Training Images-Based Stochastic Simulation on Many-Core Architectures

Tao Huang and Detang Lu

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/64276

Abstract

In the past decades, multiple-point geostatistical methods (MPS) are increasing in popularity in various fields. Compared with the traditional techniques, MPS techniques have the ability to characterize geological reality that commonly has complex structures such as curvilinear and long-range channels by using high-order statistics for pattern reconstruction. As a result, the computational burden is heavy, and sometimes, the current algorithms are unable to be applied to large-scale simulations. With the continuous development of hardware architectures, the parallelism implementation of MPS methods is an alternative to improve the performance. In this chapter, we overview the basic elements for MPS methods and provide several parallel strategies on many-core architectures. The GPU-based parallel implementation of two efficient MPS methods known as SNESIM and Direct Sampling is detailed as examples.

Keywords: geostatistics, multiple point, stochastic simulation, training image, many-core architecture

1. Introduction

Geostatistical stochastic simulation is important for the research of geological phenomenon. In the past several decades, a large number of geostatistical methods have been developed based on the spatial covariance properties of the geological data. The traditional tool to quantify the spatial covariance is known as variogram, which measures the covariance among any two points separated by a certain distance [1]. Although variogram-based methods are successfully applied to multi-Gaussian system, they have limitations for the characterization of complex systems such as the curvilinear or long-range continuous facies [2–4]. An alternative known as multiple-point geostatistical (MPS) simulation was proposed to produce geologically realistic structure by using high-order spatial statistics based on conceptual training images [5, 6]. The concept of training image is introduced from explicit to represent geological structures with a numeral image generated from outcrops, expert sketching, or conditional simulations of variogram-based methods. Since the training images can incorporate additional information such as the expert guesses, prior database, and physical models, besides the spatial features, the simulation using TIs is straightforward and smart [7].

Due to the ability of reconstructing geological realistic, MPS methods are gaining popularity, and various algorithms have been proposed including pixel-based algorithms [5, 8, 9], pattern-based algorithms [10–13], and optimal algorithms [14–16]. These algorithms have been applied to broad fields such as oil and gas industry [17–19], fluid prediction [20, 21], climate modeling [22]. However, some application suffers from the computational burden routinely. Since MPS methods need to scan the training image, abstract patterns, and reconstruct the patterns in the simulation grid, physical memory and running time are challenging or even unusable for large-scale or pattern-rich simulation models.

Many effects have been made to decrease the central processing units (CUP) and RAM expense. Approaches such as multiple grid technique [23], optimization of data storage with a list structure [24], hierarchical decomposing [25], Fourier space transform [26] have been introduced, while the computational burden remains heavy for very large grids and complex spatial models.

With the development of hardware, utilization of multiple-core central processing units (CPU), or graphic processing units (GPU), for parallel applications are increasing in popularity in various fields including geostatistical simulation. In 2010, Mariethoz investigated the possibility to parallelize the MPS simulation process on realization level, path-level, or node-level and proposed a general conflict management strategy [27]. This strategy has been implemented on a patch-based SIMPAT method [28]. Parallel implements for other geostatistical algorithms, such as the parallel two-point geostatistical simulation [29], parallel pixel-based algorithms [30], and parallel optimal algorithms [31] have been proposed constantly.

In this article, we will present the parallel several schemes of MPS simulation on many-core and GPU architectures. The Compute Unified Device Architecture (CUDA) that provides access between CUPs and GPUs is used to illustrate the parallel strategies [32, 33]. Examples of the two general MPS algorithms known as SENSIM and DS are implemented and compared [34, 35] with the original algorithms to present the ability of pattern reproduction and the improvement of computational performance.

2. Methodology

Currently, geostatistical simulations are processed with a large number of MPS algorithms using various techniques. Besides the difference in process definition and algorithmic implementation, these algorithms share similarities of the fundamental elements [36].

2.1. General framework

Generally, the overall framework of these algorithms is constructed as follows:

1. Migrate conditioning points to the corresponding grid nodes.
2. Define data template to abstract or clustering patterns from the training image.
3. Define a simulation path for the nodes to be simulated.
4. Using the conditioning data and previous simulated nodes as priori data to sample from the training image.
5. Place the sample to the simulation grid.
6. Repeat three–five until meeting the stopping criterion.

For each algorithm, one or some of the basic steps may have some identity. In this section, we will focus on the two algorithms following different theory, single normal equation simulation (SNESIM) and DS, to illustrate the application of parallel strategies on MPS simulations.

2.2. SNESIM review

The methodology of single normal equation simulation (SNESIM) is a sequential paradigm developed from the random concepts theory with the property of generating conditional distribution with local conditioning instead of global conditioning. The local conditional distribution is aggregated with the local data event. Considering a category property S with K possible states {s(k), k = 1, …, K}, a data template τ_n is comprised of a geometry of *n* vectors $\{h_\alpha, \alpha = 1, \ldots, n\}$. The data event d_n centered at u is defined with the template and the corresponding *n* values $\{s(u + h_\alpha), \alpha = 1, \ldots, n\}$. Consequently, the conditional probability of the occurrence of state s_k denoted as $\text{Prob}\{S(u) = s_k \mid S(u_\alpha), \alpha = 1 \ldots n\}$ or $\text{Prob}\{S(u) = s_k \mid d_n\}$ is defined following Bayes' theorem:

$$\text{Prob}\{S(u)=s_k|d_n\}=\frac{\text{Prob}\{S(u)=s_k \text{ and } d_n\}}{\text{Prob}\{d_n\}} \tag{1}$$

Where $\text{Prob}\{d_n\}$ and $\text{Prob}\{S(u) = s_k \text{ and } d_n\}$ are the probability of the occurrence of d_n and the associated to a central value $S(u) = s_k$ respectively. In practice, this probability can be obtained by counting the number of replicates of the training image, which is calculated by:

$$\text{Prob}\{S(u)=s_k|d_n\}=\frac{c_k(d_n)}{c(d_n)} \tag{2}$$

$c\{d_n\}$ is the number of replicates of the conditioning data d_n, and $c_k\{d_n\}$ is the number of replicates inferred from $c_k(d_n)$ of which the central node S{u} has a value of s(k).

To reduce the computational burden of scanning the training image, SNESIM analyzes the training image and constructs a search tree to store all the patterns before sequential simulation. Then the same data template is used to aggregate the local distribution of the simulation grid for sampling values from the data base to the simulation grid.

The implementation of SNESIM algorithm is shown in **Figure 1**.

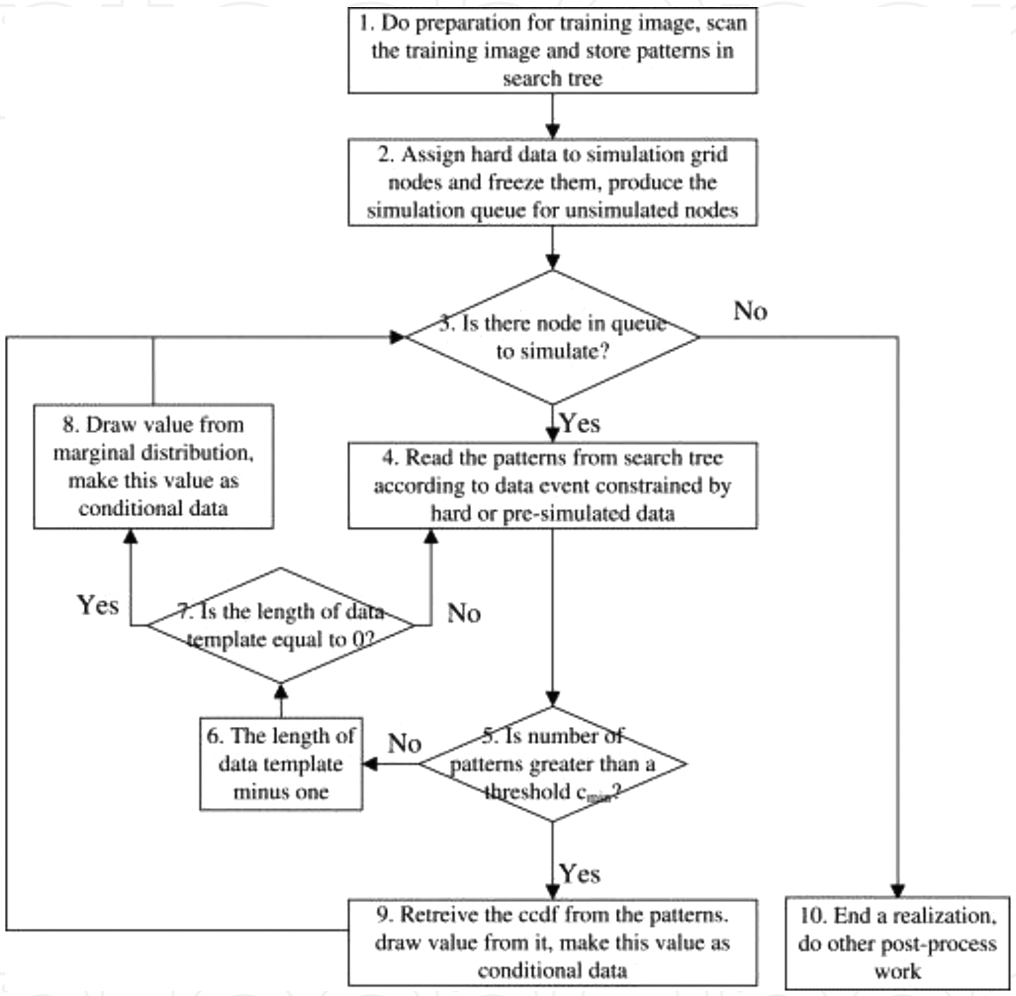
1. Do preparation for training image, scan the training image and store patterns in search tree
2. Assign hard data to simulation grid nodes and freeze them, produce the simulation queue for unsimulated nodes
3. Is there node in queue to simulate?
No
Yes
4. Read the patterns from search tree according to data event constrained by hard or pre-simulated data
5. Is number of patterns greater than a threshold c_min?
No
6. The length of data template minus one
7. Is the length of data template equal to 0?
Yes
No
8. Draw value from marginal distribution, make this value as conditional data
Yes
9. Retreive the ccdf from the patterns, draw value from it, make this value as conditional data
10. End a realization, do other post-process work

Figure 1. Flowchart for SNESIM process.

As described, the memory consumption is largely related to the size of data template and the pattern richness of the training image. At the same time, the pattern retrieving time also increases for the complex cases. To address this problem, a GPU-based parallel method is proposed in the following section.

2.3. Direct Sampling review

Direct Sampling is a novel MPS technique that borrows ideas from a sampling method introduced by Shannon totally abandoning the conditional probability approach. Instead of the conditional distribution, a measurement distance is used to calculate the similarity between the local conditioning and the data event got from the training image along a random path. The distance method enables the application to both categorical and continuous variables. During the sequential simulation, for each node to be simulated in the simulation grid, the training image is scanned along a circular unilateral path [37] and the distance is calculated. As long as the distance is smaller than the defined threshold, the current data event in the training image is sampled, and the central value is assigned to the node to be simulated in the simulation grid directly. The flowchart of Direct Sampling is shown in **Figure 2**.

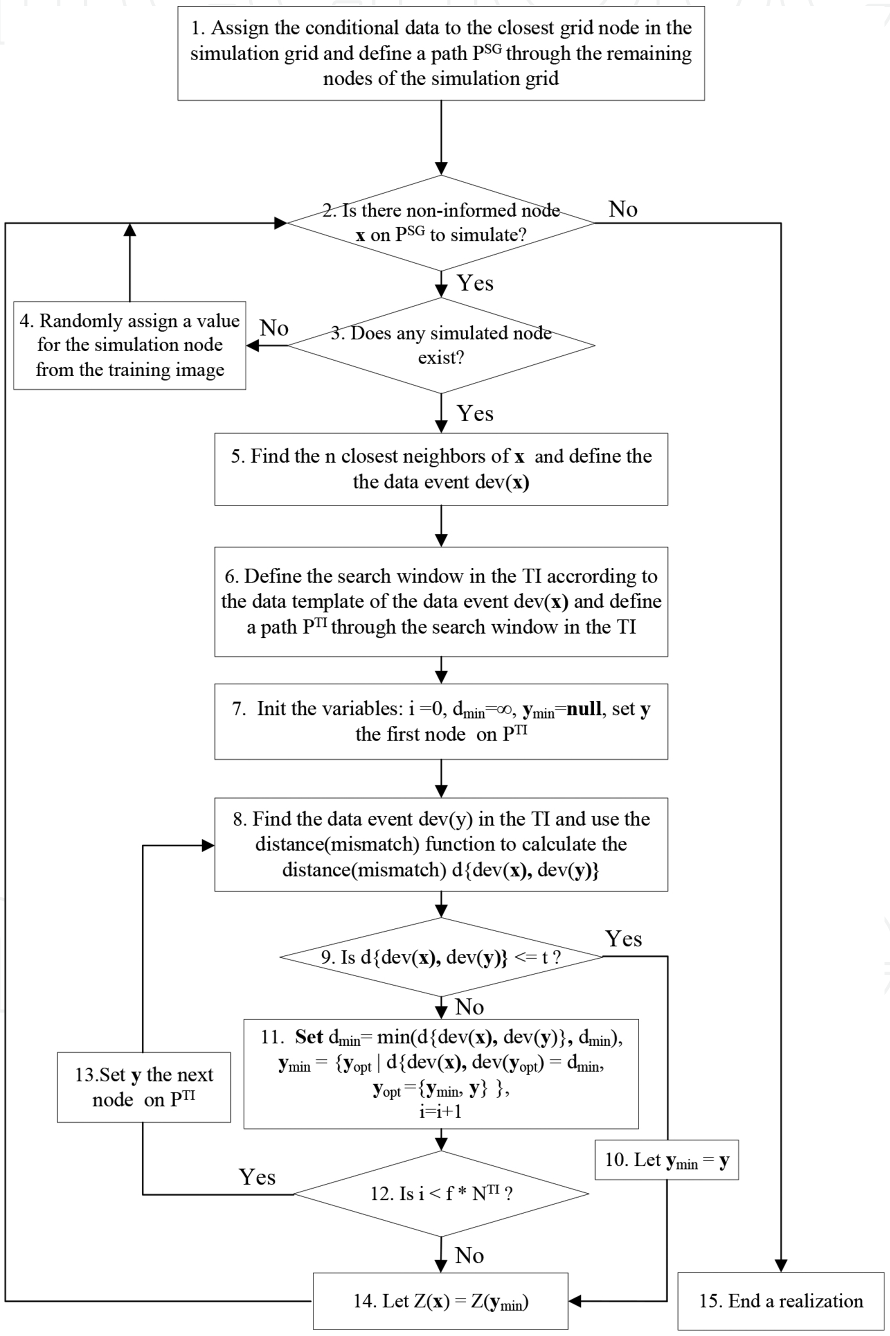
1. Assign the conditional data to the closest grid node in the simulation grid and define a path P^{SG} through the remaining nodes of the simulation grid
2. Is there non-informed node **x** on P^{SG} to simulate?
No
Yes
3. Does any simulated node exist?
No
4. Randomly assign a value for the simulation node from the training image
Yes
5. Find the n closest neighbors of **x** and define the the data event dev(**x**)
6. Define the search window in the TI accrording to the data template of the data event dev(**x**) and define a path P^{TI} through the search window in the TI
7. Init the variables: i =0, $d_{min}=\infty$, $\mathbf{y}_{min}$=**null**, set **y** the first node on P^{TI}
8. Find the data event dev(y) in the TI and use the distance(mismatch) function to calculate the distance(mismatch) d{dev(**x**), dev(**y**)}
9. Is d{dev(**x**), dev(**y**)} <= t ?
Yes
No
11. **Set** d_{min}= min(d{dev(**x**), dev(**y**)}, d_{min}), $\mathbf{y}_{min}$ = {$\mathbf{y}_{opt}$ | d{dev(**x**), dev($\mathbf{y}_{opt}$) = d_{min}, $\mathbf{y}_{opt}$ ={$\mathbf{y}_{min}$, **y**} }, i=i+1
13.Set **y** the next node on P^{TI}
10. Let $\mathbf{y}_{min}$ = **y**
Yes
12. Is i < f * N^{TI} ?
No
14. Let Z(**x**) = Z($\mathbf{y}_{min}$)
15. End a realization

Figure 2. Flowchart for Direct Sampling process.

Since this method does not need to store patterns, it releases the memory intensity; on the other hand, parameter is the key factor for Direct Sampling.

There are three main input parameters:

1. Maximum number of closest neighbors *n*, namely, the size of data template of SNESIM.
2. The distance threshold *t*. A value of $t = 0$ means a trend of verbatim copy of the training and the increasing value introduce more variabilities between realizations in pattern reproduction.
3. The fraction of scanned TI *f*. The fraction is defined to stop the process of scanning the training image while no distances are under the threshold. If the percentage of nodes in the training image reach *f*, the scanning stops and the pattern with lowest distance is sampled.

Sensitivity analysis of parameters [38] shows trade-offs between the quality of realizations and the CPU times.

3. Many-core architectures

3.1. Overview

A computing component that featured with two or more processing units to execute program instructions independently is known as a multicore processor. With the ability of running multiple instructions at the same time, multicore processors increase overall speed for many general-purpose computing. Currently, adding support for more execution threads is the norm avenue to improve the performance of high-end processors. The many-core architectures are formed by manufacturing massive multicores on a single component. For general-purpose parallel computing, many-core architectures on both the central processing unit (CPU) and the graphics processing unit (GPU) are available for different tasks.

Compared with a many-core CPU architecture known as a supercomputer, the general GPU has many more cores, which are constructively cheap and suitable for intensive computing.

3.2. GPU and CUDA

Originally, a GPU is a graphic card attached with a cluster of streaming processors aimed at graphic-oriented details that needs the ability of extremely fast processing of large-volume data sets. To apply the special-purpose GPU to general-purpose application, NVIDIA provides a user-friendly development environment that is Compute Unified Device Architecture (CUDA). The CUDA platform enables the generation of parallel codes on GPUs by driving the process from the CPU to GPU. A CUDA program is a unified code that consists of executions on both the host (CUP) and the device (GPU) by CUDA kernel functions that are called out from the host to the device for asynchronously execution. The massive parallelism is carried out in each kernel on CUDA threads that are the basic executing units on the GPU. The CUDA provides an interface named Peripheral Component Interconnect Express (PCI-e) for the intercommunication between host and device and shared memory for synchronization among the parallel threads. The general architecture of CUDA and CUDA memory is illustrated in **Figures 3** and **4**. More details could be referred from the Guides of NVIDA.

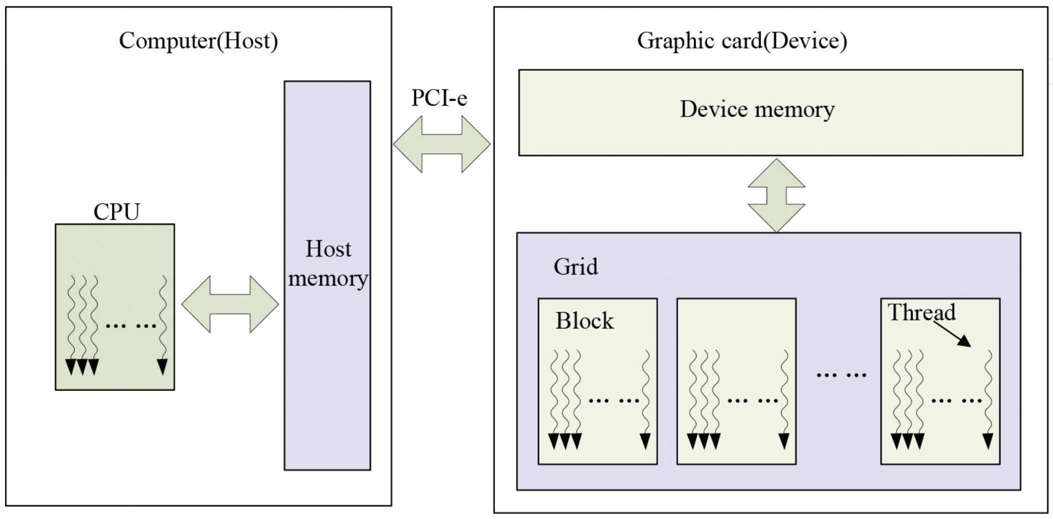

Figure 3. The general architecture of CUDA.

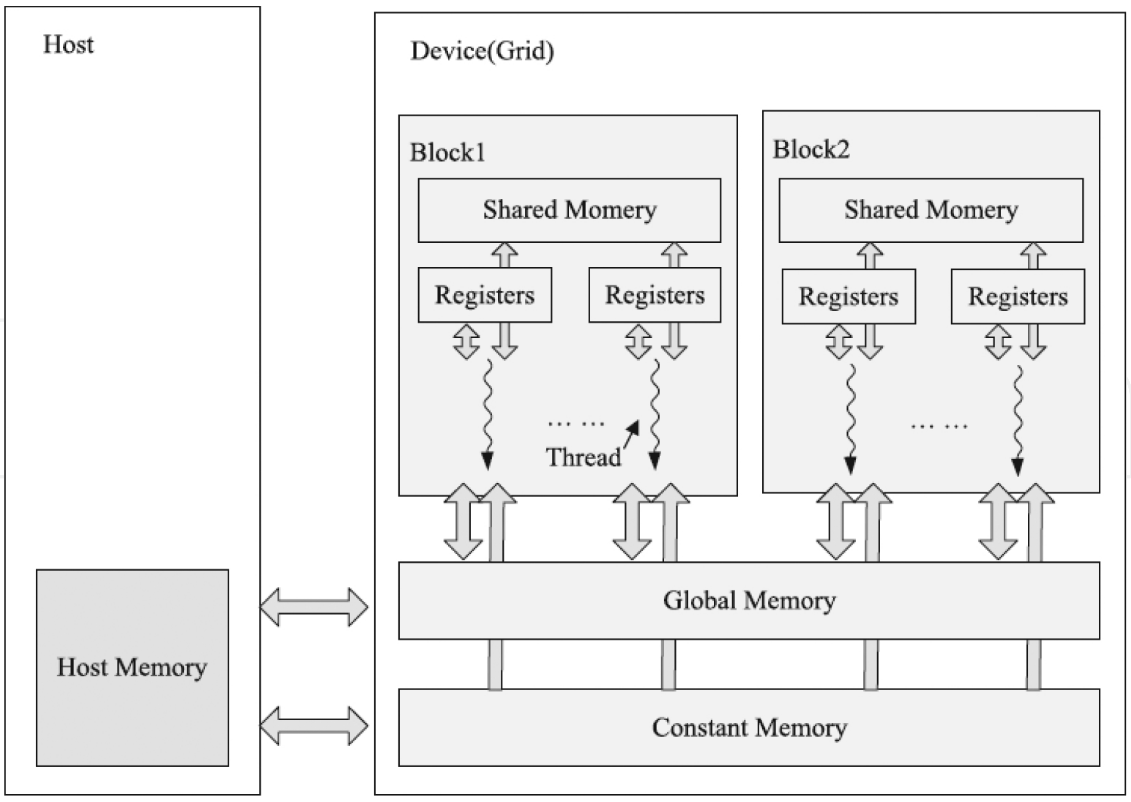

Figure 4. CUDA device memory organization.

4. Parallel strategies

Generally, there are three strategies to implement the parallelism of MPS algorithms, that is, realization-level, path-level, and the node-level parallelization. The bottleneck for large-scale application focuses on the millions' of nodes in the training image and simulation grids. So we present two parallel implementations on node-level for SNESIM and on both node-level and path-level for Direct Sampling in this section.

4.1. GPU-based SNESIM implementation

In 2013, a node-level parallelization was applied to SNESIM algorithm and achieved significant performance improvement [34]. The overall parallel scheme is illustrated in **Figure 5**.

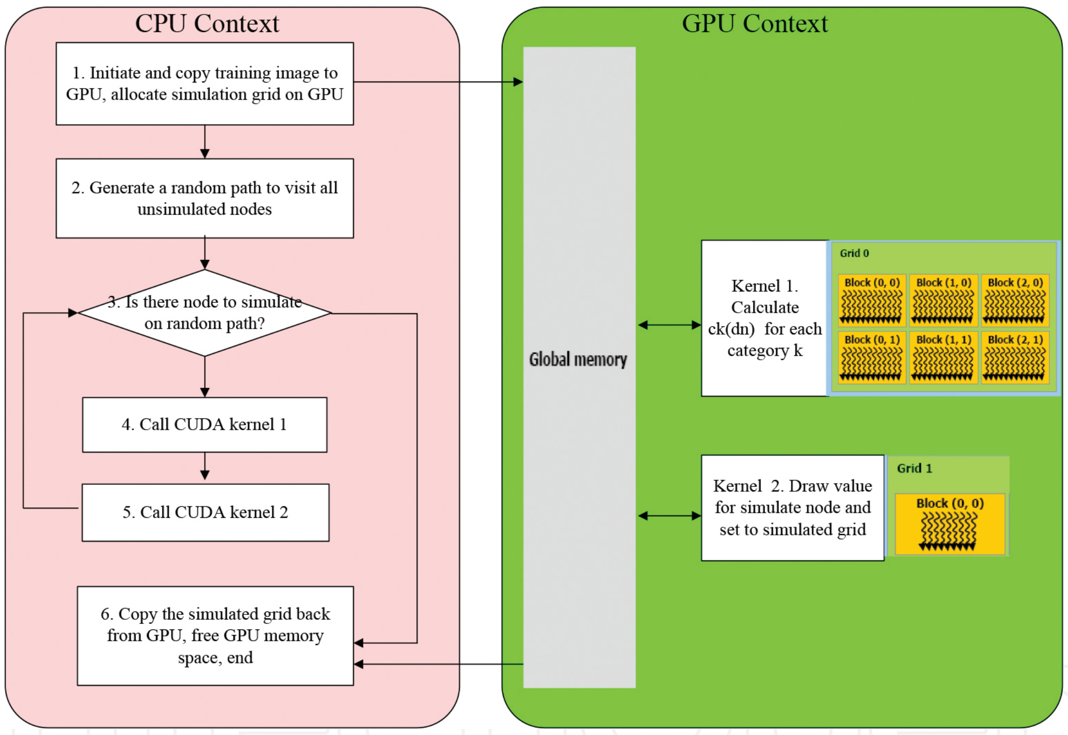
CPU Context
1. Initiate and copy training image to GPU, allocate simulation grid on GPU
2. Generate a random path to visit all unsimulated nodes
3. Is there node to simulate on random path?
4. Call CUDA kernel 1
5. Call CUDA kernel 2
6. Copy the simulated grid back from GPU, free GPU memory space, end
GPU Context
Global memory
Kernel 1. Calculate ck(dn) for each category k
Grid 0
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Kernel 2. Draw value for simulate node and set to simulated grid
Grid 1
Block (0, 0)

Figure 5. Procedure of GPU-based SNESIM implementation.

By transferring the training image and simulation grid from the CPU to GPU, each node is assigned to a CUDA thread. For each unsimulated node along the random simulation path, Kernel 1 compares each value of the given data template d_n with every central nodes in the training image simultaneously and returns the number of replicates $c_k(d_n)$ for each stage k. With these outputs, Kernel 2 calculates the conditional cumulative distribution function (CCDF) and uses Monte Carlo sampling to draw a value to this node. Repeat the kernels along a random path until all the nodes in the simulation grid are simulated. Transfer the results from GPU to CPU.

Since the most time-consuming part, that is getting data event for each node, is parallelized in Kernel 1, this GPU-based implementation gains significant speedup. Moreover, in contrast with the increasing physical memory demanding along the template sizes of original implementation, the proposed GPU implementation fixes the amount of memory.

4.2. GPU-based Direct Sampling implementation

Similar parallelization could also be applied to the Direct Sampling algorithm. Besides the parallelism of the data template, the searching fraction could also be parallelized. The two-stage parallel scheme was implemented in 2013 [35]. The procedure of the GPU-based implementation is illustrated in **Figure 6**.

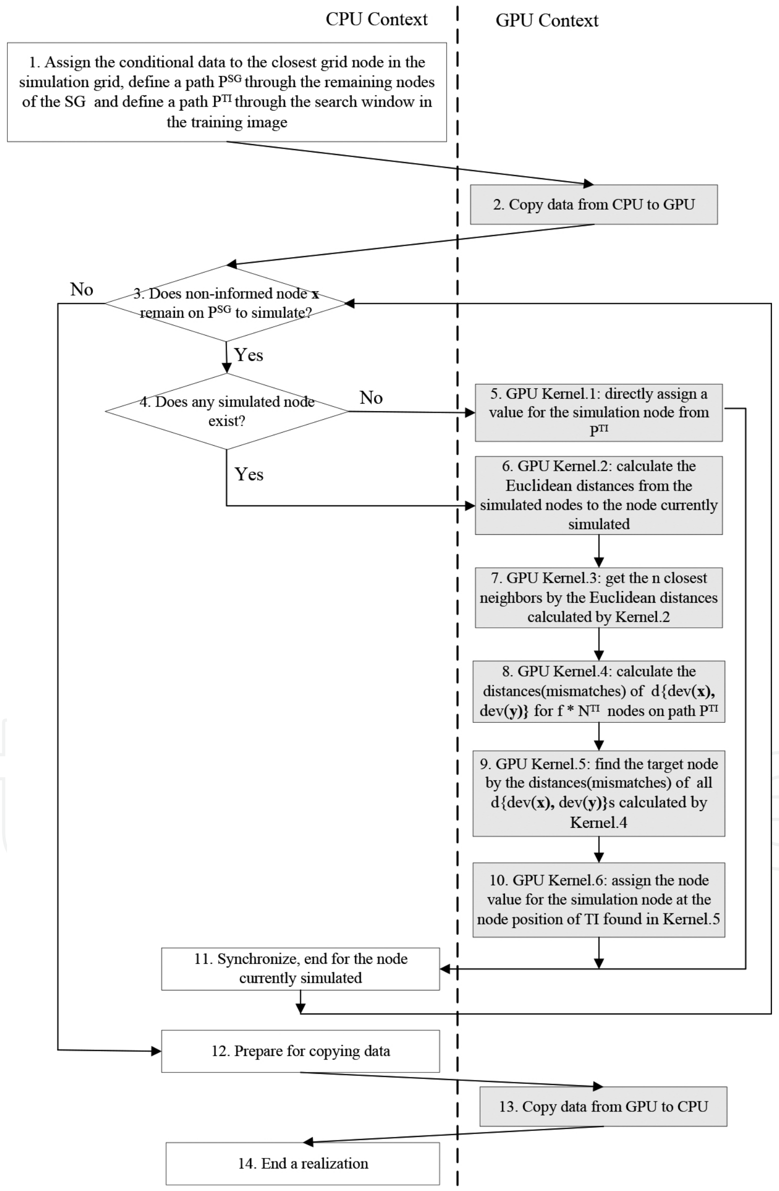
CPU Context
GPU Context
1. Assign the conditional data to the closest grid node in the simulation grid, define a path P^{SG} through the remaining nodes of the SG and define a path P^{TI} through the search window in the training image
2. Copy data from CPU to GPU
3. Does non-informed node **x** remain on P^{SG} to simulate?
No
Yes
4. Does any simulated node exist?
No
Yes
5. GPU Kernel.1: directly assign a value for the simulation node from P^{TI}
6. GPU Kernel.2: calculate the Euclidean distances from the simulated nodes to the node currently simulated
7. GPU Kernel.3: get the n closest neighbors by the Euclidean distances calculated by Kernel.2
8. GPU Kernel.4: calculate the distances(mismatches) of d{dev(**x**), dev(**y**)} for f * N^{TI} nodes on path P^{TI}
9. GPU Kernel.5: find the target node by the distances(mismatches) of all d{dev(**x**), dev(**y**)}s calculated by Kernel.4
10. GPU Kernel.6: assign the node value for the simulation node at the node position of TI found in Kernel.5
11. Synchronize, end for the node currently simulated
12. Prepare for copying data
13. Copy data from GPU to CPU
14. End a realization

Figure 6. Procedure of GPU-based Direct Sampling implementation.

As described, there are three important parameters controlling the simulation of Direct Sampling, that is *t*, *n*, and *f*. This scheme implements the parallelism on two of the three parameters that is *n* and *f*.

4.2.1. Parallelism of n

The number of neighbor is the *n* closest previously simulated nodes to define a data template. Different from the data template used in SNESIM that is controlled by the predefined geometry, the data templates consisted of the *n* closest simulation nodes and have the flexibility of adaption shape and searching range. Due to these flexibilities, this approach can directly catch large-scale structures and easily condition to hard data. On the other hand, the searching for the *n* neighbors is also the most time-consuming part in the serial implementation.

The parallelism of *n* is achieved in Kernel 2 and Kernel 3. In Kernel 2, each previously simulated node is allocated to a CUDA thread and calculates the Euclidean distance to the node to be simulated simultaneously. These distances are transferred to Kernel 3 and sorted using a parallel sorting algorithm.

4.2.2. Parallelism of f

In the serial program if the similarity-distance between data events of the simulation grid and the one sampled from the current training image node is higher than the threshold, a new central node will be sampled from the training image along a random path. Most nodes of the training image that may be visited is defined as $f \times N^{TI}$, as a result, large amount of data event will be sampled in large-scale 3-days simulations.

The parallelization of *f* is implemented in Kernel 4 that allocates $f \times N^{TI}$ threads to $f \times N^{TI}$ central nodes in the training image using a unique random path denoting each node. Thus these similarity distances are calculated simultaneously. There are two possibilities for the similarity distance and the data sampling strategy is shown as follows:

1. There are values lower than the threshold *t*, choose the one that has the smallest path index from the data events with a lower distance.
2. There are no values lower than the threshold *t*, choose the one that has the smallest path index from the data events with the lowest distance.

Finally, the central value of the chosen data event is assigned to the simulation grid by Kernel 6. Repeat all these kernels until all the nodes in the simulation grid are simulated.

5. Experiments

In this section, the performance of the GPU-based implementations is compared with that obtained using SGeMS software [39]. A computer with 4 GB main memory, Interal Core i3540 3.07 GHz CPU, and NVIDIA GeForce GTX 680 GPU that contains eight streaming multiprocessors with 192 CUDA Cores/MP and 2 GB device memory is used for the simulation. The programming platform is the NVIDIA driver version 301.42 with CUDA version 4.2 implemented on VS2010 using C++.

To test the performance of the GPU-based SNESIM algorithm, a 2D porous slice image obtained by CT scanning is used as the training image. The 200 × 200 pixels training image and the corresponding histogram for background and the pore are shown in **Figure 7**.

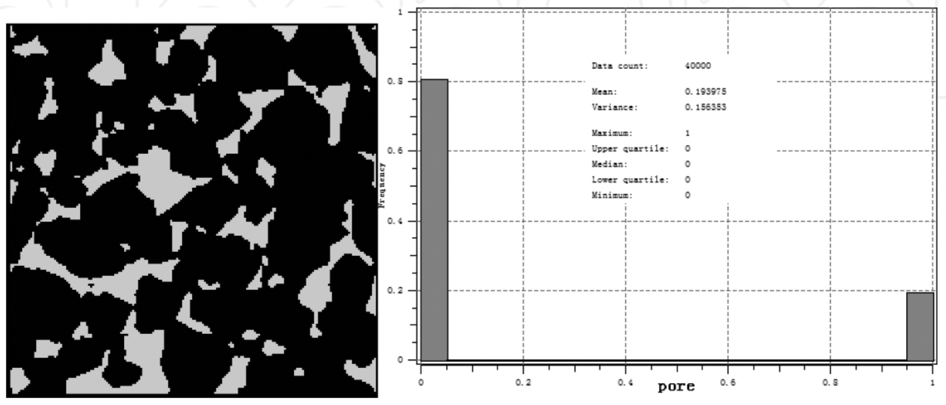

Figure 7. Training image of porous slice and the Histogram.

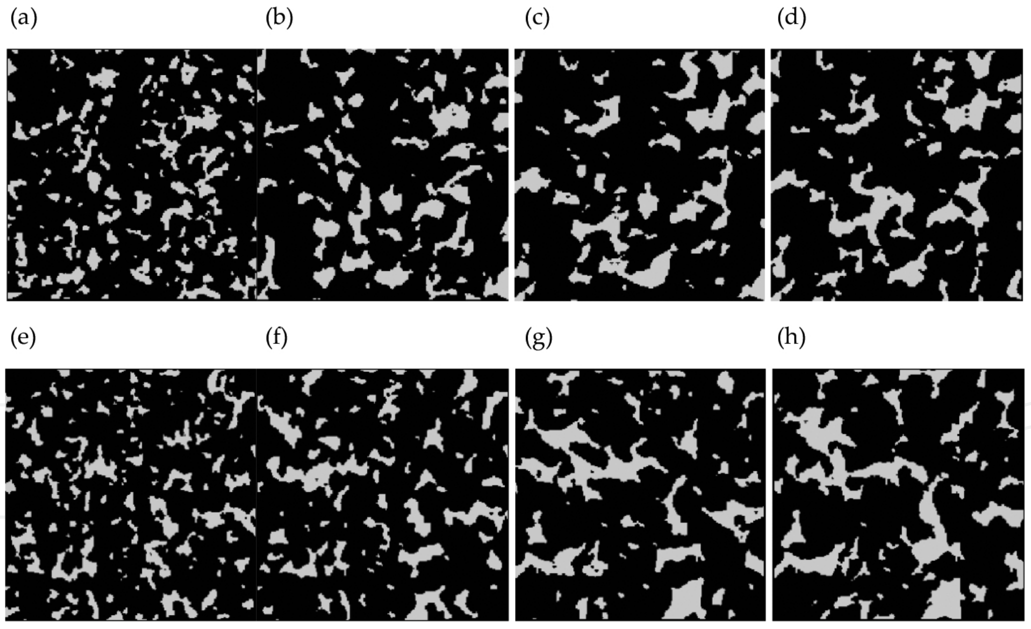

Figure 8. SNESIM realization using CUP and GPU. (a)–(d) CPU-based realizations with the number template equals 50, 120, 200 and 350, respectively; (e)–(h) GPU-based realizations with the number template equals 50, 120, 200 and 350, respectively.

Realizations of the same size as the training image using data template of 50, 120, 200, and 350 nodes are generated for each simulation. The realizations generated with CPU and GPU are shown in **Figure 8**. The average variograms for each simulation are shown in **Figure 9a**, and the performance is shown in **Figure 9b**. The results show that the proposed GPU-based algorithm can generate similar realizations as the original algorithm, whereas significantly increases the performance. The speedup ranges from six to 24 times depending on the template size than a larger template size resulting in a larger speedup.

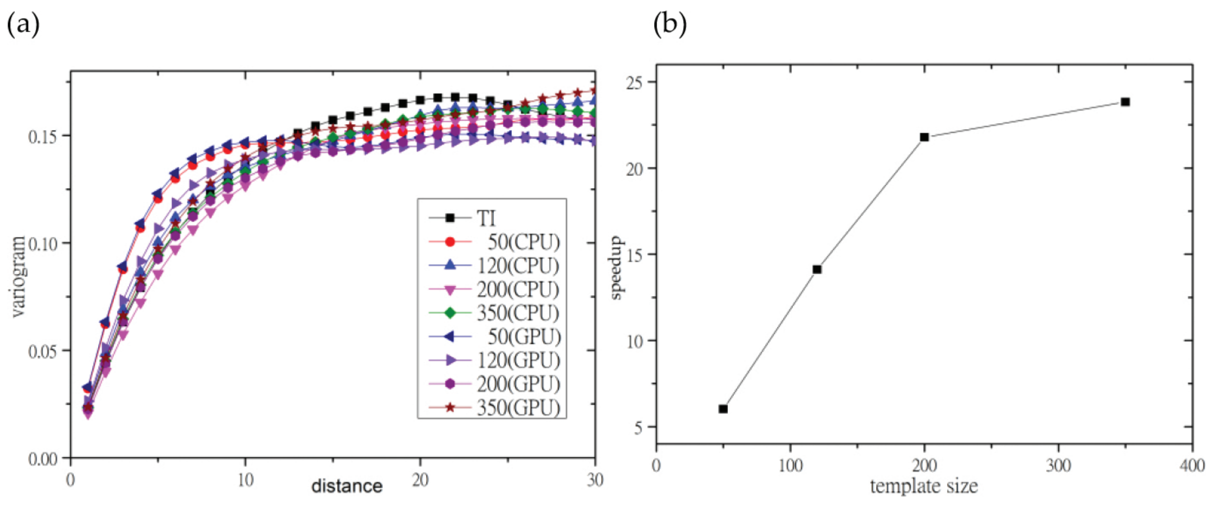

Figure 9. Results and performance comparison between the CPU and GPU implementation. (a) Variogram of the training image and the average variogram of the realizations of **Figure 8** and (b) speedup obtained by using the GPU-based parallel scheme.

The performance of the GPU-based Direct Sampling algorithm is also compared with the original algorithm on a 100 × 130 by 20 fluvial reservoir training image as shown in **Figure 10**.

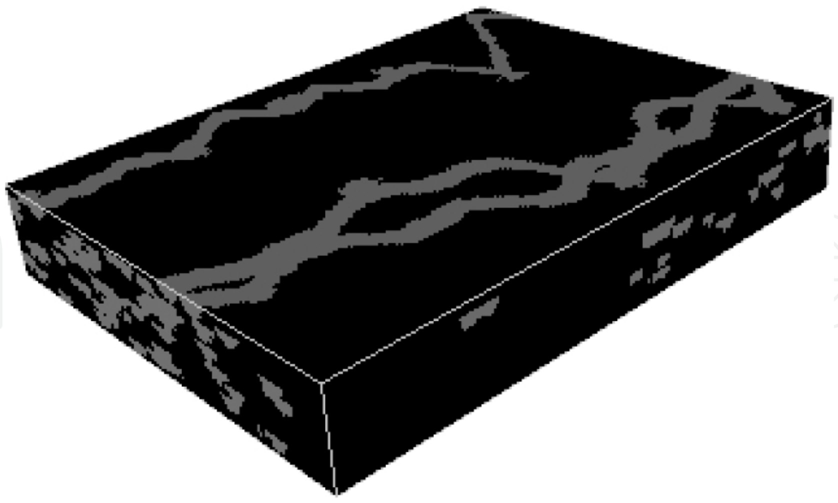

Figure 10. A 100 × 130 by 20 fluvial reservoir training image.

Parameter sensitivities are analyzed on $n = 30, 50, 100, 200$, $f = 0.005, 0.01, 0.02, 0.05$, and $t = 0.01, 0.02, 0.05, 0.1$, respectively, with which reasonable realizations are generated. The performance times are shown in **Figure 11**.

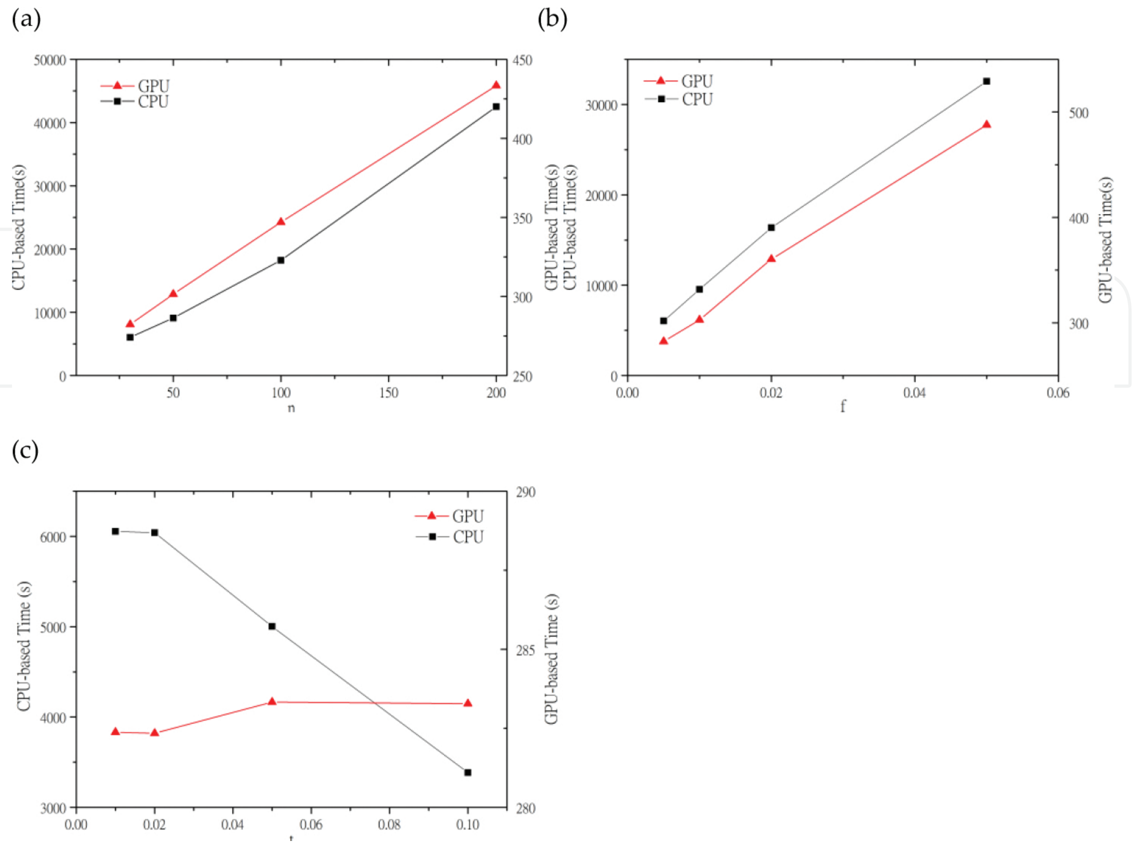

Figure 11. Performance comparison. (a) Performance with fixed $t = 0.2$, $f = 0.005$ and varying $n = 30$, 50, 100, and 200; (b) performance with fixed $n = 30$, $t = 0.02$ and varying $f = 0.005$, 0.01, 0.02, and 0.05; (c) performance with fixed $n = 30$, $f = 0.005$ and varying $t = 0.01$, 0.02, 0.05, and 0.1.

The results show that GPU-based implementation significantly improves the performance for all the tests. Moreover, the sensitivity of parameters to performance is alleviated with the parallel scheme. The time difference is around 200 s for n and f and almost none for t for the GPU-based implementation, whereas it can be as large as several magnitudes for the CPU-based implementation.

In summary, both the presented GPU-based parallel schemes for SNESIM and Direct Sampling achieve significant speedups, especially for large-scale simulations with their node-level parallelism strategy. Moreover, the parallel implementations are insensitive to parameters that are the key points not only for performance but also for simulation results implying for better results in application. These strategies could be further improved with other parallel optimization methods as well. In fact, besides the node-level parallel schemes for SNESIM and Direct Sampling, various parallelisms have been proposed and new optimizations are keeping introduced aimed at further improvements. Up to now, almost all kinds of MPS algorithms could be implemented on a parallel scheme. Many-core architectures are the current mainstreams to improve the performance of extremely massive computing tasks. The developing of computer hardware and parallel interface techniques promise the wider and wider utilization of high-performance parallelization. These parallel schemes of training images-based stochastic simulations approve the application to high-resolution and large-scale simulations.

Acknowledgements

This work was financially supported by the CAS Strategic Priority Research Program under Grant XDB10030402 and the CNPC-CAS Strategic cooperation Research Program (2015A-4812) and the Fundamental Research Funds for the Central Universities (WK2090050038).

Author details

Tao Huang and Detang Lu

*Address all correspondence to: thwang@ustc.edu.cn

University of Science and Technology of China, Hefei, Anhui, China

References

[1] Goovaerts, P. Geostatistical tools for characterizing the spatial variability of microbiological and physico-chemical soil properties. Biology and Fertility of Soils. 1998; 27(4): 315–334.

[2] Journel, A.G., Geostatistics: Roadblocks and Challenges, in Geostatistics Tróia ’92: Volume 1, A. Soares, Editor. 1993, Springer Netherlands: Dordrecht. p. 213–224.

[3] Krishnan, S., Journel, A.G. Spatial connectivity: from variograms to multiple-point measures. Mathematical Geology. 2003;35(8):915–925.

[4] Journel, A.G., Beyond Covariance: The Advent of Multiple-Point Geostatistics, in Geostatistics Banff 2004, O. Leuangthong and C.V. Deutsch, Editors. 2005, Springer Netherlands: Dordrecht. p. 225–233.

[5] Strebelle, S., Journel, A.G. Sequential simulation drawing structures from training images. Geostatistics Capetown 2000. pp. 381–392.

[6] Strebelle, S., Conditional simulation of complex geological structures using multiple–point statistics. Mathematical Geology. 2002;34(1):1–21.

[7] Hu, L., Chugunova, T. Multiple-point geostatistics for modeling subsurface heterogeneity: a comprehensive review. Water Resources Research. 2008;44(11):W11413.

[8] Mariethoz, G., Renard, P., Straubhaar, J., The Direct Sampling method to perform multiple-point geostatistical simulations. Water Resources Research. 2010;46(11):1–14.

[9] Mustapha, H., Dimitrakopoulos, R., HOSIM: a high-order stochastic simulation algorithm for generating three-dimensional complex geological patterns. Computers & Geosciences. 2011;37(9):1242–1253.

[10] Arpat, G.B. Sequential Simulation with Patterns. Ph.D. Dissertation. Stanford [dissertation]. Stanford, CA; 2005. 166 p.

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

[12] Tahmasebi, P., Hezarkhani, A., Sahimi, M. Multiple-point geostatistical modeling based on the cross-correlation functions. Computational Geosciences. 2012;16(3):779–797.

[13] Mahmud, K., Mariethoz, G., Caers, J., Tahmasebi, P., Baker, A. Simulation of Earth textures by conditional image quilting. Water Resources Research. 2014;50(4):3088–3107.

[14] Deutsch, C., Wen, X. Integrating large-scale soft data by simulated annealing. Mathematical Geology. 2000;32(1):49–67.

[15] Caers, J.K., Srinivasan, S., Journel, A.G. Geostatistical quantification of geological information for a fluvial-type North Sea reservoir. SPE Reservoir Evaluation & Engineering. 2000;3(5):457–467.

[16] Srivastava, M. An overview of stochastic methods of reservoir characterization. AAPG Computer Applications in Geology. 1995; Tulsa:3–16.

[17] Okabe, H., Blunt, M.J. Pore space reconstruction using multiple-point statistics. Journal of Petroleum Science and Engineering. 2005;46(1):121–137.

[18] Zhang, T., Bombarde, S., Strebelle, S., Oatney, E. 3D porosity modeling of a carbonate reservoir using continuous multiple-point statistics simulation. SPE Journal. 2006;11(3): 375–379.

[19] Falivene, O., Arbus, P., Gardiner, A., Pickup, G., Muoz, J.A., Cabrera, L. Best practice stochastic facies modeling from a channel-fill turbidite sandstone analog (the Quarry outcrop, Eocene Ainsa basin, northeast Spain). AAPG Bulletin. 2006;90(7):1003–1029.

[20] Hiroshi O, Blunt M J. Pore space reconstruction of vuggy carbonates using microtomography and multiple-point statistics[J]. Water Resources Research, 2007, 431(12): 179–183.

[21] Tahmasebi P., Sahimi M. Cross-correlation function for accurate reconstruction of heterogeneous media. Physical Review Letters. 2013;110(7):078002

[22] Jha, S.K., Mariethoz, G., Evans, J.P., McCabe, M.F. Demonstration of a geostatistical approach to physically consistent downscaling of climate modeling simulations. Water Resources Research. 2013;49(1):245–259.

[23] Strebelle, S. New multiple-point statistics simulation implementation to reduce memory and cpu-time demand. In: Conference of the international association for mathematical geology. September; Portsmouth, UK. 2003.

[24] Straubhaar, J., Renard, P., Mariethoz, G., Froidevaux, R., Besson, O. An improved parallel multiple-point algorithm using a list approach. Mathematical Geosciences. 2011;43(3):305–328.

[25] Maharaja, A., Journel, A.G. Hierarchical Simulation of Multiple-Facies Reservoirs Using Multiple-Point Geostatistics. SPE Annual Technical Conference and Exhibition. Society of Petroleum Engineers, Dallas, Texas. 2005

[26] Tahmasebi, P., Sahimi, M., Caers, J. MS-CCSIM: accelerating pattern-based geostatistical simulation of categorical variables using a multi-scale search in Fourier space. Computers & Geosciences. 2014;67:75–88.

[27] Mariethoz, G. A general parallelization strategy for random path based geostatistical simulation methods. Computers and Geosciences. 2010;36(7):953–958.

[28] Tahmasebi, P., Sahimi, M., Mariethoz, G., Hezarkhani, A. Accelerating geostatistical simulations using graphics processing units (GPU). Computers & Geosciences. 2012;46:51–59.

[29] Nunes, R., Almeida, J.A. Parallelization of sequential Gaussian, indicator and direct simulation algorithms. Computers and Geosciences. 2010;36(8):1042–1052.

[30] Straubhaar, J., Renard, P., Mariethoz, G., Froidevaux, R., Besson, O. An improved parallel multiple-point algorithm using a list approach. Mathematical Geoscience. 2011;43(3):305–328.

[31] Peredo, O., Ortiz, J.M., Parallel implementation of simulated annealing to reproduce multiple-point statistics. Computers and Geosciences. 2011;37(8):1110–1121.

[32] Nvidia, C. C Programming Guide v 4.0. Santa Clara, CA, USA: NVIDIA Corporation; 2011.

[33] Nvidia, CUDA C Best Practices Guide Version 4.0. Technical report. Santa Clara, CA, USA: NVIDIA Corporation; 2009.

[34] Huang, T., Lu, D.T., Li, X., Wang, L. GPU-based SNESIM implementation for multiple-point statistical simulation. Computers & Geosciences. 2013;54:75–87.

[35] Huang, T., Li, X., Zhang, T., Lu, D.T. GPU-accelerated Direct Sampling method for multiple-point statistical simulation. Computers & Geosciences. 2013;57:13–23.

[36] Mariethoz, G., Caers, J. Multiple-point geostatistics: stochastic modeling with training images. John Wiley & Sons, Ltd, Chichester, UK; 2014.

[37] Daly, C. Higher order models using entropy, Markov random fields and sequential simulation. In: Leuangthong, O., Deutsch, C. (Eds). In: Geostatistics Banff 2004; Springer Netherlands; 2005. p. 215–224.

[38] Pirot, G., Meerschman, E., Mariethoz, G., Straubhaar, J., Van Meirvenne, M, Renard, P. Optimizing Direct Sampling algorithm's parameters to performing multiple-points geostatistical simulations. AGU Fall Meeting Abstracts. 2011;1:1475.

[39] Remy, N., Boucher, A., Wu, J., editors. Applied Geostatistics with SGeMS:A User's Guide. Cambridge University Press, Cambridge, UK; 2008.