Computers & Geosciences 46 (2012) 51–59

Contents lists available at SciVerse ScienceDirect

Computers & Geosciences

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

Accelerating geostatistical simulations using graphics processing units (GPU)

Pejman Tahmasebi [a], Muhammad Sahimi [b,*], Grégoire Mariethoz [c], Ardeshir Hezarkhani [a]

[a] Department of Mining, Metallurgy and Petroleum Engineering, Amir Kabir University of Technology, Tehran 15875, Iran
[b] Mork Family Department of Chemical Engineering and Materials Science, University of Southern California, Los Angeles, California 90089-1211, USA
[c] National Center for Groundwater Research and Training, School of Civil and Environmental Engineering, University of New South Wales, NSW 2052 Sydney, Australia

ARTICLE INFO

Article history:
Received 3 January 2012
Received in revised form
28 March 2012
Accepted 29 March 2012
Available online 12 April 2012

Keywords:
Graphics processing units (GPU)
Geostatistical simulation
Random paths
Parallelization
Compute unified device architecture (CUDA)

ABSTRACT

Geostatistical simulations have become a widely used tool for modeling of oil and gas reservoirs and the assessment of uncertainty. One important current issue is the development of high-resolution models in a reasonable computational time. A possible solution is based on taking advantage of parallel computational strategies. In this paper we present a new methodology that exploits the benefits of graphics processing units (GPUs) along with the master–slave architecture for geostatistical simulations that are based on random paths. The methodology is a hybrid method in which different levels of master and slave processors are used to distribute the computational grid points and to maximize the use of multiple processors utilized in GPU. It avoids conflicts between concurrently simulated grid points, an important issue in high-resolution and efficient simulations. For the sake of comparison, two distinct parallelization methods are implemented, one of which is specific to pattern-based simulations. To illustrate the efficiency of the method, the algorithm for the simulation of pattern is adapted with the GPU. Performance tests are carried out with three large grid sizes. The results are compared with those obtained based on simulations with central processing units (CPU). The comparison indicates that the use of GPUs reduces the computation time by a factor of 26–85.

1. Introduction

The past two decades have witnessed the development of increasingly advanced algorithms for geostatistical simulations, which are used to generate models of oil and gas reservoirs. Such algorithms are highly parameterized and use complex statistical models. Although they offer several advantages over the more traditional deterministic models, their computational requirements often make them impractical. Thus, a most pressing problem in the geostatistical simulations is developing methods that can reduce the computational time, which is needed for generating high-resolutions models of oil or gas reservoirs that exhibit realistic variability in the spatial distributions of their properties, such as the permeability and porosity.

Several strategies have already been proposed for reducing the required heavy computations associated with geostatistical simulations. For example, Strebelle (2002) used a search tree as a structure that can process the data efficiently, and utilized it in single normal-equation simulation (SNESIM), in order to accelerate the computations. The method requires, however, considerable computer memory, particularly when it is used in multiple-facies simulations. Zhang et al. (2006) used cross partitioning and the *k*-means clustering techniques in simulation algorithms with filters (FILTERSIM), in order to reduce the dimensionality of the data, which led their algorithm to be used with some representative scores, instead of large data sets. Honarkhah and Caers (2010) applied multidimensional scaling and kernel space mapping to reduce the complexity and dimensionality of the data set, which led to more accurate results with shorter computational time. Most of the data reduction techniques have, however, some user-dependent parameters, the choice of which may affect the final results. In addition, there is no general criterion for optimal reduction of the dimensionality of the data.

In this paper we propose to address the problem of computational cost of geostatistical simulations by using a parallel computational strategy. If used properly, the strategy makes it possible to carry out highly efficient computations, particularly when their cost is compared with that of a single processor. In conjunction with the method one must address another important problem, namely, the fact that it may not be possible for every one to utilize a large number of computers or processors. Therefore, it is necessary to consider a more general, more accessible, and computationally cheaper device that has a similar or even improved computational capacity, when compared with machines that consist of a large number of parallel processors.

The goal of this study is to palliate the problem by using graphics computational units (GPU), a concept developed by

* Corresponding author.
E-mail address: moe@iran.usc.edu (M. Sahimi).

0098-3004/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cageo.2012.03.028

computer scientists. We present a general strategy for geostatistical simulations that are based on random paths, which may be easily implemented with other geostatistical simulation methods as well, such as FILTERSIM. The method is based on a master–slave strategy, where a central processor – the master – manages a pool of processors – the slaves – each carrying out a part of the simulations on a separate node. Such a strategy has been used heavily (Tahmasebi et al., 2012, and references therein) in large-scale parallelized scientific simulations. We also compare the efficiency of the method with an alternative parallelization strategy that shares the simulation of a single grid point in the computational grid among many threads. Although our interest is in such geoscience applications as modeling of petroleum reservoirs, mining and hydrology, the method is of interest to researchers in other fields, and in particular simulation of materials at molecular level.

The rest of this paper is organized as follows. We begin by briefly discussing the background to the present work. We then describe the GPU and compute unified device architecture (CUDA), a parallel computing architecture developed by the technology company Nvidia (2008), which we utilize in our work. CUDA makes it possible to benefit from the ability of GPU in parallel computing with general-purpose computation with GPUs (GPGPU). One may, of course, use other frameworks as well, such as the OpenCL, the Open Computing Language (Khronos, 2010). The details of the approach and its implementation are then described, including the important issue of conflict management. Two parallelization methods utilized in the present study are described in Section 5. To demonstrate the advantages of the GPU-based geostatistical simulations, we compare in Section 6 its efficiency using the method of simulation with patterns (SIMPAT) and a high-resolution model, with that obtained using the usual central processing units (CPUs) and a parallelization strategy. The paper is summarized in Section 7, where we also describe some possible future directions in this active area of research.

2. Background

The traditional methods based on two-point geostatistics include the sequential Gaussian simulation (SGS) (Journel and Isaaks, 1984; Goovaerts, 1997; Sahimi, 2011), sequential indicator simulation (SIS) (Goovaerts, 1997; Sahimi, 2011) and co-simulation methods (Goovaerts, 1997; Chiles and Delfiner, 1999). Most of such methods are point- and probability-based algorithms in which the variogram plays an important role. Recent developments have led to multiple-point geostatistical (MPS) methods and such algorithms as SNESIM (Strebelle, 2002), SIMPAT (Arpat and Caers, 2007), FILTERSIM (Zhang et al., 2006), and direct sampling (Mariethoz et al., 2010). Other methods that are based on optimization methods (for a recent review see Sahimi and Hamzehpour, 2010), such as the simulated annealing method (Hamzehpour and Sahimi, 2006a,b; Hamzehpour et al., 2007) and the genetic algorithm (Sanchez et al., 2007; Li et al., 2011) have also been used.

Most methods of geostatistical simulation are based on a sequential procedure in which the algorithm tries to construct a model for the probability distribution of a random variable (Deutsch and Journel, 1998), such as the permeability, which eventually leads to the generation of a realization of a reservoir or sample rock. In the case of multivariate stochastic systems, it is possible to use the so-called ν/τ (nu/tau) model (Bordley, 1982; Journel, 2002; Krishnan, 2004; Polyakova and Journel, 2007) to combine the local conditional cumulative distribution functions (CCDFs), which are conditioned on each of the local or smaller data sets that they represent, into a single distribution function. Depending on the simulation method used, various techniques may be used for determining the CCDF. For example, the variograms are used in the SIS or SGS, and training images in the MPS. Therefore, the essence of a randomly based sequential geostatistical simulation may be summarized as one in which all the grid points in the computational grid along a random path are visited. The conditioning data, which consist of the neighborhood hard (quantitative) data and the previously computed grid points, to which values of the stochastic variables have been attributed, are selected. Next, the local CCDF is determined, given a geostatistical simulation method. Finally, a value from the constructed CCDF is drawn, attributed to the currently visited grid point, and added to the simulated data. The procedure is continued until all the grid points in the computational grid have been assigned a value, constituting a single realization of the actual system. It is then possible to produce another realization with a different starting point for the random path.

But, the sequential nature of geostatistical simulations makes them difficult to parallelize efficiently. In addition, one reason for the difficulty is due to steps taken during the parallelization that may contradict others in terms of the parallelization. This issue will be discussed further below. Comprehensive discussions of the parallelization of geostatistical simulations are given by Mariethoz (2010), who describes several strategies for various aspects of the problem. A straightforward solution for the parallelization problem is to use several processors, each given the task of generating one realization independent of the others. However, in the initial phase of any modeling one often wishes to investigate the sensitivity of the results to specific parameters by carrying out simulations using only a few realizations. In such cases, it is necessary to generate quickly one or a few realizations and, thus, alternative parallelization methods are necessary.

Another promising method is to divide the simulation grid into a few sections, according to the number of the available processors, in a way that they overlap in an area half the size of the neighborhood. Such a procedure is applicable only to a case in which the size of the computational grid is greater than the range of the variogram used (Kerry and Hawick, 1998; Gebhardt, 2003; Pedelty et al., 2003; Dimitrakopoulos and Luo, 2004; Vargas et al., 2007; Ingram and Cornford, 2008; Strzelczyk et al., 2009). But, problems may arise in the communications between the processors that can slow down the simulation, if a large number of processors are used. Furthermore, it is possible to break down the computational load for each grid point by parallelizing the computations for each grid point (Ortiz and Peredo, 2008; Nunes and Almeida, 2010; Straubhaar et al., 2011).

Mariethoz (2010) proposed to distribute the grid points among several machines that provide a large number of processors, consisting of two main types: some subprocessors – the slaves – that carry out most of the computations, and a master processor that communicates with the slave processors in order to combine the outcomes of their simulations and avoid conflicts (see below). Although such a strategy is an effective method for parallelization of geostatistical simulations, in the case of large grids the master processor must process a huge amount of data, which becomes problematic. Moreover, providing a large number of serial machines is very expensive.

Thus, we proposes an alternative that is much more efficient, and conceptually straightforward to implement. In what follows we first explain the main concepts and ideas, and then describe its application to the problem that we study, namely, speeding up geostatistical simulations.

3. Graphics processing units

The GPUs and central processing units (CPUs) are completely different computational architectures. The former were initially

designed with the specific purpose of accelerating video data processing. In recent years GPUs have become a hardware component specifically designed for parallel computations. Let us illustrate this with an example. Suppose that we wish to look for a specific word in a book. If the task is given to a CPU, it reads the book in its entirety, from the first to the last page, in order to find the word because the CPU is a serial processor and does the search sequentially. If, however, we use the GPU, which is a parallel processor, it divides the book into a large number of parts and reads all the parts *simultaneously*. Even if each part is read slower than by the CPU, the entire book is read in a much shorter time. The use of graphical processors for nongraphical purposes is called general-purpose computing on graphics processing units (GPGPU).

One of the best processors, Intel Core IV, is composed of eight cores, with four of them being virtual core kernels simulated by *hyper-threading* technology, which is used for improving parallelization of computations. For each process core that is physically present, the operating system addresses two virtual processors and then shares the workload between them, when possible. Concurrently, a cheap graphic card presents a structure that is composed of tenfold of Core IV. For example, in Nvidia GTX 295, 480 threads or processors are combined. Such a large number of processors endow GPUs with tremendous ability for parallel computing. GPUs also have wide applications in fast Fourier transform computations (Moreland and Angel, 2003), signal processing (Fung and Mann, 2004), digital image processing (Jeong et al., 2011), molecular computing (Monte Carlo and molecular dynamics simulations) (Ivan et al., 2008; Alerstam et al., 2008), etc. The trump card of Nvidia is the CUDA that, in essence, is a parallel computing architecture and is available for users by a variety of programming languages. In other words, CUDA provides an opportunity to use the GPUs for computational programming.

GeForce: This is a line of GPUs designed by Nvidia, and used in the present study. The GPU architecture is composed of several stream processors on a chip; see Fig. 1(a). They are grouped in clusters of multiprocessors that are organized hierarchically; see Fig. 1(b). The device and the host are the GPU and CPU, respectively. In other words, most of the computations and data are off-loaded onto the device (GPU), with each of them having their own random access memory (RAM). The RAMs for CPU and GPU are called dynamic RAM and device memory, respectively. Each multiprocessor has a shared memory that can be accessible from each of its processor. Moreover, there is a larger memory space, called global memory, for each of the multiprocessors; see Fig. 1(c). The global memory is a main tool of read/write or R/W information transmitting between the host and the device. Normally, the local memory has a small and fast space, used for calling the data from the global memory, while the threads are stored on their local registers. According to Fig. 1(c), the threads have access to *both* local and global memory. Furthermore, some caches are set to speed up the access to their memories. The constants are initialized by host and are visible to all threads.

An operation, named kernel, is used for executing the various portions of an application in a parallel mode, which can be executed independently many times on different data sets. Thus, it is equivalent to a function that is called on from the host and runs on the device. A thread block is a collection of threads that share the data through the local memory and synchronize the execution work together. But, the threads in different blocks cannot work together. A set of blocks represents the grid, as shown in Fig. 1(c). At any time, several threads on a kernel are executed; they are referred to as the grid of the threads blocks. The threads and blocks have their own identities, e.g. block ID, one or two dimensional, and thread ID, which is 1D, 2D or 3D, which simplify the memory addressing for high speed processing.

It should be pointed out that there are some differences between the threads of CUDA and CPU. For example, the CUDA threads are very lightweight, which is due to very little creation overhead and instant switching. CUDA uses thousands of threads in parallel, whereas the available CPUs can use only a few. This architecture is shown in Fig. 2. The main steps for CUDA are summarized as follows (see Fig. 2). First, the memory is allocated on the host and device separately. Then, by using the CUDA application programming interface (API), the data from the host are copied into the device, after which the numbers of threads and blocks are defined. Next, on each thread the kernel function is executed in parallel with those on other kernels. Finally, the data from the device are copied onto the host using the CUDA API.

Despite all the advantages of GPU, it does have some limitations as well. For example, since in the currently available GPUs all the threads in a block work simultaneously, it is inefficient to have each of them perform a different operation. As soon as one of them has finished its work, we feed it with new data. In other words, in the first group that is composed of several grid points, the GPU performs parallel computations. Since the group that is fed into the GPU is free of any conflict, the simulation is done easily. Then, the next group of grid points is transmitted to the GPU. An advantage of using groups of grid points is avoiding data transmission between the CPU and GPU, because the data bandwidth transmission between the two is relatively low, which makes it critical to transfer the minimum amount of data between the two.

4. Conflict management

In this study we define a conflict as a condition in which the neighborhood of two grid points that are sending data to GPU are also used by another slave processor. This is shown in Fig. 3. Managing such conflicts, which slow down the computations, is an important aspect of the method that we propose. Thus, we first discuss this issue.

4.1. Supermaster architecture

Before describing the conflict management, it is important to mention general processors architecture by which we manage the conflicts. In this paper we use the idea introduced in computer science for CPU architectures by Tanenbaum (2003), which involves one or several CPUs – called the masters – controlling several other CPUs – the slaves – and "supervise" data transfer and analysis. With this method, distributing the segmented data among the slaves allows for fast computations. The slaves act like a chained block, each of which doing its own computations. After finishing their computations, the slaves return their results to the master CPU. With efficient management of the slaves, it is possible to achieve high parallelization. Therefore, the master is responsible for all the duties of and orders to the slaves. Usually, most of such systems have one master and a set of slaves. In this study we used a processor as the master and several processors in GPU as slave. The number of the slaves is equal to the number of stream processors, i.e. 240. However, in some cases it is possible to use a *supermaster* that is responsible for some lower level masters that manage groups of slaves. Therefore, the lower-level masters are defined as slaves for the first-level master or the supermaster. This concept is shown in Fig. 4 in which both simple and complex master–slave architectures are shown.

4.2. Methods for conflicts management

In general conflicts are likely to appear between the slaves. According to the master-centered strategy, the master should

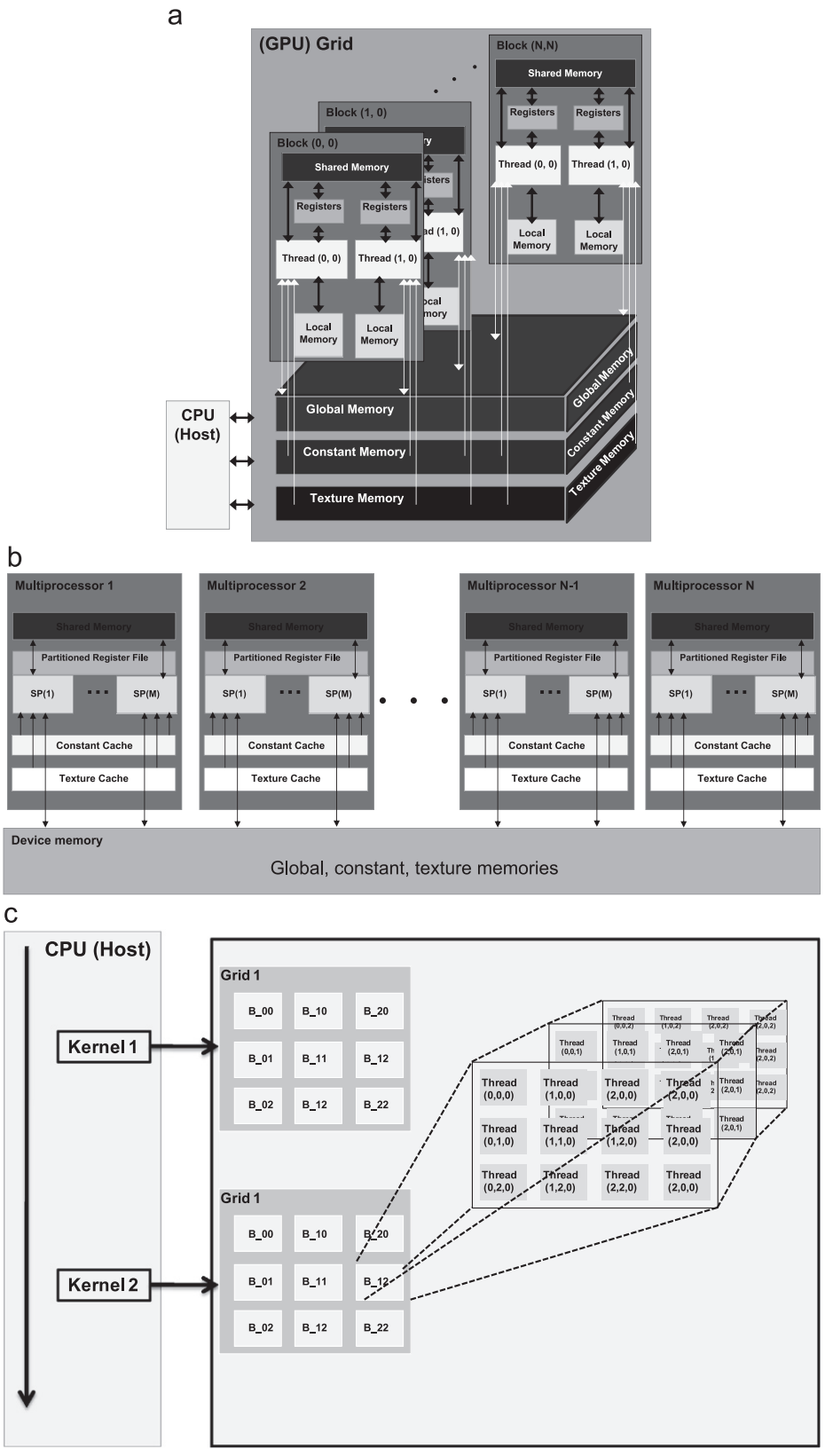

Fig. 1. CUDA programming model. (a) A recall execution model that shows the global GPU structure by which it is able to parallelizes the commands (SP is instant for stream processor); (b) a detailed image of CUDA device memory allocation with different parts of global, constant and texture memory, and (c) a schematic image of the threads and blocks allocation that depicts the kernels as a collection of threads in different blocks and grids (with a modification, adopted from Nvidia, 2008).

solve the problem by deciding and managing the order of the data transmission. There are several methods for resolving the conflict.

Blocking: In this method the conflict is blocked or ignored. In other words, it is assumed that there is no conflict and the algorithm ignores the new grid point that has given rise to the conflict. The

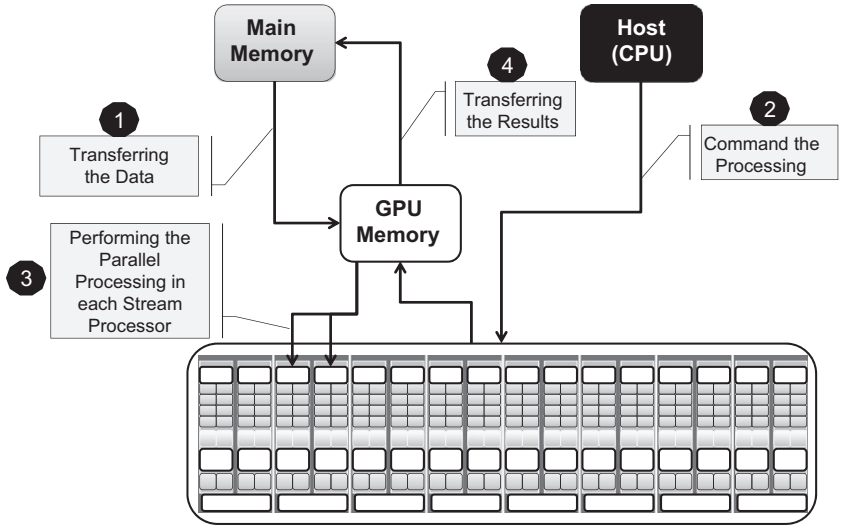

Fig. 2. The schematics of GPU that is used for fast simulation. Shown are the parallel processors and also their connections.

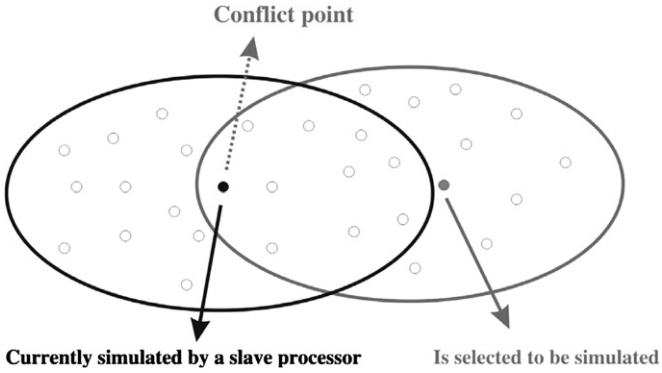

Fig. 3. Illustrating the conflict problem in parallel GS. The black ellipsoid belongs to a point that is currently simulated by a slave and the gray ellipsoid indicates a new point which is to be simulated. As shown, one of the neighborhood points of the second point is used by a slave that causes conflict.

method is suitable for pattern-based simulations in which the blocked grid points may take on a value through their neighborhood patterns in the subsequent grid points that are simulated. The method is not very efficient when a large number of conflicts occur during the simulations, because in such a case ignoring the conflict leads to artifacts in the realizations. If, however, the ratio of the stream processors and the data density and grid size is small, then the method may be applicable because in this case very few conflicts are likely to happen (Mariethoz, 2010).

Busy-wait: In this method the slave that has conflict stops waits up to the time at which there is no conflict in its neighborhood. The strategy is efficient for the case in which the number of conflicts is relatively small. Otherwise, a large number of slave processors should wait and be idle, which increases the computation's cost tremendously. In order to save time, one can use a conflict indicator (CI) that allows the slave processors to know about the grid points that give rise to conflicts. Initially, the CI value for all the grid points is zero, but when a grid point causes a conflict, the CI changes to 1. Clearly, the conflicts are not perpetual and, therefore, as the simulation proceeds a grid point with CI$=1$ may again take on a zero value, a unit value again, and so on.

Wait-t: This method is a comprehensive strategy in which a threshold is defined for the number of times t that the algorithm checks the conflicted grid points sequentially. In other words, the user defines the number t of the tries or waiting periods, or checks that the algorithm should execute, or return to the grid point in order to resolve the conflicts. Clearly, the CI concept may be used in *wait-t* (Mariethoz, 2010).

Postponing: In this strategy the conflict points are transferred to the latter stages of simulation. In other words, one tries to overcome a conflict by postponing the conflict points to the next or latter groups that should be simulated. The strategy is illustrated in Fig. 5.

The aforementioned strategies may be used for parallel computing. However, not all of them are suitable for the GPU applications. In fact, some of them are applicable to only specific simulation methods. For example, the blocking strategy is suitable for pattern-based geostatistical simulations. In addition, when all the multiple processors among the available GPUs have finished their tasks, the results are transferred to the global memory and then to the CPU. Therefore, the GPU should wait for the results from all the multiple processors before the next kernel can be executed.

In the present study we selected an efficient conflict management to minimize such limitations. After some preliminary simulations we found the postponing strategy to be the most efficient for the problem that we study. According to the strategy the CPU exchanges the grid point that causes the conflict with the first grid point of the next group. When applying the postponing strategy, we used the maximum number of the grid points that did not cause conflicts, in order to take into account the low bandwidth between the CPU and GPU, thus minimizing the frequency of communications between the two. In other words, in each loop for using the GPU, we transmitted the maximum possible nonconflicted grid points to the GPU. As pointed out earlier, due to the limitation in the data transferring, it is better to save time and minimize the CPU interactions with GPU. Furthermore, it is possible, when using very large computational grid, to postpone the conflicted grid points without any major modification in the random path. In other words, since a large grid contains a very large number of grid points, postponing represents simply another random or possible path, for which the number of conflict occurrences is very small, hence preventing the simulation from generating any artifacts.

5. Algorithm

We now present an efficient solution for sequential geostatistical simulations based on random paths by a combination of the

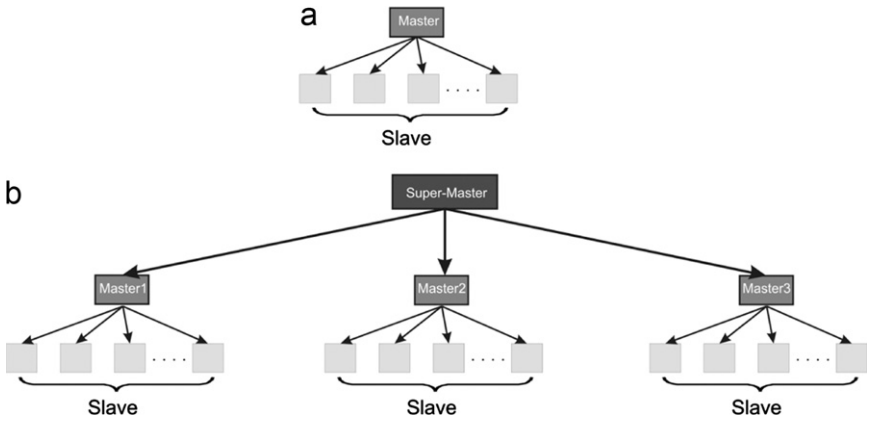

Fig. 4. Graphical scheme of, (a) a simple master–slave, and (b) a master–slave subsystem architectures.

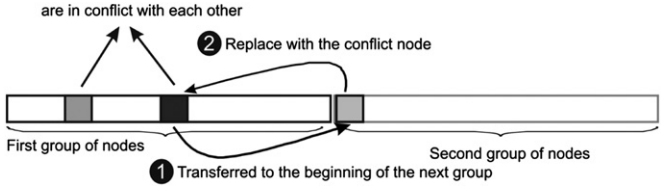

Fig. 5. The illustration of conflict management by the postponing strategy in which simulation of the conflicted point is postponed to the next group of points to be simulated.

GPU structure and the master–slave strategy. For this purpose, we used Nvidia GeForce GTX 285 with 1024 MB of GDDR3 memory, 240 stream processors and 159 GB/s memory bandwidth on a 2.66 GHz Intel core 7 CPU with 4 GB of RAM, together with the architecture of the master–slave strategy. The SIMPAT (Arpat, 2005; Arpat and Caers, 2007) was used for the geostatistical simulations, and two parallelization levels, one at the path level and a second one at level of grid points, were implemented. As is well known, such the SIMPAT algorithm is as follows. First, the training image, used for generating the realizations, is scanned with a template T, and the extracted patterns are stored in patdb_T. Then, the algorithm continues along a random path and at each grid point performs a large number of distance calculations. For example, in the simplest case the algorithm calculates the distances between points in the current pattern of the data event, $\{x\}$ or dev_T, and those in the existing patterns in patdb_T, and then selects randomly one of the patterns among all the candidates.

5.1. GPU parallelization at the path level

As the first step the master selects some initial points that have a reasonable distance from each other, in such a way that their neighborhoods do not overlap. It then sends the grid points to the slaves, so that they begin to do the calculations. In the simplest case of Fig. 4, Fig. 4(a), all the stream processors in the GPU include the slaves, while a CPU is defined as the master. Designating a large number of processors as the slaves and using a single master would not be efficient, because the computational cost for the master would increases dramatically, resulting in most of the slaves waiting for the master, hence leading to many conflicts.

The methodology proposed in this study prevents such problems and allows us to manage the random path very efficiently with a large number of GPUs. The idea is to define several memory sections and utilize the very fast computational ability of multiple processors of the GPU for highly efficient simulations. Moreover, among other advantages, we also benefit from using many threads in parallel, which breaks down the computational load into different processors.

The overall algorithm is summarized in Fig. 6. First, all the necessary data and parameters are transferred from the host to the global memory of GPU, in order to be used by it. This represents one of the most time-consuming steps in the application of GPU to geostatistical simulations, because a large amount of data is sometimes necessary, e.g. for the MPS methods. Then, the CPU considers the maximum number of grid points in the random path that have no conflict with each other, and sends their coordinates and neighbors to the global memory of the GPU. In this study, depending on the size of the simulation grid, we assumed the maximum number of grid points to be equal to the ones that have no conflict with each other. Next, the data are shared and allocated to their local registers of the slaves. Each piece of the data in each block is subsequently divided by CUDA into smaller sets called *threads*, which makes the kernel commands to be executed in parallel. Since the number of threads and processors can be different, the threads execution will be managed by CUDA. Such an advantage can be used effectively in geostatistical simulations by a proper definition of the neighborhood and dividing the grid points among them. Then, when the execution of a thread on a GPU core is terminated, CUDA uses it for the next thread. After this step, each stream processor simulates the given grid point, writes the results in the global memory, and then waits for the computations for all the other grid points in the GPU to be finished, because they need to be transferred to the host memory *together*.

5.2. GPU parallelization at the level of grid points

Another approach for parallelization that we examined was one in which the grid points are presented sequentially to the GPU. This approach was tested to demonstrate the ability of the conflict management methodology. It was aimed at increasing the search performance between dev_T, the data event with the size of template T, and patdb_T, the data base of the patterns. From the perspective of design complexity, rather than the path-level parallelization, this method is relatively simple and consists of dividing the search procedure into different stream processors of the GPU. Furthermore, the method does not need any conflict management, because the grid points are simulated sequentially. Instead of searching all the patterns in patdb_T, it is possible to divide patdb_T into several parts, where each processor is responsible for a specific part of the patdb_T. Therefore, each of the stream processors identifies a most similar pattern, which is then sent to the master processor. The final best pattern is then determined,

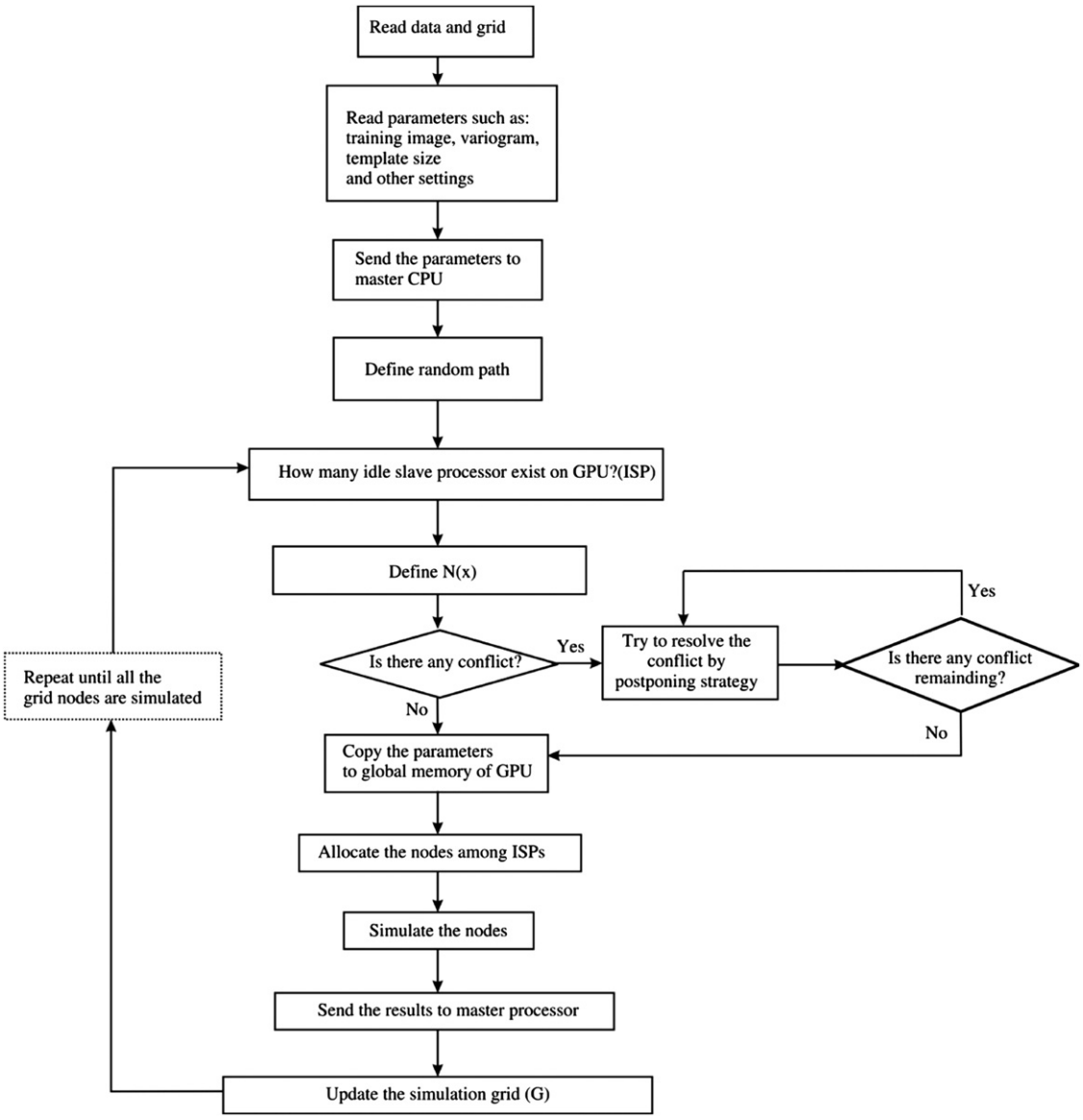

Fig. 6. The flowchart for incorporating GPU-CUDA that is used in this study.

and the rest of the simulation, which has very small computational cost, is continued by the CPU.

6. Results and discussion

The parallelization procedures that were described were utilized with the SIMPAT, which is an MPS pattern-based simulation method. Recall that the main differences between the pattern-based methods, such as FILTERSIM, and algorithms based on grid points, such as the SGS and SIS, are the simulation prototypes. In the former case the aim is to simulate one set of grid points at a time, whereas in the latter case only one grid point is simulated in one loop. The parallelization strategy presented in this paper may be easily extended to other grid point-based simulations. For example, the pattern can be replaced by a set of grid points in the random path, while the essence of the strategy remains the same.

To test the efficiency of the method, we considered simulation grids with the sizes $50 \times 50 \times 30$, $100 \times 100 \times 50$ and $150 \times 200 \times 30$. The training image, shown in Fig. 8, has two facies with proportions of 28% sand and 72% of shale, and contains a relatively complex distribution of flow paths and barriers. It is an example of a large and complex system with flow channeling. Due to the overlaps between the channels, modeling of this type of system is very difficult, with most of the traditional simulation methods being unable to reproduce the image efficiently. This is why one must use an MPS method for simulation of this type of system.

The efficiency of the proposed method of conflict management, namely, the postponing method, was also tested and compared with the parallelization at the level of the grid points. The results of the path-level and grid point-level parallelization and their comparison with the CPU performance for the three grids are presented in Tables 1 and 2, respectively. The speed-up ratio for the GPU over the CPU is shown in Fig. 7. Some of the generated realizations of the system are shown in Fig. 8.

According to Tables 1 and 2, it is clear that in both strategies the GPU implementation is significantly superior over the CPU implementation. Moreover, the hybrid master–slave strategy and path-level parallelization exhibit clear advantages over the grid point-level approach. The grid point-level strategy simply divides the pattern's data base into some classes in order to calculate

Table 1
Comparison of computation times t_{GPU} and t_{CPU} for the path-level parallelization strategy.

	Simulation 1	Simulation 2	Simulation 3
Template size (T)	15	15	15
Improvement (t_{GPU}/t_{CPU})	53	73	85

Table 2
Comparison of computation times t_{GPU} and t_{CPU} for the point-level parallelization strategy.

	Simulation 1	Simulation 2	Simulation 3
Template size (T)	15	15	15
Improvement (t_{GPU}/t_{CPU})	26	38	60

faster the distance between the data event and the patterns, but it does not use efficiently the ability of the CUDA for GPU programming. However, this approach is still much faster than using only the CPU, as shown in Fig. 7, which presents the improvement ratio for both master–slave and second strategies with the GPU.

We point out that, without the conflict management strategy, there would be a decrease in the performance with increasing the number of the GPU stream processors, due to the increased communication and synchronization caused by the conflicts. In other words, due to the large number of processors and random-based simulation, several conflicts can arise, hence the necessity of having a strategy for managing them. It should also be noted that for a successful application of the GPU for parallel programming, the data transmission between the GPU and CPU must be minimized. For example, it is inefficient to use the GPU parallelization for an algorithm that transfers large amounts of data between the CPU and the GPU. In the case of geostatistical simulations in which the volume of the transferred data is single-valued, such as that for pixel- or point-based geostatistical simulations, or a small set of values as in, for example pattern-based geostatistical simulations, the data transmission between the CPU and GPU does not have a significant effect on the overall computing time.

7. Summary and future work

This paper investigates the possibility of using the GPUs with the CUDA libraries for applications to geostatistical simulations, which are becoming increasingly CPU-expensive, due to the advent of novel simulation methods. The limitations of the CPU-computing led us to propose the GPU parallelization as a solution. This is important because computational limitations can prevent reservoirs modelers from using geostatistical simulations. For example, there is always a trade-off between the speed and the template size, or the number of grid points in the neighborhood. Existing parallelization methods are designed for multi-CPU machines that are expensive and not available to many modelers.

The GPUs offer an affordable alternative and, as we show in this paper, large computational speed-up by up to nearly two orders of magnitude over the CPU in the examples that we studied. However, there are challenges specific to the use of the GPU, such as the large number of processors that requires using an efficient method to manage its performance. In this paper we used a master processor for managing the stream processors – the slaves – on the GPU.

It should be clear that the computational cost of conflict management is directly related to the size of the simulation grid. With a small grid and a large number of stream processors, many

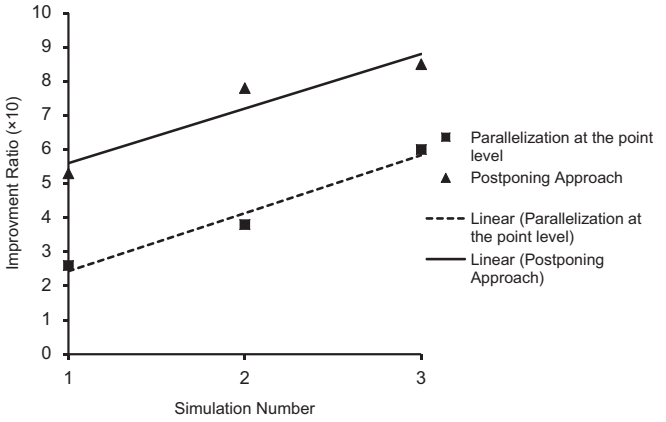

Fig. 7. Comparison of improvement ratio between the master–slave and the second proposed strategies for GPU application.

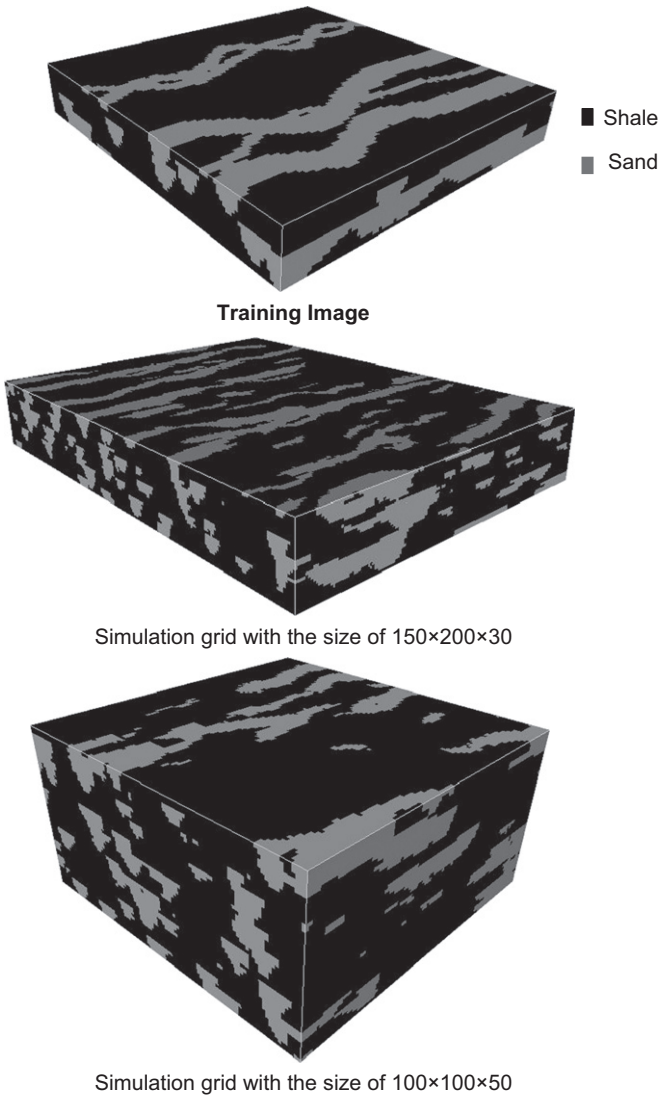

Fig. 8. (a) The training image. The generated realizations with size (b) $150 \times 200 \times 30$, and (c) $100 \times 100 \times 50$.

conflicts may arise, due to the small distances between the data events. If the simulation grid is too small, it will be more efficient not to use the GPU or other parallel CPUs. Otherwise, the postponing method for conflict management can help determining

the maximum number of conflict-free threads and, subsequently, lead to faster simulation. Taken together, such strategies lead the GPU to be much more efficient than the CPU application.

But GPUs do have some limitations. One is the limited speed of data transfer between the CPU and GPU, and the absence of random access to the output. Moreover, since the amount of memory on graphic cards is relatively small, it may not be suitable for use in certain cases. Despite such limitations, we believe that using the GPU enables simulations with higher precision, computed in shorter times.

The method that we proposed can still be developed further. For example, it is possible to combine several GPUs to further accelerate the simulations. Although the GPU technology has been used in geophysical surveys (Gaishan, 2007; Wang et al., 2010) and flow simulation (Kuznik et al., 2010), it should be studied further for use in problems with a large number of constraints that require large computations. Furthermore, some parallel and GPU enhanced versions of various softwares, such as the SGeMS (Remy et al., 2008), and algorithms, such as the so-called impala algorithm (Straubhaar et al., 2011) have been proposed recently for geostatistical simulations.

The present study was dedicated to random-based MPS. We are currently working on the development of an efficient parallelization method for both serial machines and the GPU-based simulations for raster-based MPS simulations (Tahmasebi et al., 2012). The results will be reported in future papers.

Acknowledgment

Work at USC was supported in part by the Department of Energy.

References

Alerstam, E., Svensson, T., Andersson-Engels, S., 2008. Parallel computing with graphics processing units for high speed Monte Carlo simulation of photon migration. Journal of Biomedical Optics 13, 060504.

Arpat, B., 2005. Sequential Simulation with Patterns. Ph.D. Thesis. Stanford University, Stanford, California.

Arpat, B., Caers, J., 2007. Stochastic simulation with patterns. Mathematical Geology 39, 177–203.

Bordley, R.F., 1982. A multiplicative formula for aggregation probability assessments. Management Science 28, 1137–1147.

Chiles, J.P., Delfiner, P., 1999. Geostatistics: Modeling Spatial Uncertainty. Wiley, New York.

Deutsch, C.V., Journel, A., 1998. GSLIB: Geostatistical Software Library and User's Guide, second ed. Oxford University Press, New York.

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

Fung, J., Mann, S., 2004. Computer vision signal processing on graphics processing units. In: Proceeding of the IEEE International Conference on Acoustics, Speech, and Signal (ICASSP 04), vol. 93(6), http://dx.doi.org/10.1109/ICASSP.2004.1327055.

Gaishan, Z., 2007. New alternative to geophysical high performance computing: GPU computing. Information Technology 30, 399–404.

Gebhardt, A., 2003. PVM kriging with R. In: Proceedings of the 3rd International Workshop on Distributed Statistical Computing, Vienna.

Goovaerts, P., 1997. Geostatistics for Natural Resources Evaluation. Oxford University Press, Oxford, New York.

Hamzehpour, H., Rasaei, M.R., Sahimi, M., 2007. Development of optimal models of porous media by combining static and dynamic data: the permeability and porosity distributions. Physical Review E 75, 056311/1–056311/17.

Hamzehpour, H., Sahimi, M., 2006a. Generation of long-range correlations in large systems as an optimization problem. Physical Review E 73, 056121/1–056121/9.

Hamzehpour, H., Sahimi, M., 2006b. Development of optimal models of porous media by combining statistic and dynamic data: the porosity distribution. Physical Review E 74, 026308/1–026308/12.

Honarkhah, M., Caers, J., 2010. Stochastic simulation of patterns using distance-based methods. Mathematical Geosciences 42, 487–517.

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

Ivan, S., Fimtsev, U., Todd, J., 2008. Quantum chemistry on graphical processing units. 1. Strategies for two-electron integral evaluation. Journal of Chemical Theory & Computation 4, 222–231.

Jeong, W.K., Pfister, H., Fatica, M., 2011. Medical image processing using GPU-accelerated ITK image filters. In: Hwu, W.-m. (Ed.), GPU Computing Gems Emerald Edition. Elsevier, Amsterdam, pp. 737–749.

Journel, A., 2002. Combining knowledge from diverse source: an alternative to traditional data independence hypotheses. Mathematical Geosciences 34, 573–596.

Journel, A., Isaaks, E., 1984. Conditional indicator simulation: application to a Saskatchewan deposit. Mathematical Geology 16, 685–718.

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

Khronos, 2010. OpenCL—the open standard for parallel programming of heterogeneous systems, Khronos Groups. ⟨http://www.khronos.org/opencl/⟩.

Krishnan, S., 2004. Combining diverse and partially redundant information in the earth sciences. Ph.D. Thesis. Stanford University, Stanford, California.

Kuznik, F., Obrecht, C., Rusaouen, G., Roux, J.J., 2010. LBM-based flow simulation using GPU computing processor. Computers & Mathematics with Applications 59, 2380–2392.

Li, H., Sanchez, R., Qin, S.J., Kavak, H.I., Webster, I.A., Tsotsis, T.T., Sahimi, M., 2011. Computer simulation of gas generation and transport in landfills. V: use of artificial neural network and genetic algorithm for short- and long-term forecasting and planning. Chemical Engineering Science 66, 2646–2659.

Mariethoz, G., 2010. A general parallelization strategy for random path based geostatistical simulation methods. Computers & Geosciences 36, 953–958.

Mariethoz, G., Renard, P., Straubhaar, J., 2010. The direct sampling method to perform multiple-point simulations. Water Resources Research 46, W11536.

Moreland, K., Angel, E., 2003. The FFT on a GPU. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on graphics hardware.

Nunes, R., Almeida, J., 2010. Parallelization of sequential Gaussian, indicator and direct simulation algorithms. Computers Geosciences 36, 1042–1052.

Nvidia, 2008. Nvidia Compute-Unified Device Architecture (CUDA) programming guide, version 2.0. ⟨http://www.nVIDIA.com/object/cuda_develop.html⟩.

Ortiz, J.M., Peredo, O., 2008. Multiplepoint geostatistical simulation with simulated annealing: implementation using speculative parallel computing. In: Proceedings of geoENVVII-International Conference on Geostatistics for Environmental Applications. Southampton, Britain, p. 384.

Pedelty, J.A., Schnase, J. L., Smith, J.A., 2003. High performance geostatistical modeling of biospheric resources in the Cerro Grande Wild-re Site. Los Alamos, New Mexico and Rocky Mountain National Park, Colorado.

Polyakova, E.I., Journel, A., 2007. The Nu expression for probabilistic data integration. Mathematical Geology 39, 715–733.

Remy, N., Boucher, A., Wu, J., 2008. Applied Geostatistics with SGeMS: a Users Guide. Cambridge University Press, Cambridge.

Sahimi, M., 2011. Flow and Transport in Porous Media and Fractured Rock. Wiley-VCH, Weinheim.

Sahimi, M., Hamzehpour, H., 2010. Efficient computational strategies for solving global optimization problems. Computing in Science & Engineering 12 (4), 74–82.

Sanchez, R., Tsotsis, T.T., Sahimi, M., 2007. Computer simulation of gas generation and transport in landfiils. III: development of landfills' optimal model. Chemical Engineering Science 62, 6378–6390.

Straubhaar, J., Renard, P., Mariethoz, G., Froidevaux, R., Besson, O., 2011. An improved parallel multiple-point algorithm using a list approach. Mathematical Geosciences 43, 305–328.

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

Strzelczyk, J., Porzycka, S., Lesniak, A., 2009. Analysis of ground deformations based on parallel geostatistical computations of PSInSAR data. In: Proceedings of the 17th International conference on Geoinformatics.

Tahmasebi, P., Hezarkhani, A., Sahimi, M., 2012. Multiple-point geostatistical modeling based on the cross-correlation functions. Computational Geosciences, in press http://dx.doi.org/10.1007/s10596-012-9287-1.

Tanenbaum, A.S., 2003. Computer Networks, fourth ed. Pierson Education, Inc., New Jersy, p. 428.

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

Wang, S.Q., Gao, X., Yao, Z.X., 2010. Accelerating POCS interpolation of 3D irregular seismic data with Graphics Processing Units. Computers & Geosciences 36, 1292–1300.

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