.

6.3 Simulation Accounting for an Auxiliary Variable

Before starting the simulation and building the list, the auxiliary variable t is normalized in the interval $[0, 1]$, via the linear transformation $[a, b] \to [0, 1]$, $t \mapsto (t - a)/(b - a)$, where a and b are the minimum and the maximum of the auxiliary variable $t(v)$, v in $TI_{aux} \cup SG_{aux}$, respectively. To simulate a facies at a node u

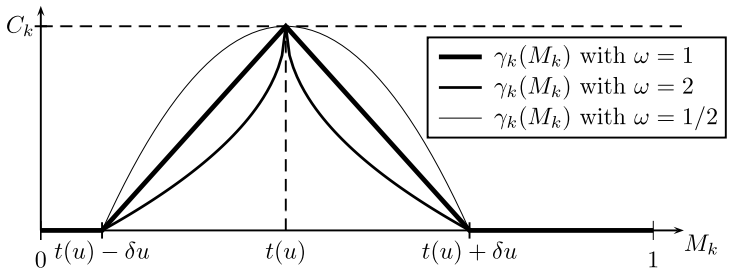

Fig. 9 Selection and penalization with an auxiliary variable; $t(u)$ denotes the value of the auxiliary variable at u in the simulation grid, δu is the tolerance error on the auxiliary variable, C_k is the number of replicates of the data event $d(u)$ found in the training image, for which the value of the auxiliary variable at the central node is in $[t(u) - \delta u, t(u) + \delta u]$, M_k is the mean of the values of the auxiliary variable at the central node of these replicates, and ω is a positive power controlling the penalization

of the grid SG, knowing the data event $d(u)$ and the auxiliary variable $t(u)$ (which is known because SG_{aux} is exhaustively informed), we proceed as follows. A tolerance error $\delta u \in [0, 1]$ is fixed. For each facies k, we retain the set E_k of elements in the list that are compatible with the data event $d(u)$ and that satisfy

$$\left|m_k - t(u)\right| \leq \delta u. \tag{21}$$

Then, we compute the sum C_k of the occurrences counter c_k of the elements of E_k and the resulting mean M_k for the auxiliary variable

$$C_k = \sum_{e \in E_k} c_k^{(e)} \quad \text{and} \quad M_k = \frac{1}{C_k} \sum_{e \in E_k} c_k^{(e)} \cdot m_k^{(e)}, \tag{22}$$

where the exponent (e) denotes the corresponding element in the list. The resulting means are used to penalize the counters. For each facies k, the penalized counter γ_k is defined as

$$\gamma_k = \left(1 - \left(\frac{|M_k - t(u)|}{\delta u}\right)^{\omega^{-1}}\right) \cdot C_k, \tag{23}$$

where ω is a weight (positive real parameter) controlling the penalization (Fig. 9). Then, the conditional PDF knowing the data event $d(u)$ and the auxiliary variable $t(u)$, used to draw the facies at the node u, is given by

$$\mathbb{P}\big(s(u) = k \mid d(u), t(u)\big) = \frac{\gamma_k}{\gamma}, \quad 0 \leq k < M, \tag{24}$$

where $\gamma = \sum_{k=0}^{M-1} \gamma_k$.

In summary, the presence of an auxiliary variable adds a step of selection (21) and a step of penalization (23) to the process of retrieving the conditional PDF. The method requires two parameters, δu and ω, which could depend on location u, while also making it possible to guide the simulation according to the auxiliary variable, as shown in Fig. 9. Note that in practice, the criterion of minimal number of replicates

is applied on the sum C_k of the counters before the penalization and that the CPDF is retrieved in the same manner as in the previous situation, namely with a stationary training image (Sect. 2.4). If all nodes in the data event must be dropped (i.e., $C^{(1)} = \sum_{k=0}^{M-1} C_k^{(1)} < C_{min}$), all the elements of the list are scanned and the counters for each facies are retrieved using only (21) as a criterion for matching elements. If the minimal number of replicates is still not reached, the counters are computed considering all elements of the list, and the penalization is then done. In this last case, the marginal PDF, conditioned to the auxiliary variable, is used

$$\mathbb{P}\big(s(u) = k \mid t(u)\big) = \frac{\gamma_k^{(marg)}}{\gamma^{(marg)}}, \quad 0 \le k < M, \tag{25}$$

where

$$\gamma_k^{(marg)} = \left(1 - \left|M_k^{(marg)} - t(u)\right|^{\min(\delta u, \omega^{-1})}\right) \cdot C_k^{(marg)} \tag{26}$$

and

$$\gamma^{(marg)} = \sum_{k=0}^{M-1} \gamma_k^{(marg)}, \tag{27}$$

with $C_k^{(marg)}$ being the number of nodes in the training image with the facies k and $M_k^{(marg)}$ being the mean of the auxiliary variable at these nodes. For this case only, the power ω^{-1} is replaced by $\min(\delta u, \omega^{-1})$ to account for the parameter δu.

Note that this method can easily be generalized to apply to cases with several auxiliary variables. In the presence of n auxiliary variables, n conditions (one per auxiliary variable) of type (21) are considered, and the factor of penalization in (23) is replaced by a product of n factors (one per auxiliary variable). Both parameters δu (tolerance error) and ω (weight) can be different for each auxiliary variable.

7 Case Studies

7.1 Three-dimensional Application with Two Auxiliary Variables

We consider a nonstationary training image (Fig. 10) representing a turbidite reservoir (Strebelle 2002). Two auxiliary variables are used t describe the nonstationarity of this image: the x-coordinate and the z-coordinate of the nodes. The first one describes a change in the type of structures present in the training image, whereas the second one corresponds to a variation of proportion. A simulation grid of the same dimensions as those in Strebelle (2002) is considered. The same auxiliary variable maps for the training image and the simulation grid are used. One of the resulting realizations is displayed in Fig. 11. Three levels of multigrid are used with a search template with sizes $N = 514, 304, 56$ for coarse, middle, and fine multigrid, respectively. Both auxiliary variables are controlled with the parameters $\delta u = 0.25, 1.00, 1.00$ and $\omega = 1.00, 3.00, 3.00$ for coarse, middle, and fine multigrid, respectively. No post- or syn-processing method was applied.

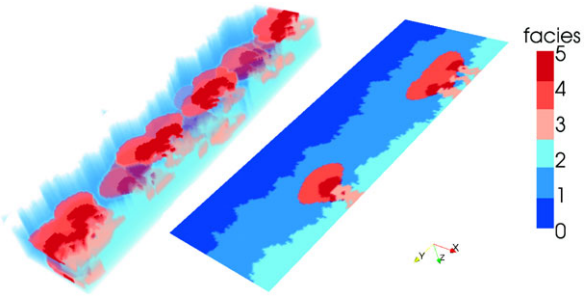

Fig. 10 Three-dimensional training image of dimensions $70 \times 183 \times 20$, with 6 facies representing turbidite reservoir (Strebelle 2002) (three-dimensional view and one xy slice)

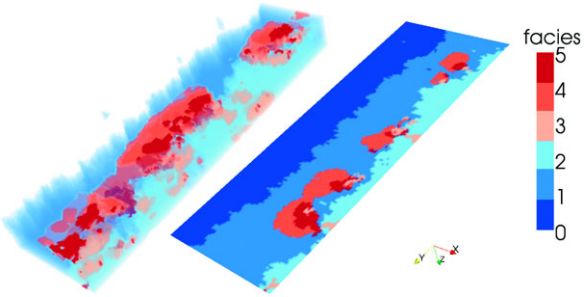

Fig. 11 One realization of dimensions $70 \times 183 \times 20$ obtained with a multiple–point statistics simulation algorithm based on lists, using the training image in Fig. 10 and 2 auxiliary variables (x and z coordinates)

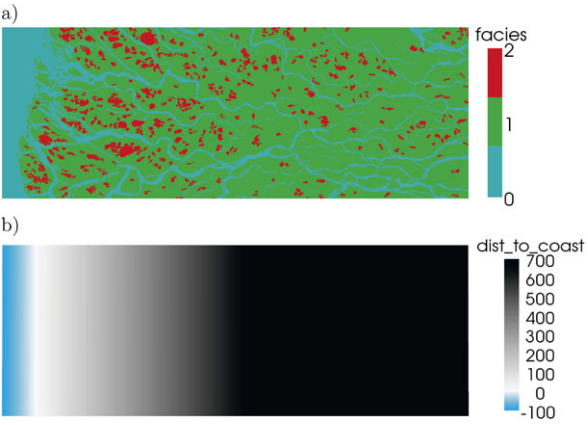

Fig. 12 Reference images of dimensions 1501×501: (**a**) two-dimensional training image with 3 facies (Lena River delta, *Landsat 7 image, USGS/EROS and NASA Landsat Project*); and (**b**) auxiliary variable (distance to the coast, bounded between -100 and 700)

7.2 Two-dimensional Application with One Auxiliary Variable and Rotations

The two-dimensional training image shown in Fig. 12(a) is used. The image represents a part of the Lena River delta and contains 3 facies: value 0 for the sea and rivers, value 1 for land, and value 2 for lakes. Although both rivers and lakes are water bodies, they are represented by different facies because they are structurally different. The non-stationarity of this training image is described by one auxiliary variable, that is, the distance to the coast (Fig. 12(b)). The distance is negative for nodes in the sea and positive for inland nodes, and it is bounded between -100 and 700 (unit: one pixel).

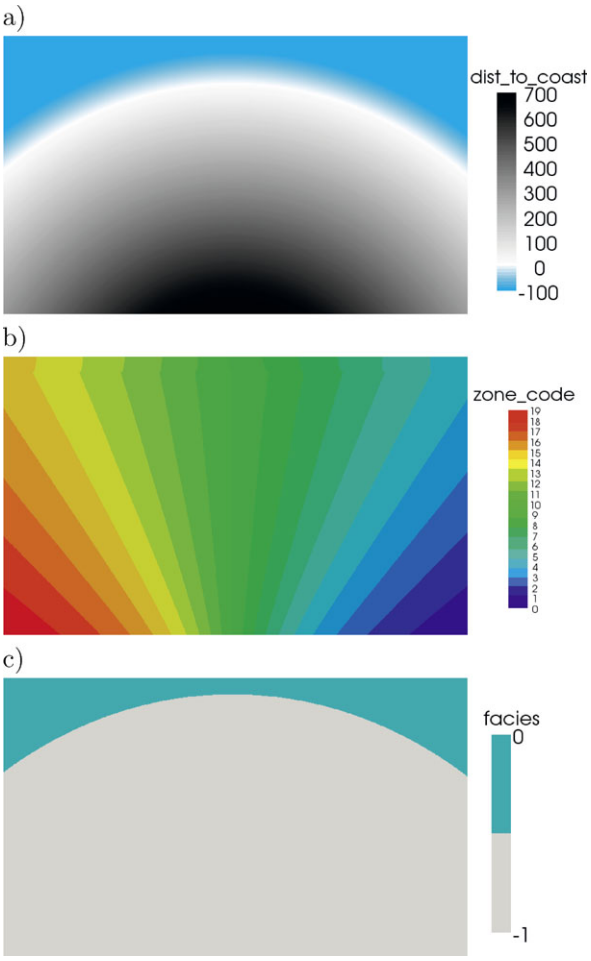

Fig. 13 Input maps for the simulation, dimensions 1500×900: (**a**) auxiliary variable (distance to the coast, bounded between -100 and 700); (**b**) zone maps (20 zones for rotations); and (**c**) conditioning hard data (value -1 for uninformed nodes)

In addition to the nonstationarity expressed by the distance to the coast, we propose to integrate the nonstationarity provided by rotations. This is done using zones of simulation. In each zone, the angle (azimuth) of rotation to be applied to the training image is given. Thus, two maps are required for the simulation grid: an auxiliary variable map (distance to the coast, Fig. 13(a)) and a zone map for rotations (Fig. 13(b)). Moreover, the nodes in the simulation grid that are in the sea and whose distance to the coast is at least 100 (unit: one pixel) are considered as conditioning hard data (Fig. 13(c)). One realization obtained with the proposed algorithm is shown in Fig. 14. Three levels of multigrid are used with search templates with sizes of $N = 120$, 96, 56 for coarse, middle, and fine multigrid, respectively. The parameters controlling the auxiliary variable were set to $\delta u = 1.00$ and $\omega = 2.00$ for each multigrid. No post- or syn-processing method was applied.

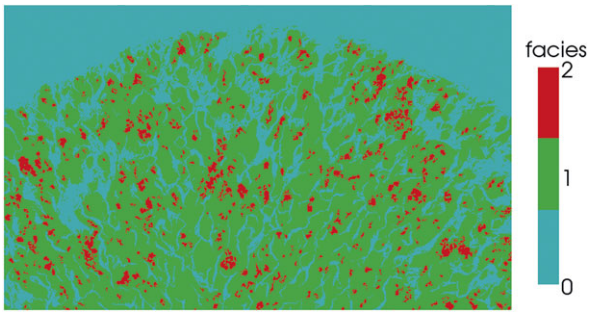

Fig. 14 One realization of dimensions 1500×900 obtained with a multiple-point statistics simulation algorithm based on lists, using the training image shown in Fig. 12(a), one auxiliary variable (Fig. 12(a) for the training image and Fig. 13(a) for the simulation grid), 20 zones for rotation (Fig. 13(b)) and some conditioning hard data (Fig. 13(c))

8 Conclusions

The proposed new algorithm for performing multiple-point statistics simulations overcomes some of the limitations of previous implementations and enlarges the spectrum of possible applications amenable to the method. The resulting parallel code, *impala*, is implemented in ANSI C. In simple cases using a stationary training image, the results obtained by *snesim* (Strebelle 2002) and *impala* are almost identical. The few differences that emerge can be attributed to the way of scanning the border of the training image and to the way of dealing with conditioning data. However, they are insignificant when comparing the resulting simulations. The main differences compared to other methods are, first, the use of lists instead of search trees to store the statistics inferred from the training image and, second, the parallelization of the algorithm. Using lists allows one to significantly reduce the amount of memory (RAM) required by the algorithm and to parallelize the code straightforwardly.

The reduction of memory usage has profound implications for the applications of multiple-point statistics: first, it allows one to deal with realistic applications on large three-dimensional grids with a high number of categories. Second, it allows one to deal with non-stationary training image. Indeed, the memory is a limiting factor in such a situation, as the conclusions of de Vries et al. (2009) suggest. The lists can be extended to include information on secondary variables. Third, it allows one to deal with nonstationarity through rotations and/or homothetic transformations. In such a situation, the simulation grid is generally divided up into zones and a unique transformation is considered for each zone. A different list is then required for each simulation zone (and each multigrid level). Since the memory load required by a list is low, numerous zones can be considered.

In addition to these advantages, the simple structure of a list makes it possible to obtain a parallel code straightforwardly. Indeed, in a parallel environment, the lists are split into as many sub-lists as the number of available processors. This offers a direct and simple way to parallelize the computation of the CPDF, while also making the proposed method particularly efficient on multicore computers. This parallelization method is potentially more scalable on very large machines than were previous approaches, which mainly considered the parallelization of the simulation of simultaneous non-interacting pixels (sufficiently distant so that the simulation of one pixel should not influence the simulation of the other).

Finally, because the core of the proposed methodology remains based on the classical multiple-point statistics framework proposed by Strebelle (2002), all the usual extensions of the method (post- or syn-processing, affinities, rotations, simulation by zones, etc.) can be applied straightforwardly.

Acknowledgements Funding for this research was provided by the Swiss Confederation's Innovation Promotion Agency (CTI project No. 8836.1 PFES–ES) and the Swiss National Science Foundation (grant No. PPOOP2–106557). We thank Alessandro Comunian and Alexandre Walgenwitz for their help in developing and testing the algorithms, and Pierre Biver, Tatiana Chugunova, and Sébastien Strebelle for providing data, as well as many stimulating discussions. We are also grateful to Philip Brunner for his helpful comments, and to Marita Stien as a reviewer for her relevant remarks.

References

Arpat G, Caers J (2007) Conditional simulation with patterns. Math Geol 39(2):177–203
Caers J (2005) Petroleum geostatistics. Society of Petroleum Engineers, Richardson
Caers J, Strebelle S, Payrazyan K (2003) Stochastic integration of seismic data and geologic scenarios: a west Africa submarine channel saga. Lead Edge 22(3):192–196
Chugunova T, Hu L (2008) Multiple-point statistical simulations constrained by continuous auxiliary data. Math Geosci 40(2):133–146
Daly C, Caers J (2010) Multipoint geostatistics—an introductory review. First Break 28(9). doi:10.3997/1365-2397.2010020
Daly C, Knudby C (2007) Multipoint statistics in reservoir modelling and in computer vision. In: Petroleum geostatistics 2007, EAGE, Cascais, Portugal
de Vries LM, Carrera J, Falivene O, Gratacos O, Luit JS (2009) Application of multiple point geostatistics to non-stationary images. Math Geosci 41(1):29–42
Dongarra J, Huss-Lederman S, Otto S, Snir M, Walker D (1998) MPI—the complete reference: the MPI core, 2nd edn, vol. 1. MIT Press, Cambridge
Gropp W, Huss-Lederman S, Lumsdaine A, Lusk E, Nitzberg B, Saphir W, Snir M (1998) MPI—the complete reference: the MPI extensions, vol 2. MIT Press, Cambridge
Guardiano F, Strivastava R (1993) Multivariate geostatistics: beyond bivariate moments. In: Soares A (ed) Geostatistics Troia, vol 1. Kluwer Academic, Dordrecht, pp 133–144
Hu L, Chugunova T (2008) Multiple-point geostatistics for modeling subsurface heterogeneity: a comprehensive review. Water Resour Res 44:W11413
Journel A, Zhang T (2006) Necessity of a multiple-point prior model. Math Geol 38(5):591–610
Liu Y (2006) Using the snesim program for multiple-point statistical simulation. Comput Geosci 32:1544–1563
Liu Y, Harding A, Abriel W, Strebelle S (2004) Multiple-point simulation integrating wells, three-dimensional seismic data, and geology. Am Assoc Pet Geol Bull 88(7):905–921
Mariethoz G, Renard P, Straubhaar J (2009) The direct sampling method to perform multiple-points geostatistical simulations. Water Resour Res (submitted)
Okabe H, Blunt MJ (2007) Pore space reconstruction of vuggy carbonates using microtomography and multiple-point statistics. Water Resour Res 43:W12S02
Remy N, Boucher A, Wu J (2009) Applied geostatistics with SGeMS: a user's guide. Cambridge University Press, New York
Renard P (2007) Stochastic hydrogeology: what professionals really need? Ground Water 45(5):531–541
Rivoirard J, Cojan I, Renard D, Geffroy F (2008) Advances in quantification of process–based models for meandering channelized reservoirs. In: Ortiz J, Emery X (eds) VIII international geostatistics congress, GEOSTATS 2008, Santiago, Chile
Ronayne M, Gorelick S, Caers J (2008) Identifying discrete geologic structures that produce anomalous hydraulic response: an inverse modeling approach. Water Resour Res 44:8
Stien M, Hauge R, Kolbjørnsen O, Abrahamsen P (2007) Modification of the snesim algorithm. In: Petroleum geostatistics 2007, EAGE, Cascais, Portugal
Strebelle S (2002) Conditional simulation of complex geological structures using multiple-points statistics. Math Geol 34(1):1–21

Strebelle S, Remy N (2005) Post-processing of multiple-point geostatistical models to improve reproduction of training patterns. In: Leuangthong O, Deutsch C (eds) Geostatistics Banff 2004. Springer, Berlin, pp 979–988

Suzuki S, Strebelle S (2007) Real–time post–processing method to enhance multiple-point statistics simulation. In: Petroleum geostatistics 2007, EAGE, Cascais, Portugal

Tran TT (1994) Improving variogram reproduction on dense simulation grids. Comput Geosci 20(7):1161–1168

Vargas H, Caetano H, Mata-Lima H (2008) A new parallelization approach for sequential simulation. In: Soares A, Pereira MJ, Dimitrakopoulos R (eds) geoENV VI geostatistics for environmental applications. Springer, Berlin, pp 489–496

Wu J, Zhang T, Journel A (2008) Fast filtersim simulation with score-based distance. Math Geosci 40(7):773–788

Zhang T, Switzer P, Journel AG (2006) Filter-based classification of training image patterns for spatial simulation. Math Geol 38(1):63–80

Zhang T, Pedersen SI, McCormick D (2008) Patched path and recursive servo system in multiple-point geostatistics simulation. In: Proceedings of the eighth international geostatistics congress, vol 2, pp 1119–1124. Gecamin, Santiago