Contents lists available at ScienceDirect

Computers and Geosciences

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

Research paper

Acceleration strategies for large-scale sequential simulations using parallel neighbour search: Non-LVA and LVA scenarios

Oscar F. Peredo [*], José R. Herrero

Computer Architecture Department, UPC-BarcelonaTech, Spain

ARTICLE INFO

Keywords:
Geostatistics
Anisotropy
Parallel computing
Algorithms

ABSTRACT

This paper describes the application of acceleration techniques into existing implementations of Sequential Gaussian Simulation and Sequential Indicator Simulation. These implementations might incorporate Locally Varying Anisotropy (LVA) to capture non-linear features of the underlying physical phenomena. The implementation focuses on a novel parallel neighbour search algorithm, which can be used on both non-LVA and LVA codes. Additionally, parallel shortest path executions and optimized linear algebra libraries are applied with focus on LVA codes. Execution time, speedup and accuracy results are presented. Non-LVA codes are benchmarked using two scenarios with approximately 50 million domain points each. Speedup results of 2× and 4× were obtained on SGS and SISIM respectively, where each scenario is compared against a baseline code published in Peredo et al. (2018). The aggregated contribution to speedup of both works results in 12× and 50× respectively. LVA codes are benchmarked using two scenarios with approximately 1.7 million domain points each. Speedup results of 56× and 1822× were obtained on SGS and SISIM respectively, where each scenario is compared against the original baseline sequential codes.

1. Introduction

Classical geostatistics can be used in many applied cases where the values under study show isotropic or regular trends of preferential directions. However, in complex scenarios the results obtained can be unrealistic since the underlying phenomena shows highly anisotropic trends which cannot be reproduced by the classical approach. These kind of complex scenarios arise in geological modelling of faults and veins in mineral reserves, sedimentary deposits in oil and gas reservoirs, environmental modelling of pollution spread, rain fall patterns or animal migration, and mobility patterns in highly populated urban areas. Additional computational issues arise when large-scale domains should be analysed.

Concretely, this work is an effort to advance the state of the art in the field of Sequential Simulation algorithms, both for classical and LVA-based implementations, such as described previously, by presenting the following contributions:

- A novel parallel neighbour search is presented which introduces a new performance improvement for a well-known bottleneck in sequential simulation.
- Parallelization of two well-known classical sequential simulation algorithms (SGSIM and SISIM), which were already parallelized in Peredo et al. (2018), but contained a major bottleneck specifically in the neighbour search strategy.
- Parallelization of two well-known LVA-based sequential simulation algorithms (SGS and SISIM), with remarkable speedups, specially in SISIM.
- Performance evaluation is presented using different three-dimensional scenarios.

This article is organized as follows: Section 2 describes the theoretical background of LVA geostatistics and its main difference with classical geostatistics. Section 3 contains all aspects of the baseline sequential implementations. Section 4 shows the parallelization strategies explored in this work. Sections 5 and 6 summarize the numerical results with a final analysis. Section 7 presents conclusions and future work.

2. Theoretical background

Anisotropy manifests itself as preferential directions of continuity in the underlying phenomena, i.e. properties are more continuous in one orientation than in another. If constant anisotropy is present, a single trend or drift can be observed in the sampled data set or secondary sources of information (Isaaks and Srivastava, 1990). In the case of local anisotropy, each location of the domain in study presents different preferential directions of continuity (Boisvert, 2010; Boisvert

* Corresponding author.
E-mail address: operedo@ac.upc.edu (O.F. Peredo).

https://doi.org/10.1016/j.cageo.2021.105027
Received 22 June 2021; Received in revised form 5 November 2021; Accepted 22 December 2021
Available online 11 January 2022

and Deutsch, 2011), which is commonly known as Locally Varying Anisotropy (LVA).

The LVA geostatistical approach sets the initial path for future developments in terms of numerical implementations that can potentially scale to large scenarios. However, no further analysis or open source code improvements were developed in previous years. This is in part because non-standard algorithms, acceleration and distribution techniques must be applied to the inner kernels of the proposed LVA codes for geostatistical analysis. These inner kernels can be clustered in two groups: the classical geostatistical inner kernels, such as variogram computing, kriging estimation and sequential simulation (Deutsch and Journel, 1998; Chilès and Delfiner, 1999); and dimensionality reduction techniques in high-dimensional spaces (Huo et al., 2004). Both groups are connected by the property of positive definite covariance functions (Curriero, 2006). In order to use the classical methods in contexts where LVA is present, non-euclidean distances must be used instead of straight lines connecting different points in the domain. The non-linear path that connects two points is the shortest path that follows the underlying LVA field. By computing the non-euclidean distances between each pair of points in the domain of study, a distance matrix is obtained. This matrix should be embedded into a similar distance matrix generated in a higher-level space using euclidean distances. If large domains are being studied in $\mathbb{R}^d$ with $d \in \{2, 3\}$, a prohibitive amount of computational resources will be necessary to obtain a sequential simulation using an LVA field as proxy of the underlying anisotropy. Additionally, a common bottleneck for non LVA and LVA approaches is related with the neighbourhood search to perform kriging interpolation across the random path selected for the sequential simulation. This step can be prohibitive in terms of computational resources if large search windows are used.

Regarding previous works related to accelerating large scale geostatistical simulations, novel attempts in isotropic modelling have been reported in Vargas et al. (2007), Nunes and Almeida (2010), Peredo et al. (2015) and Rasera et al. (2015), in order to accelerate classical methods using different algorithmic approaches combined with multicore and distributed architectures, particularly MPI and OpenMP. A recent work described in Peredo et al. (2018) follows the same path, preserving the original values of the single-core execution by splitting the neighbour search and simulation steps. In the same track, Nussbaumer et al. (2018) shows a similar approach using a constant path for multiple simulations, proposing a parallel search for neighbour, as first task, followed by the simulation step, with focus on execution of multiple realizations.

In this work, a deep dive into a specific algorithm for parallel neighbour search is presented, which is also coupled with a previous work from the same authors. The proposed parallelization is applied to four different scenarios, non LVA and LVA based, and also using sequential Gaussian and sequential indicator simulation codes as baseline implementations.

3. Sequential implementation

Regarding classical algorithms, the baseline implementation used in this work was proposed by Peredo et al. (2018), with baseline codes for sequential Gaussian simulation (SGSIM) and sequential indicator simulation (SISIM). Regarding LVA-based algorithms, the baseline implementation used in this work was proposed by Boisvert and Deutsch (2011),[1] with baseline codes for variogram computation, kriging estimation and sequential Gaussian simulation (SGS). Gutierrez and Ortiz (2019) contributed posteriorly with the implementation of sequential indicator simulation (SISIM). All of these codes are based on the well known GSLIB code base (Deutsch and Journel, 1998).

[1] http://www.ualberta.ca/~jbb/LVA_code.html.

The existing LVA implementations are based on the L-ISOMAP manifold learning method (Tenenbaum et al., 2000). In this method, landmark or ”anchor” points are used to approximate non-euclidean distances in an origin space by euclidean distances in a higher or lower dimensional destination space. The mapping of each origin space point to the destination space is denoted as the *embedding* $\mathcal{Z}$.

The steps of the sequential classical SGSIM and SISIM algorithms can be reviewed in Peredo et al. (2018), and briefly in Algorithm 1 from line 6 to 16 (removing non used parameters such as $\mathcal{Z}$, k^{cova} and k^{search}). For the sake of simplicity, the inner loop in lines 12 to 14 was left as is, which is the case of LVA-based SGS, but it is worth to mention that the actual codes can use this simulation loop outside of the domain loop of line 9 as well, even including a new random path for each simulation, which is the case of LVA-based SISIM, non LVA-based SISM ans SGSIM. The steps of the sequential LVA-based SGS and SISIM algorithms are depicted in Algorithm 1 from steps 1 to 16. The first steps of the algorithm (lines 2, 3 and 4) correspond to the L-ISOMAP algorithm, using specific LVA parameters such as the LVA field $\mathbf{F}$, the graph connectivity policy π and the number of dimensions k of the resulting embedding $\mathcal{Z}$. With $\mathcal{Z}$ computed, the next steps of the algorithm are standard sequential simulation, which consists in running an inner loop for multiple simulations using a single random-path (lines 6 to 15). Specifically, we can observe that the baseline implementation contains 4 main parts: graph building, distance matrix building, embedding building and standard sequential simulation routines (SGS and SISIM contain different methods to simulate). Each one contains several subroutines and code parts that were released by the authors without specific focus on large-scale usage. Additionally, the baseline implementations were based on Fortran 90 coupled with C++ code through system calls and disk I/O communications, which entangles the code readability and exposes potential performance issues. In this section, a brief summary of the baseline implementation features of each part is included.

3.1. Graph building routines

As depicted in Algorithm 1, the first step of the L-ISOMAP routines adapted for LVA-based sequential simulations consists in calculating the connectivity graph (line 2) for the domain Ω. The domain must be a regular grid, which is a constraint of the current implementation. The inner steps of the routine `build_connectivity_graph` are depicted in Algorithm 2. The inputs of this algorithm are the domain Ω and specific LVA parameters. These parameters are the LVA field $\mathbf{F}$ and the graph connectivity policy π.

The parameter π is used to define the neighbourhood $\mathcal{N}$ for each domain point which will be considered in the connectivity graph, i.e. for each neighbour an edge will be added to the graph (line 3). In practical terms, the policy π consists of a value Δ which sets the number of separation edges (“hops”) in the regular grid. For instance, $\Delta = 1$ will set a neighbourhood $\mathcal{N}$ of at most 6 points located at 1 hop of separation. The LVA field $\mathbf{F}$ is defined for each domain point as a tuple of five values, namely the angles azimuth (or strike) α, dip β and plunge φ, and the directional ratios r_1 and r_2, representing the ratios between axis X and Y, and axis Z and Y respectively. Using this tuple, a rotation matrix $\mathbf{R} := \mathbf{R}(\alpha, \beta, \varphi, r_1, r_2)$ is computed for each domain point in order to calculate the local anisotropic distance in each cell of the gridded domain (line 4). With the neighbourhood and rotation matrix computed, for each neighbour `neig`, an edge is added to the graph, defined as $e = \{\texttt{ixyz}, \texttt{neig}\}$. Each edge has weight equal to $d = \sqrt{\mathbf{h}^T \mathbf{R}^T \mathbf{R} \mathbf{h}}$ with $\mathbf{h}$ lag vector between the edge endpoints (lines 5 to 9). The last step of the algorithm performs a removal of redundant edges already computed, since the connectivity graph is undirected (line 10). The resulting graph $\mathbf{G}$ is stored in disk in file `grid.out` (line 12).

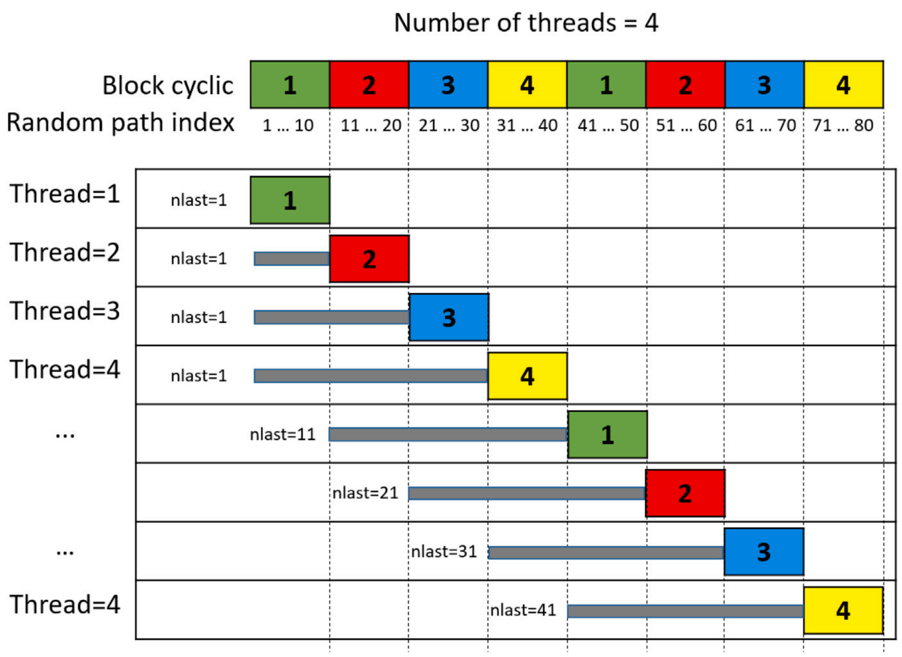


Fig. 1. Load balancing of workload through a block cyclic strategy for parallel neighbour search. In this example, 4 threads are computing neighbours of different blocks of points (block size equal to 10, domain size equal to 80). Before processing a block of points, each thread should declare as *marked* all previous points which are not marked yet by this thread (grey colour line). Variable `nlast` is used to indicate the starting index of marked points for the next block. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

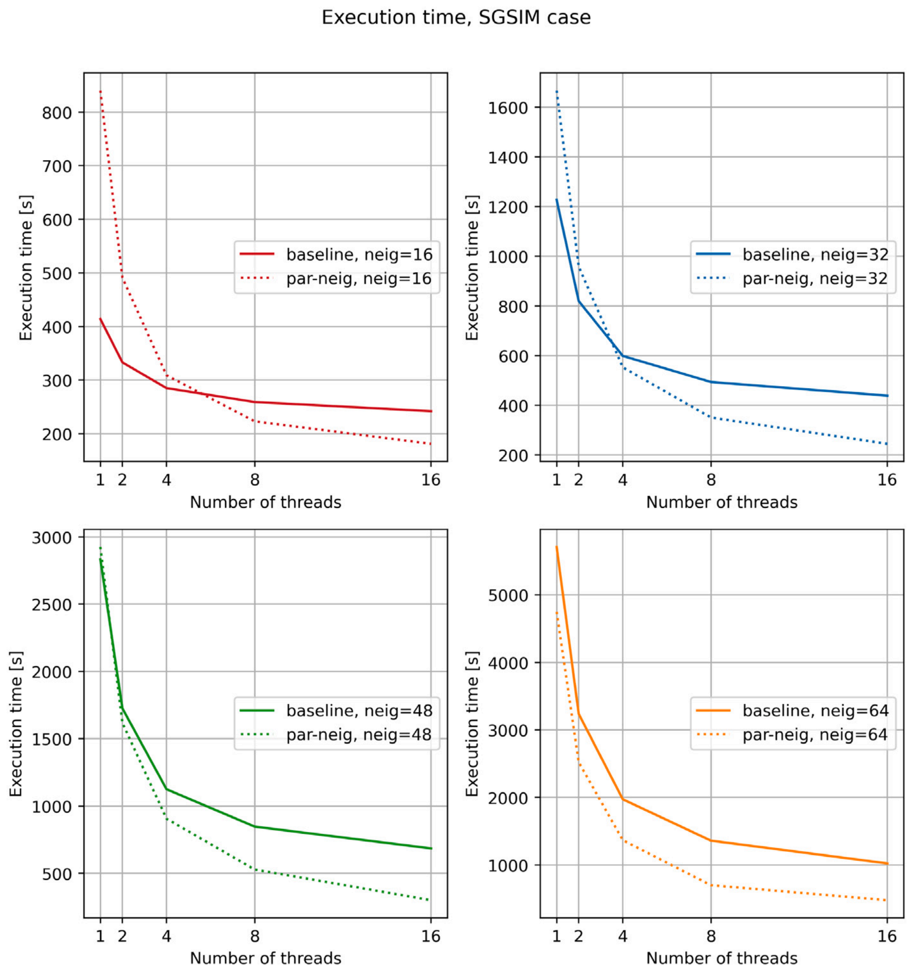


Fig. 2. Execution time [seconds] comparison baseline SGSIM parallel code and adapted SGSIM parallel code using the parallel neighbours search algorithm.

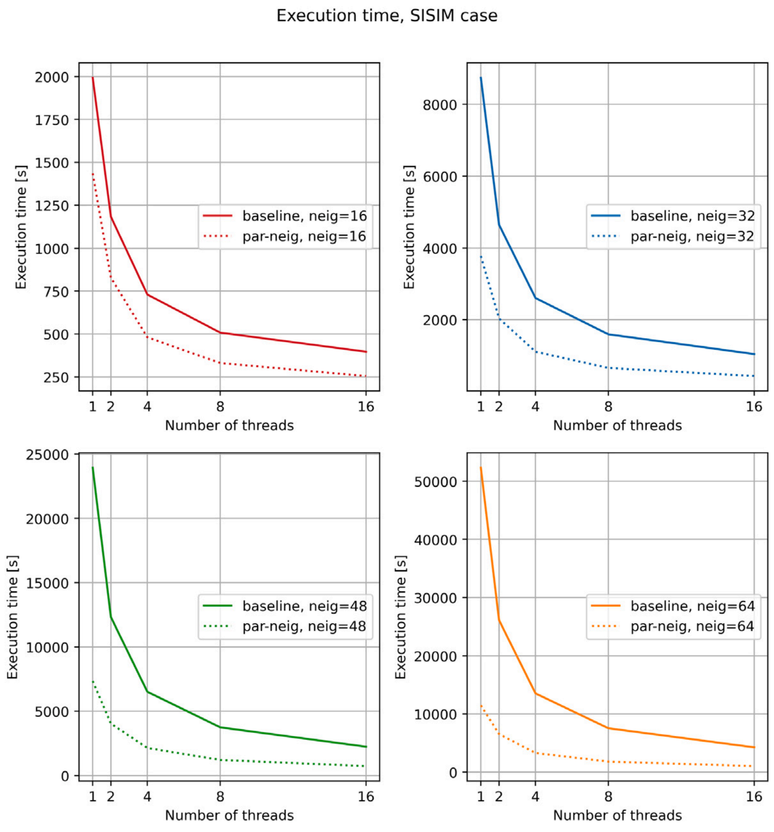


Fig. 3. Execution time [seconds] comparison baseline SISIM parallel code and adapted SISIM parallel code using the parallel neighbours search algorithm.

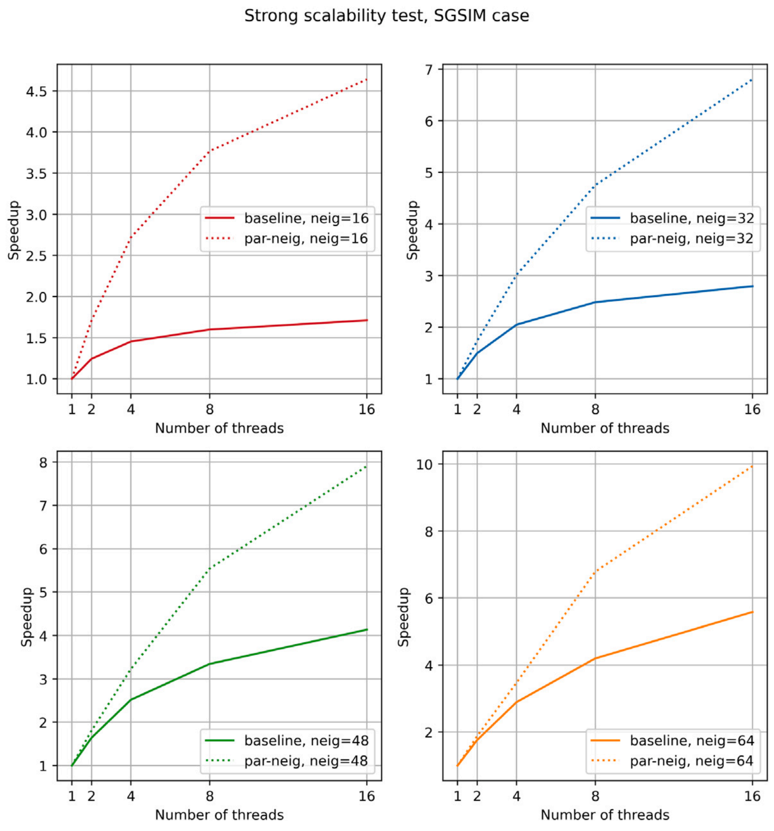


Fig. 4. Speedup comparison between baseline SGSIM parallel code and adapted SGSIM parallel code using the parallel neighbours search algorithm.

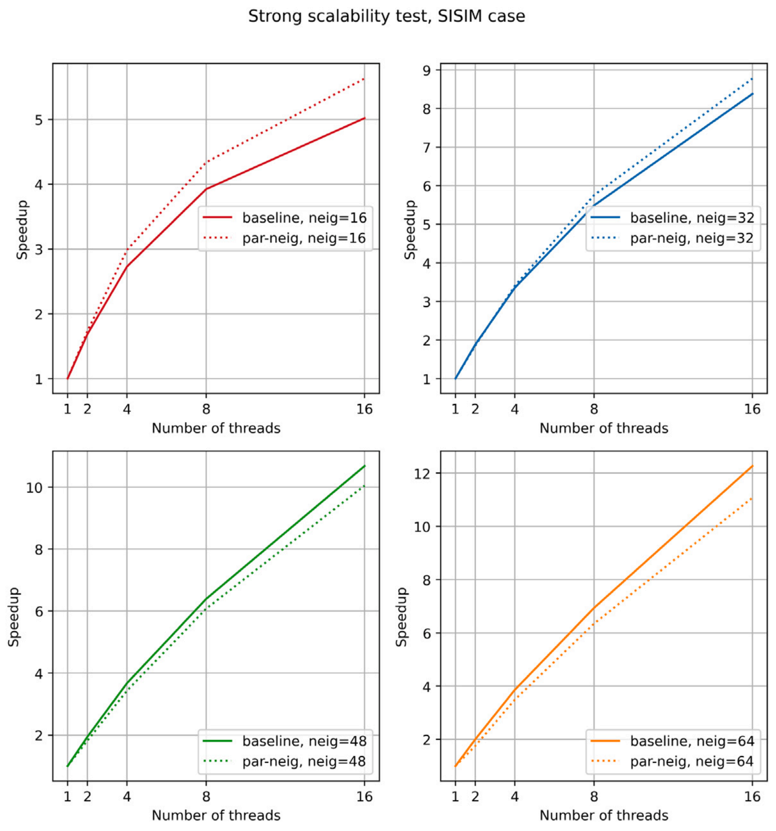


Fig. 5. Speedup comparison between baseline SISIM parallel code and adapted SISIM parallel code using the parallel neighbours search algorithm.

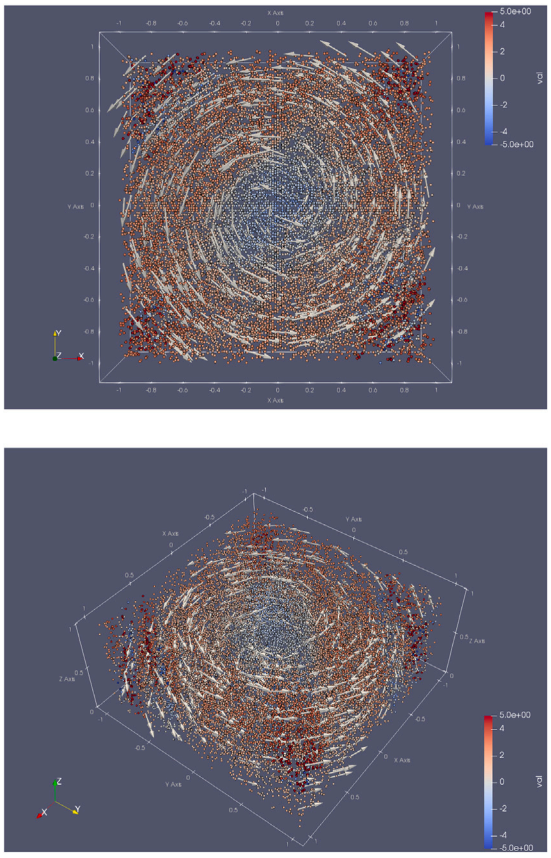


Fig. 6. *swiss-roll*: different views of sample points with LVA field dataset (sample).

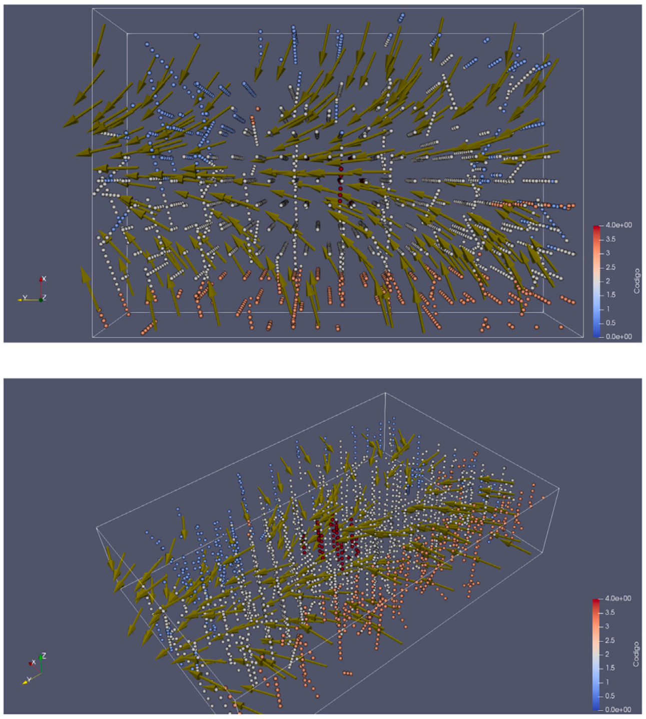


Fig. 7. *escondida*: different views of sample drillhole points with LVA field dataset (sample).

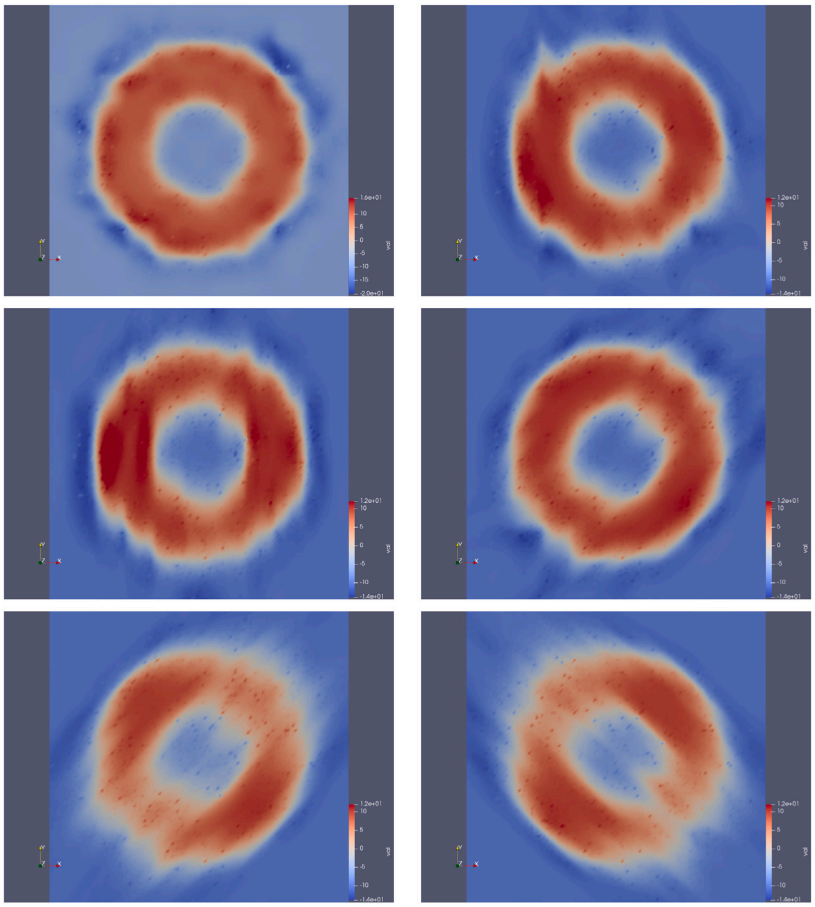


Fig. 8. Slices of simulated domains for *swiss-roll* scenario using parallel LVA-based SGS with different r_1 ratio values from LVA field parameters.

```
Input:
    (V, Ω): sample database values V defined in a 3D domain Ω;
    Ω_L: landmark 3D domain (subset of Ω) [Only LVA];
    F: LVA field defined in Ω [Only LVA];
    π: connectivity graph policy [Only LVA];
    k^search: maximum search distance dimensions [Only LVA];
    k^cova: maximum covariance distance dimension [Only LVA];
    n^max: maximum neighbours for interpolation;
    κ: local interpolation parameters;
    τ: seed for pseudo-random number generator;
    S: number of generated simulations;
    output.txt: output file name
1  // Only LVA: First calculate the embedding using L-ISOMAP
2  $\mathbf{G} \leftarrow$ build_connectivity_graph($\Omega, \mathbf{F}, \pi$)
3  $\mathbf{D} \leftarrow$ build_distance_matrix($\mathbf{G}, \Omega, \Omega_L$)
4  $\mathcal{Z} \leftarrow$ build_embedding($\mathbf{D}$)
5  //non-LVA and LVA: Then proceed with the simulation routines using the
     embedding to calculate distances
6  $\mathbf{P} \leftarrow$ create_random_path($\Omega, \tau$) //Array with index random
     re-ordering
7  $\mathbf{V}^{tmp} \leftarrow$ zeros($|\Omega| \times S$)
8  $\mathbf{V}^{tmp} \leftarrow$ assign($\mathbf{V}$) //Sample data assignment
9  for ixyz $\in \{1, \dots, |\Omega|\}$ do
10     //$\mathbf{P}_{\mathrm{ixyz}}$ corresponds to the index ixyz of the random path P
11     LocalNeighbours $\leftarrow$ search_neighbours($\mathbf{P}_{\mathrm{ixyz}}, \kappa, \tau, \mathcal{Z}, k^{search}$)
12     for isim $\in \{1, \dots, S\}$ do
13         $\mathbf{V}^{tmp}(\mathbf{P}_{\mathrm{ixyz}}, \mathrm{isim}) \leftarrow$
             simulate($\mathbf{P}_{\mathrm{ixyz}}$, LocalNeighbours, $\mathcal{Z}, k^{cova}, n^{max}$)
14     end
15 end
16 write(output.txt, $\mathbf{V}^{tmp}$)
Output: S stochastic simulations stored in file output.txt
```

Algorithm 1: Sequential Simulation for non-LVA and LVA scenarios

```
Input:
    Ω: 3D domain Ω;
    F: LVA field in each domain point of Ω;
    π: graph connectivity policy;
1  $\mathbf{G} \leftarrow \emptyset$ //Empty graph
2  for ixyz $\in \{1, \dots, |\Omega|\}$ do
3      $\mathcal{N} \leftarrow$ Compute all neighbours of point ixyz according to policy
         $\pi$
4      $\mathbf{R} \leftarrow$ Compute rotation matrix of point ixyz according to LVA
         field $\mathbf{F}$
5      for neig $\in \mathcal{N}$ do
6          $\mathbf{h} \leftarrow$ Lag vector between points ixyz and neig
7          $d \leftarrow$ Compute anisotropic distance between point ixyz and
             neig according to $\sqrt{\mathbf{h}^T \mathbf{R}^T \mathbf{R} \mathbf{h}}$
8          $e \leftarrow$ {ixyz, neig} //Definition of graph edge with weight $d$
9          Add $(e, d)$ to graph $\mathbf{G}$
10     end
11     Remove redundant edges from $\mathbf{G}$
12 end
13 Write G to file grid.out
Output: G in file grid.out
```

Algorithm 2: Routine `build_connectivity_graph`

3.2. Distance matrix building routines

The second step of the L-ISOMAP routines adapted for LVA-based sequential simulation is the computation of the distance matrix (line 3 of Algorithm 1). This matrix is computed between domain and landmark points using the connectivity graph **G** computed in the previous step. The inner steps of the routine `build_distance_matrix` are depicted in Algorithm 3.

The baseline implementation reads two files from disk: `nodes2cal.out` (landmark points list) and `grid.out` (connectivity graph) (lines 3 and 4). With both files loaded into memory, for each landmark point a shortest path calculation must be performed through the connectivity graph **G** (line 6). This step is computed using Dijkstra's shortest path algorithm (Dijkstra, 1959), implemented in the C++ Boost Library (Boost.org, 2012). All distances from a landmark to all graph nodes are appended to the file `dist_cpp.out` (line 7). As mentioned before, the baseline implementation performs a system call to launch the execution of a compiled C++ code with the Boost Library call to Dijkstra routine, and the data transfer between Fortran and C++ is performed through expensive disk I/O communication.

3.3. Embedding building routines

Based on the distance matrix **D**, the third step of the L-ISOMAP routines is the computation of the embedding $\mathcal{Z}$ (line 4 of Algorithm 1). The inner steps of the routine `build_embedding` are depicted in Algorithm 4. The input of the algorithm is file `dist_cpp.out`, computed in Algorithm 3, that contains the shortest path distances between landmark and domain points.

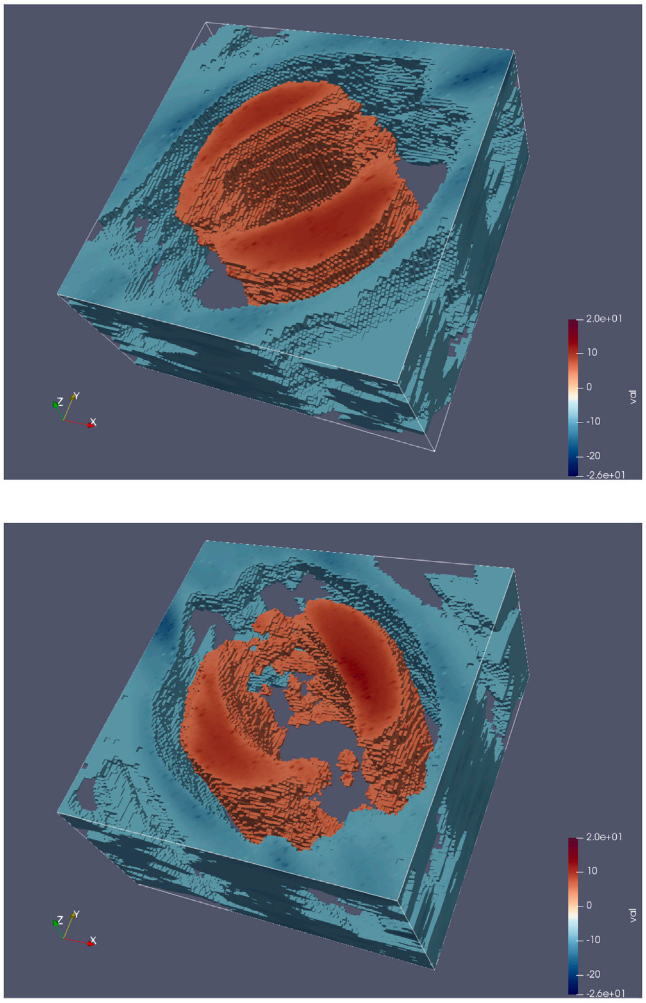


Fig. 9. Top: Simulated domain for *swiss-roll* scenario using parallel LVA-based SGS with $r_1 = 5$ and two threshold views. Bottom: Similar view with LVA parameter $r_1 = 0.2$.

```
Input:
    grid.out: file with graph $\mathbf{G}$ based on domain points $\Omega$;
    nodes2cal.out: file with landmark points indices of $\Omega_L$
1 $(N,n)$ ← Read size of domain points $N$ and landmark points $n$ from
  nodes2cal.out
2 $\mathbf{D}$ ← zeros$(N,n)$
3 $\mathcal{N}^{Landmark}$ ← Read landmark point indexes from nodes2cal.out
4 $\mathbf{G}$ ← Read connectivity graph from grid.out
5 for $i \in \mathcal{N}^{Landmark}$ do
6     $\mathbf{D}_{:,i}$ ← run_dijsktra$(i,\mathbf{G})$
7     Write distances $\mathbf{D}_{:,i}$ into file dist_cpp.out
8 end
Output: File dist_cpp.out
```

Algorithm 3: Routine build_distance_matrix

First, the file dist_cpp.out is loaded in matrix $\mathbf{D}$ (line 2) and transformed into $\mathbf{B}$ (line 3). After this step, matrix $\mathbf{B}^T\mathbf{B}$ is factorized and the largest positive $k \leq n$ eigenvalues are selected, being n the number of landmark points. Finally, the embedding $\mathcal{Z}$ is defined as the rows of the matrix $\mathbf{Y}$ with columns $\lambda_i^{-1}\mathbf{B}\mathbf{v}_i$ for $i \in \{1,\dots,k\}$ (lines 5 and 6), being $\mathbf{v}_i$ the corresponding eigenvector of λ_i.

```
Input:
    dist_cpp.out: file with distance matrix values $\mathbf{D}$;
1 $(N,n)$ ← Read size $N$ of domain points $\Omega$ and size $n$ of landmark
  points $\Omega_L$ from dist_cpp.out
2 $\mathbf{D}$ ← Read distance matrix from dist_cpp.out with size $N \times n$
3 $\mathbf{B} \leftarrow \mathbf{H}_N\mathbf{A}\mathbf{H}_n$, with $a_{ij} = -\frac{1}{2}d_{ij}^2$, $d_{ij}$ $(i,j)$ value of $\mathbf{D}$ and $\mathbf{H}_k$ centering
  matrix of size $k$
4 $(\mathbf{\Lambda},\mathbf{V})$ ← Find the $k \leq n$ positive largest eigenvalues $\lambda_1 \geq \dots \geq \lambda_k > 0$
  of $\mathbf{B}^T\mathbf{B}$ with corresponding eigenvectors $\{\mathbf{v}_1,\dots,\mathbf{v}_k\}$ which satisfy
  $\left(\frac{\mathbf{B}\mathbf{v}_i}{\sqrt{\lambda_i}}\right)^T\left(\frac{\mathbf{B}\mathbf{v}_i}{\sqrt{\lambda_i}}\right) = \lambda_i$ for all $i \in \{1,\dots,k\}$
5 $\mathbf{Y} \leftarrow \left[\ \frac{1}{\lambda_1}\mathbf{B}\mathbf{v}_1 \ \middle|\ \cdots \ \middle|\ \frac{1}{\lambda_k}\mathbf{B}\mathbf{v}_k\ \right] \in \mathbb{R}^{N\times k}$
6 $\mathcal{Z} \leftarrow \{\mathbf{z}_1,\dots,\mathbf{z}_N\} \subset \mathbb{R}^k$ where $\mathbf{z}_i$ is the $i$-th row of $\mathbf{Y}$ for all
  $i \in \{1,\dots,N\}$. The following property holds:
  $\|\mathbf{z}_i - \mathbf{z}_j\|_2 = d_{ij}$, $\forall i \in \{1,\dots,N\}$, $\forall j \in \{1,\dots,n\}$ (after row and column
  reordering if necessary)
Output: Embedding $\mathcal{Z} \subset \mathbb{R}^k$, with $k \leq n$
```

Algorithm 4: Routine build_embedding

3.4. Sequential simulation routines

The final routines are related to the classical Sequential Simulation algorithm, as described in (Deutsch and Journel, 1998; Chilès and Delfiner, 1999). The random path $\mathbf{P}$ (line 6, Algorithm 1) represents a random re-ordering of the domain point indexes from 1 to $|\Omega|$. We will assume that the embedding $\mathcal{Z}$ is a subset of $\mathbb{R}^k$, with $k \leq n$ and n the number of landmark points. Regarding the neighbour search, represented by the routine search_neighbours (line 10), spiral search is implemented in the baseline non-LVA code, and two alternatives are implemented in the baseline LVA code, exhaustive search and KDTree-based search (Kennel, 2004), being the last one the most efficient in large-scale scenarios. The parameters of this routine are the domain point index $\mathbf{P}_{\mathtt{ixyz}}$, local interpolation parameters κ (such as maximum/minimum number of neighbours for further kriging interpolation, number of sample data values and previously simulated values, and maximum search distance) and a pseudo-random number generator seed τ. Additional parameters only for LVA codes are the embedding $\mathcal{Z}$ and LVA parameters represented as k^{search} and k^{cova}. A key parameter in this routine is the number of dimensions used for search k^{search}. This parameter controls which dimensions of the embedding $\mathcal{Z}$ will be used to perform distance comparisons to identify proximity. As discussed by Boisvert (2010), Boisvert and Deutsch (2011), reducing the number of dimensions in k^{search} can impact negatively in the accuracy of the obtained results, with a trade-off in speed of execution. Once the neighbours are computed, a simulation can be performed in the domain point, represented by the routine simulate (line 12). This routine is different in SGS, SGSIM and SISIM implementations. In SGS, the routine is based on the classical Fortran 90 GSLIB routine ktsol, which solves a universal kriging linear system through Gaussian elimination. In SGSIM, one execution of the GSLIB routine ksol is computed, which solves a kriging linear system. In SISIM, several executions of the GSLIB routine ksol are computed, one for each category (indicator) defined in the inputs, and solving a kriging linear system of equations each time. As with the previous search_neighbours routine, a key parameter in these routines is the number of dimensions used for covariance distance-based estimation k^{cova}, only used in LVA codes. Similarly to k^{search}, the reduction of this number impacts negatively in the accuracy of the obtained results with a trade-off in the execution time. The parameter k^{cova} controls which dimensions of the embedding $\mathcal{Z}$ will be used to compute distance differences for covariance models implemented also following the classical GSLIB routine cova3. Finally, simulate is executed S times per point (S stochastic simulations), using the same random path for all simulations (line 6). As mentioned in the beginning if this section, non-LVA codes SGSIM and SISIM does not work in this order, with the simulation loop placed outermost and using different random paths for each simulation. The final simulation results are stored in a matrix $\mathbf{V}^{tmp}$, which is saved in file output.txt.

4. Accelerated/parallel implementation

The first step that must be performed in order to accelerate or parallelize an application is to get an elapsed time profile of each part of the code. Based on that information, further code modifications are prioritized. In Tables 1 and 2 we can observe the percentage of elapsed time in each set of routines, using four scenarios denoted *sgsim*, *sisim*, *swiss-roll* and *escondida* (described in Section 5). For the non-LVA scenarios, *sgsim* and *sisim*, elapsed time spent on neighbours calculation and simulation can be splitted, since the baseline code already separates these tasks. We can observe that 60% and 26% correspond to neighbours calculation and 34% and 72% correspond to simulation, based on execution using 16 threads for parallel processing. For the LVA scenarios, no task separation is present in the code, so the major portion of elapsed time corresponds to the simulation routines (neighbour calculation + simulation) with 79% and 96% respectively, followed by the embedding and the distance matrix building. In the next subsections we will describe different strategies applied on these routines.

An initial code optimization step is applied to the LVA scenarios, since disk I/O communication and C++ code execution is performed in the L-ISOMAP routines. Software refactoring tasks are applied to the corresponding code in order to optimize the execution. The proposed refactoring changes are in favour of a unified in-memory execution (sequential and parallel) which improves performance, code development, debugging and allows future modifications more easily. These modifications which avoid launching other processes and communication through disk I/O are not described in this paper and they can be analysed by the reader directly in the available code.

As result of all refactor, acceleration and parallelization strategies applied, the proposed implementation can be reviewed in Algorithm 5. For non-LVA scenarios, steps 1 to 4 can be skipped, and also LVA parameters such as $\mathcal{Z}$, k^{search} and k^{cova} are not used. In the next sections, a detailed explanation of each aspect of this algorithm is included.

```
Input:
    Same parameters as Algorithm 1;
    $T$: number of parallel threads of execution;
    $b_1$: block size for parallel execution of neighbour search;
    $b_2$: block size for parallel execution of simulation;
1  //Only LVA: First calculate the embedding using L-ISOMAP
2  $\mathbf{G} \leftarrow$ build_connectivity_graph($\Omega, \mathbf{F}, \pi$)
3  $\mathbf{D} \leftarrow$ parallel_build_distance_matrix($\mathbf{G}, \Omega, \Omega_L$)
4  $\mathcal{Z} \leftarrow$ parallel_build_embedding($\mathbf{D}$)
5  //non-LVA and LVA: Then proceed with the simulation routines using the
   embedding to calculate distances
6  $\mathbf{P} \leftarrow$ create_random_path($\Omega, \tau$)
7  $\mathbf{V}^{tmp} \leftarrow$ zeros($|\Omega| \times S$)
8  $\mathbf{V}^{tmp} \leftarrow$ assign($\mathbf{V}$) //Initial conditioning data assignment
9  Level, Neighbours $\leftarrow$
   parallel_neighbour_search($|\Omega|, \mathbf{P}, \mathcal{Z}, k^{search}, n^{max}, T, b_1$)
10 $\mathbf{V}^{tmp} \leftarrow$
   parallel_simulation(Level, Neighbours, $|\Omega|, \mathbf{P}, \kappa, \tau, \mathcal{Z}, k^{cova}, n^{max}, T, b_2$)

11 write(output.txt, $\mathbf{V}^{tmp}$)
Output: S stochastic simulations stored in file output.txt
```

Algorithm 5: Parallel Sequential Simulation for non-LVA and LVA scenarios

4.1. Parallel neighbour search

Following the profiling from Tables 1 and 2, the first routine group that should be accelerated corresponds to neighbour search and sequential simulation routines as described in Section 3. Code modifications are applied in lines 9 to 14 of Algorithm 1, in case decoupling of both parts is needed (LVA scenarios), and subsequently accelerate each with different strategies. The framework used in this work follows an exact path-level parallelization of non-LVA codes based on Peredo et al. (2018), which was adapted to the LVA codes, as shown in Algorithm 5.

In case of neighbour search routines, the proposed parallelization is based on a combination of: an optimized version of sequential KDTree neighbour search from Kennel (2004), and a parallel implementation presented in Algorithm 6, which is used in line 9 of Algorithm 5. These search routines were implemented on the four codes in study, non LVA-based SGSIM and SISIM, and LVA-based SGS and SISIM.

In LVA-based SGS, the neighbour search can be applied using KDTree-based search originally. However, in LVA-based SISIM and non LVA-based SGSIM and SISIM, the only option implemented in the original baseline code was the exhaustive search or spiral search. Thus in these cases the first task was to adapt the original code to include the KDTree method. KDTree search is algorithmically faster than exhaustive search ($O(\log N)$ for KDTree-based search and $O(N)$ for exhaustive search), so no further analysis was performed for the last method. Spiral search on the other hand, as described in Deutsch and Journel (1998), is an efficient method to search for neighbours in gridded data, however its sequential nature makes it difficult to parallelize it efficiently.

In the existing implementation of KDTree search, the inner-most part of the computation should calculate distances between the query point and all points inside a terminal node of the tree. The points that are inside a fixed-size ball around the query points are marked as neighbours until the maximum number is reached. Using the line profiler tool of `gprof` (Graham et al., 2004), we identify lines in KDTree code which are top contributors in the sequential part of execution. The optimization applied is based on the unrolling of the loop that computes the squared distance between the query point and each potential neighbour, which reduces the number of conditional evaluations and branch instructions processed by the CPU.

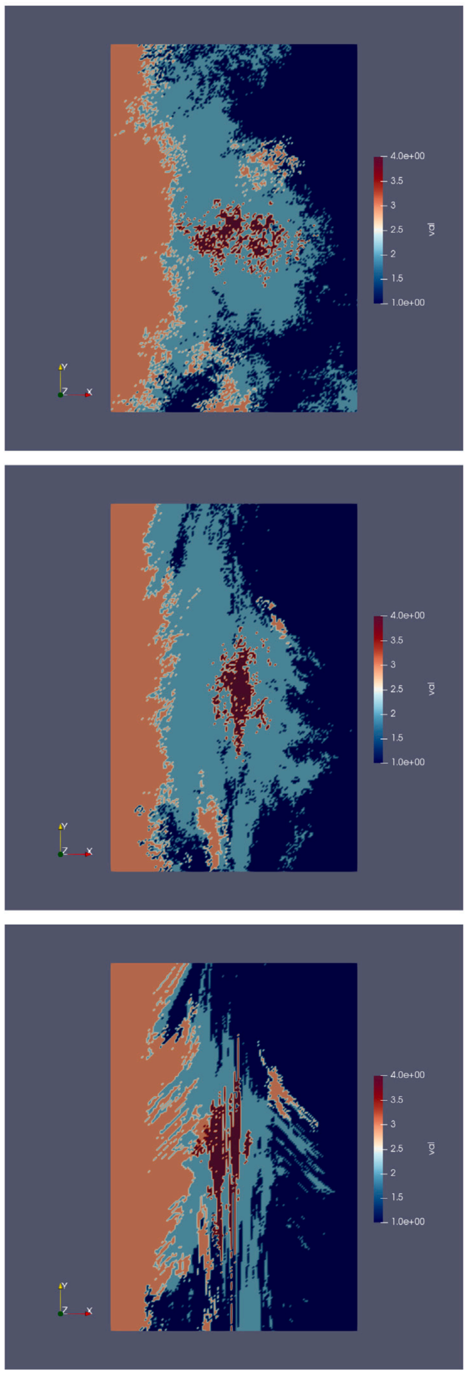
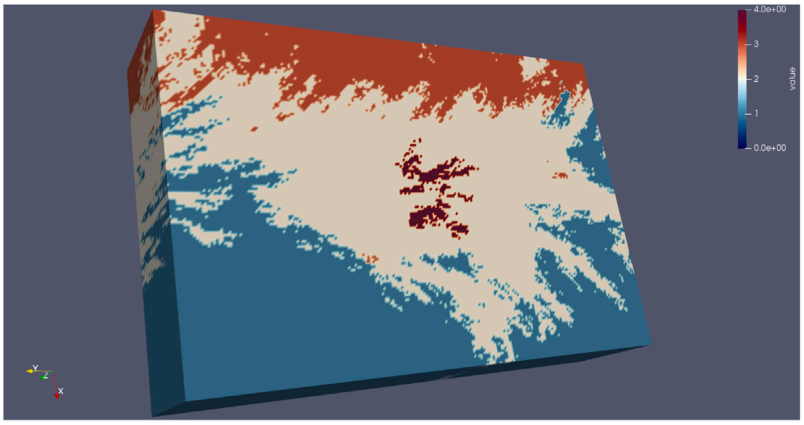


Fig. 10. Slices of simulated domains for *escondida* scenario using parallel LVA-based SISIM with different r_1 ratio values from LVA field parameters.

The parallel implementation is based on a modified OpenMP-compliant version of KDTree search and a block cyclic decomposition strategy of the random path. The block cyclic approach is necessary since an underlying unbalance exists in the amount of work for neighbour search (early points require more effort than later points). In order to use the existing KDTree implementation, a private variable used internally in the corresponding Fortran module should be declared as `threadprivate` with an OpenMP directive. With this change, different threads can create private trees and search independently for neighbours on different points without sharing data structures (line 6 of Algorithm 6). Since the neighbour search should be compliant with the sequential simulation search, only previously



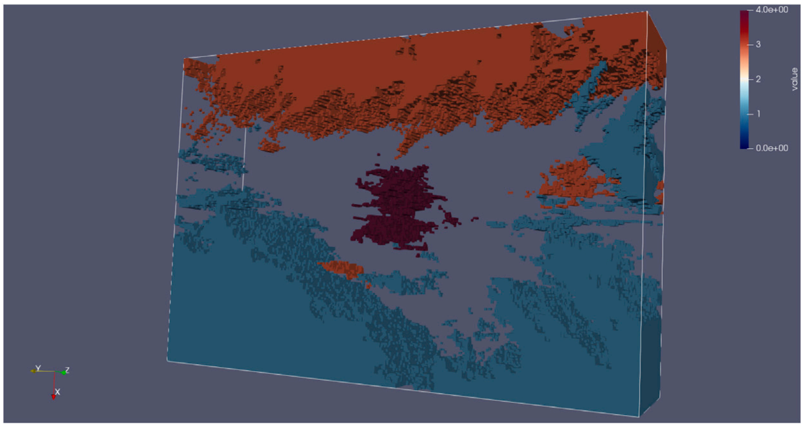


Fig. 11. Top: Simulated domain for *escondida* scenario using parallel LVA-based SISIM. Bottom: Three categories of the simulation on top.

Table 1
Profiling of executions [% of elapsed time] with baseline non-LVA codes. Left: *sgsim* scenario using baseline non LVA-based SGSIM with 50×10^6 domain points, 48 maximum neighbours for kriging and 16 threads, total elapsed time was 11 min and 25 s. Right: *sisim* scenario using baseline non LVA-based SISIM with 50×10^6 domain points, 48 maximum neighbours for kriging and 16 threads, total elapsed time was 37 min and 21 s.

Execution step	$\%t_{total}$ (*sgsim*)	$\%t_{total}$ (*sisim*)
Read params	1.311	0.004
Neighbours calculation	68.965	26.092
Simulation	24.535	72.216
Write out	5.187	0.012

simulated nodes (*marked* points) or initial conditioning data can be considered as valid neighbours. This constraint is applied in lines 12 to 14 of Algorithm 6, by setting as *marked* all previous nodes for each thread of execution. By combining this strategy with a block cyclic distribution of iterations, the final workload is balanced through all threads, as shown in Fig. 1. The final steps, depicted in lines 23 to 27 of Algorithm 6, use the computed neighbours to infer the *level* of each point, which indicates the degree of dependency of that point on previously simulated or initial conditioning points. Specifically, initial conditioning points are level 0 and a point is level n if the maximum level of all its neighbours is $n - 1$.

In Section 5 we include performance tests of this specific algorithm applied to non-LVA and LVA scenarios.

4.2. Parallel sequential simulation

As mentioned in the previous section, code modifications are applied in lines 9 to 14 of Algorithm 1, in order to decouple neighbour search and simulation parts for LVA scenarios. Regarding acceleration of the simulation part, an exact path-level parallelization is applied, following an exact path-level strategy described in Peredo et al. (2018). We refer the reader to that article for further details of the existing parallel algorithm, but essentially it allows to simulate in parallel every point located in the same level of the random path, as described in the previous section.

The steps of this strategy applied to LVA-based SGS and SISIM are depicted in Algorithm 7. The reader can refer to Peredo et al. (2018) for further reference of the inner routines and arrays used in this algorithm.

4.3. Parallel embedding building

The second group of routines with large percentage of elapsed time is related to the assembling of the embedding $\mathcal{Z}$, which is only relevant for LVA scenarios. After the software refactoring was completed, code optimizations can be applied properly on this part. By examining Algorithm 4, several matrix algebraic operations are involved in these routines. In order to accelerate them, Intel Math Kernel Library (Intel, 2020) was selected as optimized implementation for Matrix-Vector (DGEMV), Matrix-Matrix (DGEMM) and Eigenvalue solver computation (DSYEV). Additionally, several memory accesses were modified from row-major to column-major order, which is the optimal memory access pattern in Fortran code.

4.4. Parallel distance matrix building

Regarding the computation of the distance matrix, also relevant for LVA scenarios only, several executions of Dijkstra's shortest path Boost implementation are launched. These executions are orchestrated by the routine `dijkstra_cpp`, integrated in the source after the software

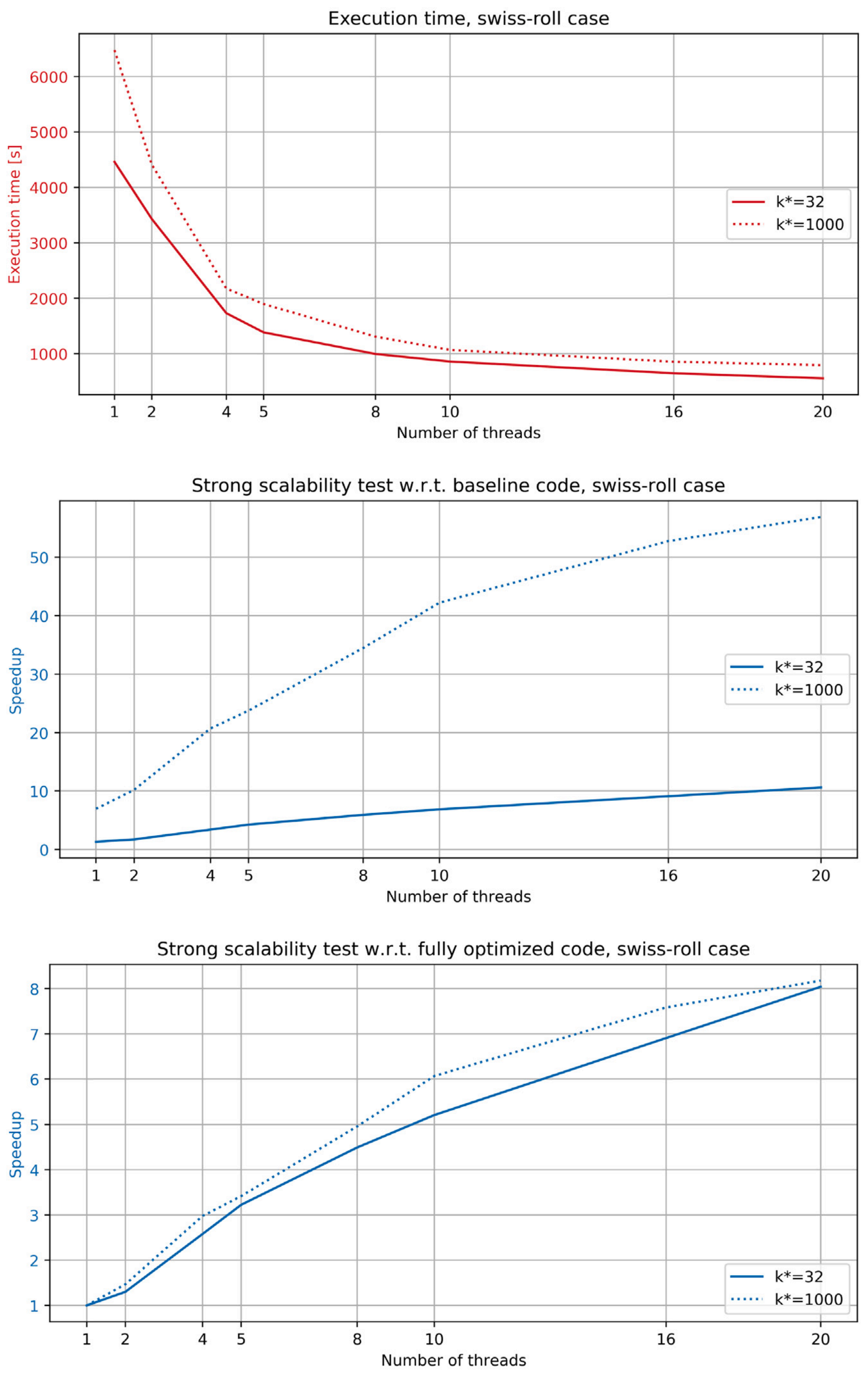


Fig. 12. Execution time [seconds] and speedup results for *swiss-roll* scenario using parallel LVA-based SGS code.

Table 2
Profiling of executions [% of elapsed time] with baseline LVA codes. Left: *swiss-roll* scenario using baseline LVA-based SGS with 1.7×10^6 domain points, 48 maximum neighbours for kriging and 1000 landmarks, total elapsed time was 12 h and 31 min. Right: *escondida* scenario using baseline LVA-based SISIM with 1.7×10^6 domain points, 48 maximum neighbours for kriging and 1344 landmarks, total elapsed time was 509 h and 17 min (21 days and 5 h).

Execution step	$\%t_{total}$ $k^{cova} = k^{search} = 1000$ (*swiss-roll*)	$\%t_{total}$ $k^{cova} = k^{search} = 1344$ (*escondida*)
Read params	0.012	0.001
Connectivity graph building	0.233	0.001
Distance matrix building	10.341	2.178
Embedding building	9.831	5.134
Neighbours calculation + Simulation	79.576	96.585
Write out	0.006	0.001

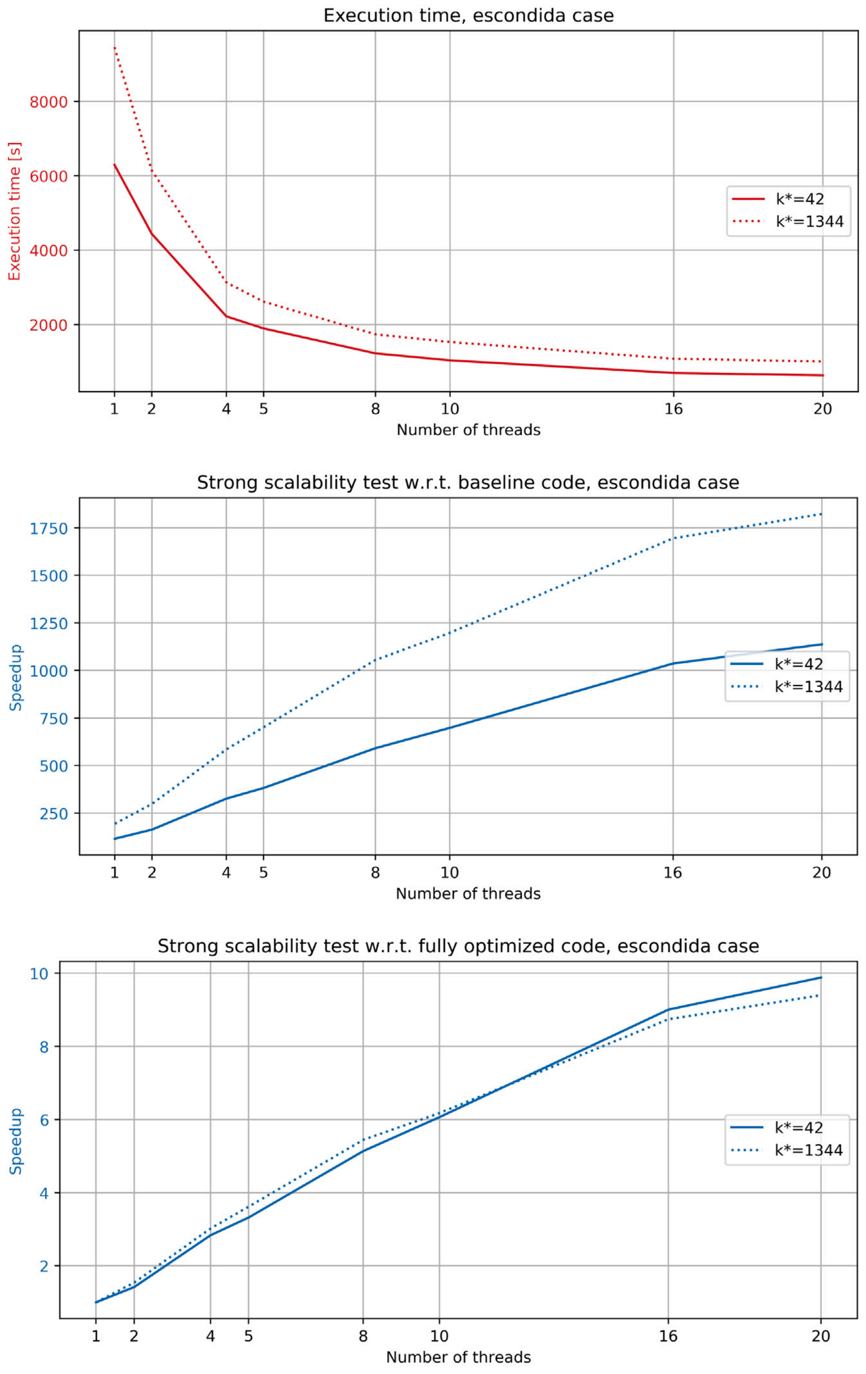


Fig. 13. Execution time [seconds] and speedup results for *escondida* scenario using parallel LVA-based SISIM code.

refactoring. Since C++ code can be parallelized with OpenMP (OpenMP Architecture Review Board, 2008) in a straightforward way, we add parallel pragmas to this code, in order to parallelize the landmark loop of lines 5 to 7 of Algorithm 3.

5. Results

This section is divided in two subsections. The first one shows performance tests of the proposed implementation from Section 4.1 using two scenarios extracted from Peredo et al. (2018), with adapted parallel versions of non LVA-based SGSIM and SISIM respectively. In the second subsection, performance tests of the proposed implementation are presented for two LVA-based scenarios, using parallel LVA-based versions of SGS and SISIM codes.

5.1. *Performance tests for parallel non LVA-based codes*

In order to measure the performance of the proposed parallel algorithm, simulations are generated from non LVA-based codes SGSIM and SISIM. Both scenarios are denoted *sgsim* and *sisim* respectively. Scenario *sgsim* uses an initial 3D dataset of 2376 diamond drill-hole samples with information of copper grade composites. Scenario *sisim* uses a synthetic 3D dataset of 3000 random samples with 10 categories. The parameters in each scenario can be viewed in Table 3.

All runs were executed in a single-node machine with Ubuntu 18.04.5 LTS with 2 × 10-cores Intel(R) Xeon(R) CPU Silver 4210R at frequency 2.40 GHz and a main memory of 128 GB RAM. All Fortran programs were compiled using GNU Fortran version 4.8.5 supporting OpenMP version 3.1, with options -cpp -O2 -ffast-math -ftree-vectorize.

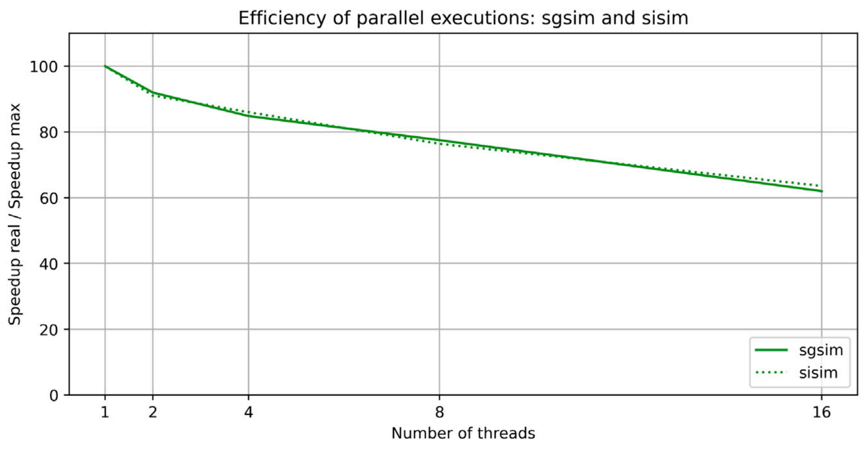


Fig. 14. Efficiency of parallel executions for *sgsim* and *sisim* scenarios compared against theoretical maximum speedup. In this case a maximum of 16 threads are used in order to compare results with a previous work from Peredo et al. (2018).

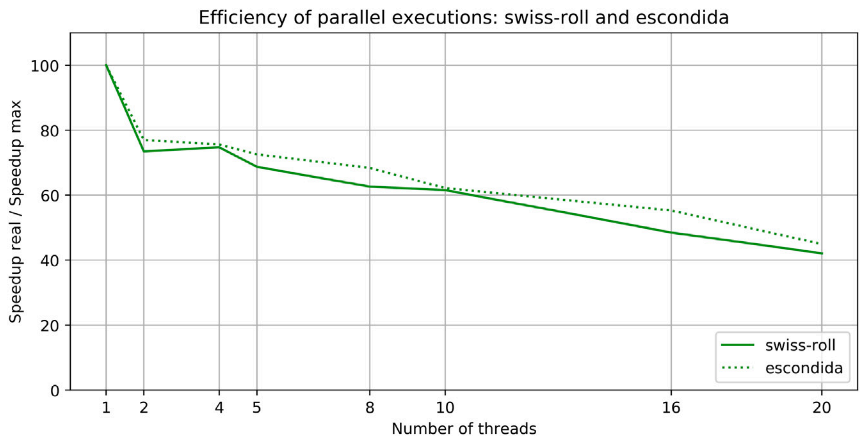


Fig. 15. Efficiency of parallel executions for *swiss-roll* and *escondida* scenarios compared against theoretical maximum speedup.

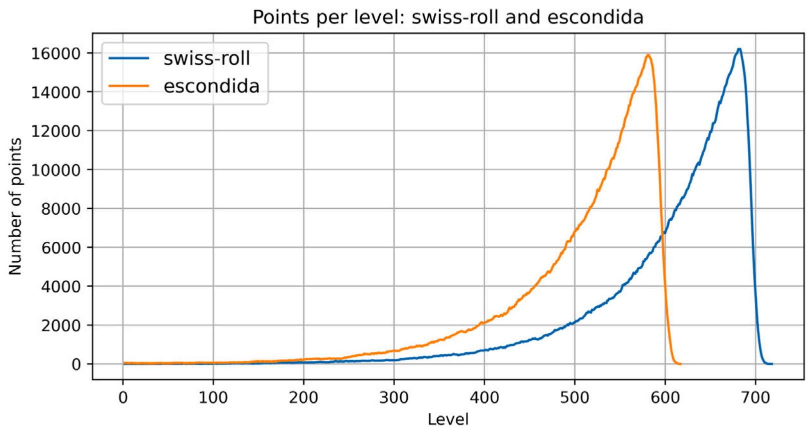


Fig. 16. Points per level on both test scenarios, *swiss-roll* (396 initial conditioning data) and *escondida* (2313 initial conditioning data). All points in the same level are simulated in parallel by P threads.

Execution time results are depicted in Figs. 2 and 3, and speedup results are depicted in Figs. 4 and 5.

In *sgsim* scenario depicted in Fig. 4 we can observe that the speedup increased consistently across all tests, with improvements using 16 threads (this number of threads is selected in order to compare results with previous work from the same authors). From Fig. 2, speedups using 16 threads between the proposed implementation against the baseline version without parallel neighbour search are $1.33\times$, $1.79\times$, $1.85\times$ and $2.14\times$ in cases with 16, 32, 48 and 64 maximum neighbours for simulation respectively. However, it is important to notice that using lower numbers of neighbours, such as 16 and 32, the execution time is considerably lower in the baseline scenario using less than 8 and 4 threads of execution respectively. The reason for this behaviour is the amount of work that needs to be done to initialize the KDTree parallel structures, which in these cases is higher than the baseline search methods (the reader can check `srchsupr` method from Deutsch and Journel (1998)). On all other cases the parallel adaptation has lower execution time than the baseline code.

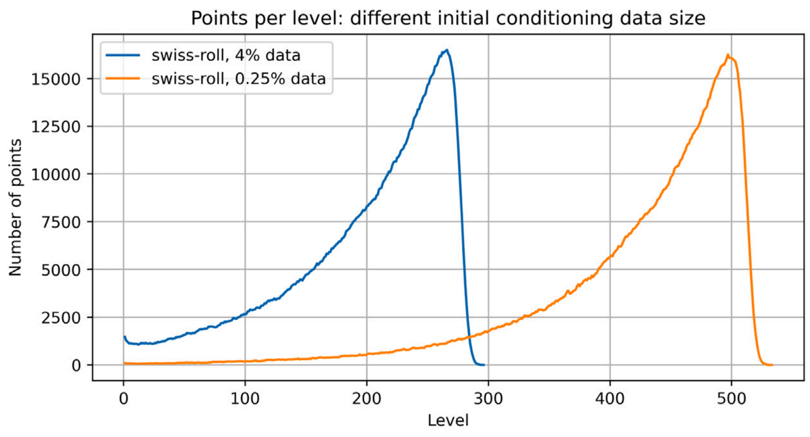


Fig. 17. Points per level on *swiss-roll* scenario using different percentages of initial conditioning data (4% and 0.25%, from a total of 1,728,000 points).

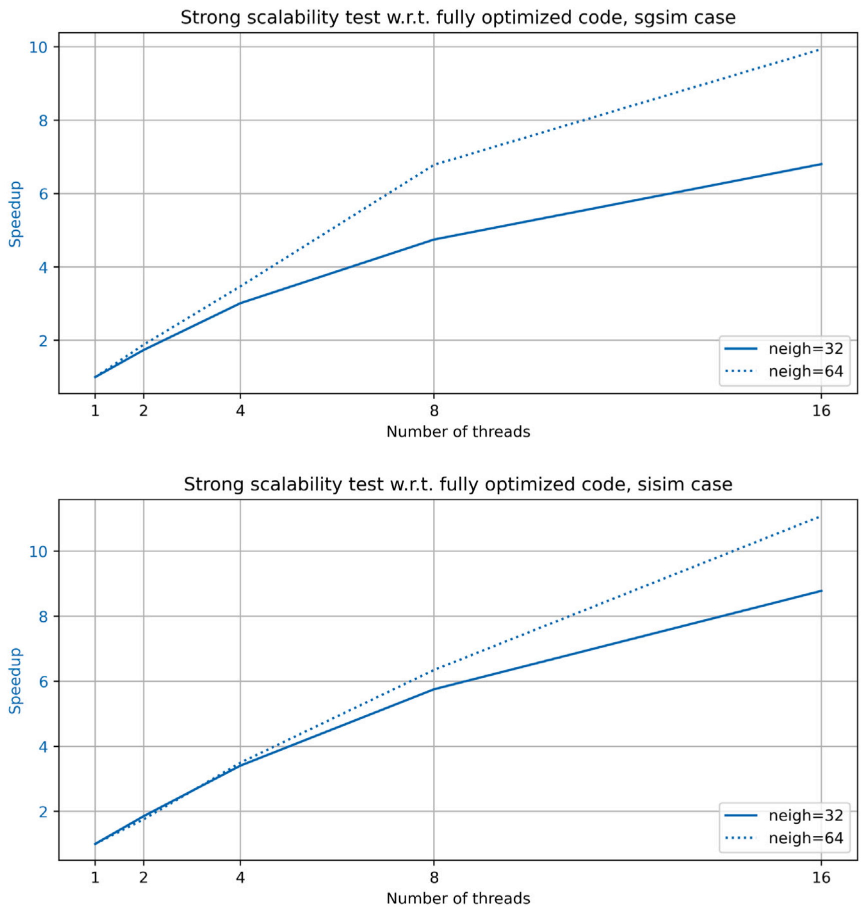


Fig. 18. Speedup results for scenarios *sgsim* and *sisim* using 32 and 64 maximum number of neighbours.

In the *sisim* scenario depicted in Fig. 5 we can observe no significative differences in speedup trends between the baseline and parallel adaptation. From Fig. 3, speedups using 16 threads between the proposed implementation against the baseline version without parallel neighbour search are 1.55×, 2.42×, 3.06× and 4.11× in cases with 16, 32, 48 and 64 maximum neighbours for simulation respectively. In this scenario the execution time is consistently lower in the proposed implementation. This can be explained since this scenario involves more work (10 kriging linear systems should be solved for each domain point versus only one linear system in *sgsim*), so the initialization of KDTree structures is not significative in the execution time.

Finally, if we compare the aggregated contribution to speedup of the new parallel neighbour search and optimizations, plus the previous parallelization work from Peredo et al. (2018), against a single thread execution of the previous parallelization work, the obtained speedups using 16 threads for *sgsim* scenario are 2.2×, 5.0×, 7.6× and 11.9×, using 16, 32, 48 and 64 maximum neighbours respectively. Similarly for *sisim* scenario, speedups using 16 threads are 7.8×, 20.3×, 32.7× and 50.4×, using 16, 32, 48 and 64 maximum neighbours respectively.

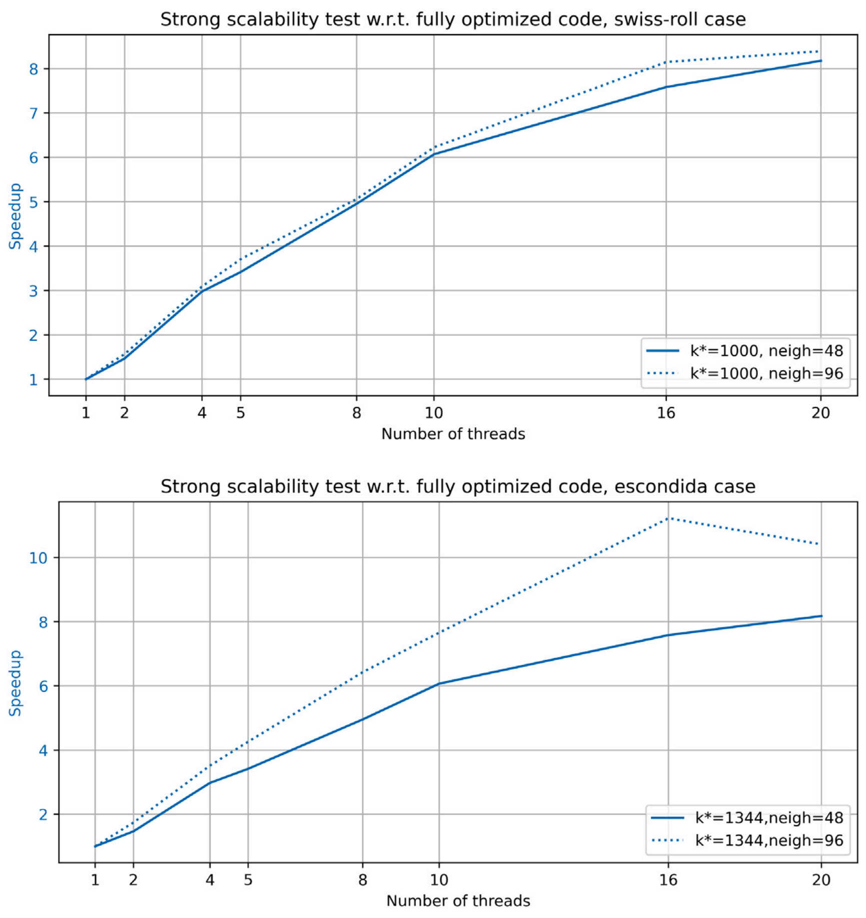


Fig. 19. Speedup results for scenarios *swiss-roll* and *escondida* using 48 and 96 maximum number of neighbours.

5.2. Performance tests for parallel LVA-based codes

The first scenario, namely *swiss-roll*, is based in the swiss roll testing scenario, which is extensively used in the Machine Learning community (Tenenbaum et al., 2000). In our case, a 3D swiss roll is prepared, which is posteriorly transformed into a 3D LVA field. A synthetic dataset of 17280 samples is designed and attached to the domain as sample data. The second scenario, namely *escondida*, is based on real mining 3D data of 2376 diamond drill-hole samples with information of copper grade composites and lithologies per sample. In this case, a synthetic fold-like LVA field is used for simulation. The parameters of each scenario are detailed in Table 4. A schema of the LVA fields and the drillhole data are depicted in Figs. 6 and 7. In Figs. 8 and 9 we can observe simulated domains in the *swiss-roll* scenario using LVA-based SGS. In the first figure, 6 slices of simulated domains are presented, each one generated with different values of r_1 ratio. The second figure shows two 3D simulated domains generated with LVA fields with different r_1 ratio values. In Figs. 10 and 11 we can observe simulated domains in the *escondida* scenario using LVA-based SISIM, with 4 categorical values. As in the previous figures, the first one shows 3 slices of simulated domains each one generated with different values of r_1 ratio. The second one shows a 3D simulated domain generated with an LVA field.

All runs were executed in a single-node machine with Ubuntu 18.04.5 LTS with 2 × 10-cores Intel(R) Xeon(R) CPU Silver 4210R at frequency 2.40 GHz and a main memory of 128 GB RAM. All Fortran programs were compiled using Intel ifort version 13.1.1 supporting OpenMP version 3.1, with options -fpp -mkl -openmp -O3 -mtune=native -march=native. All C++ programs were compiled using GCC g++ version 6.2.0 supporting OpenMP version 4.5, with options -Ofast -fopenmp -funroll-loops -finline-functions -ftree-vectorize.

Fig. 12 shows execution time and speedup of the LVA-based SGS parallel code for the *swiss-roll* scenario, using $k^{cova} = k^{search} = 32$ and $k^{cova} = k^{search} = 1000$ control dimensions. Fig. 13 shows execution time and speedup of the LVA-based SISIM parallel code for the *escondida* scenario, using $k^{cova} = k^{search} = 42$ and $k^{cova} = k^{search} = 1344$ control dimensions. On both figures, speedup is computed using the baseline sequential code (middle plot) and fully optimized code (bottom plot). Differences in speedup values on both plots, can be explained by large differences in elapsed time of the original baseline code and the optimized single threaded execution of the OpenMP code. In case of SISIM, this effect reduces three orders of magnitude the baseline execution time, mostly due to the refactoring of neighbour search using KDTree instead of exhaustive search.

Finally, to illustrate the contribution of the neighbour search acceleration and parallelization to the overall speedup, in Table 5 we can observe execution time and speedup against the baseline code version for LVA-based SGS and SISIM using $k^* = 1000$ and $k^* = 1344$ respectively. The purpose of this information is to show the relevance of the acceleration and parallelization of the neighbour search, which allows to improve the speedup an order of magnitude for each scenario. Special relevance has the inclusion of KDTree search in LVA-based SISIM which delivered a single contribution of two orders of magnitude in a sequential execution.

6. Analysis

According to the results of Section 5, two aspects of the proposed implementation are reviewed in detail in this section: accuracy and efficiency.

```
Input:
    Levels: array defined in Algorithm 5 (line 9);
    Neighbours: array defined in Algorithm 5 (line 9);
    $|\Omega|$: number of points in 3D domain $\Omega$;
    P: random path defined in Algorithm 5 (line 6);
    $\mathcal{Z}$: embedding of point coordinates as defined in Algorithm 5
 (line 4). For non-LVA scenarios, the embedding is exactly the 3D
 coordinates in $\Omega$;
    $k^{search}$: defined as input in Algorithm 1. For non-LVA scenarios,
 this value is 3;
    $n^{max}$: defined as input in Algorithm 1;
    $T$: number of parallel threads of execution;
    $b$: block size
$n \leftarrow |\Omega|$
Level $\leftarrow$ zeros($n \times 1$)
Neighbours $\leftarrow$ zeros($n \times n^{max} \times 2$)
for threadId $\in \{1, \dots, T\}$ in parallel do
    $B \leftarrow \lceil n/b \rceil$
    //Each thread will have a copy of its own KDTree structure
    Tree $\leftarrow$ kdtree_create($\mathcal{Z}, n^{max}, k^{search}$)
    //Block cyclic through blocks
    nlast $\leftarrow 1$
    for iblock $\in \{1, \dots, B\}$ do
        nmin, nmax $\leftarrow (b-1) * B$ , $\min\{b * B, n\}$
        if iblock % $T$ == (threadId $- 1$) then
            for ixyz $\in$ {nlast, ..., nmin $- 1$} do
                point_marked(Tree, ixyz)
            end
            nlast $\leftarrow$ nmax $+ 1$
            //Each thread computes the valid neighbours of each point
            for ixyz $\in$ {nmin, ..., nmax} do
                //Search and push operation to save neighbours in array
                Neighbours($P_{ixyz}$) $\leftarrow$
                 search_neighbours_push($P_{ixyz}, \kappa, \mathcal{Z}$, Tree, $k^{search}$)
            end
        end
    end
    omp_barrier //Only thread 1 will compute the levels (intrinsically
     sequential)
    if threadId $= 1$ then
        for ixyz $\in \{1, \dots, n\}$ do
            Level($P_{ixyz}$) $\leftarrow$ build_level($P_{ixyz}, \kappa$, Neighbours)
        end
    end
end
Output: Neighbours, Levels
```

Algorithm 6: Routine `parallel_neighbour_search` (KDTree-based)

```
Input:
    Levels: array defined in Algorithm 5 (line 9);
    Neighbours: array defined in Algorithm 5 (line 9);
    $|\Omega|$: number of points in 3D domain $\Omega$;
    P: random path defined in Algorithm 5 (line 6);
    $\kappa$: defined as input in Algorithm 1;
    $\mathcal{Z}$: embedding of point coordinates as defined in Algorithm 5
 (line 4). For non-LVA scenarios, the embedding is exactly the 3D
 coordinates in $\Omega$;
    $k^{cova}$: defined as input in Algorithm 1. For non-LVA scenarios,
 this value is 3;
    $n^{max}$: defined as input in Algorithm 1;
    $T$: number of parallel threads of execution
IndexSort, LevelCount, LevelStart $\leftarrow$
 order_nodes_by_level(Level)
for lev $\in \{1, \dots, max$(Level)$\}$ do
    lbegin $\leftarrow$ LevelStart(lev)
    lend $\leftarrow$ LevelStart(lev) + LevelCount(lev) $- 1$
    for ixyz $\in$ {lbegin, ..., lend} in parallel do
        index $\leftarrow$ IndexSort($P_{ixyz}$)
        //Pop operation to extract neighbours from array
        LocalNeighbours $\leftarrow$
         search_neighbours_pop(index, $\kappa$, Neighbours)
        for isim $\in \{1, \dots, S\}$ do
            $V^{tmp}(P_{ixyz}$, isim) $\leftarrow$
             simulate($P_{ixyz}$, LocalNeighbours, $\mathcal{Z}, k^{cova}, n^{max}$)
        end
    end
    write(output.txt, $V^{tmp}$)
end
Output: S stochastic simulations stored in file output.txt
```

Algorithm 7: Algorithm `parallel_simulation`

6.1. Accuracy

In terms of accuracy, all parallel codes match exactly the baseline results (assuming the same pseudo-random number generator seed and the same neighbour search method), regardless of the number of threads used in the execution. This level of accuracy is obtained as result of the explicit replication of the random path simulations, although re-ordering non-conflicting nodes in order to allow parallel simulation of nodes in the same level.

Particularly for LVA-based codes, by decreasing the values of control variables k^{search} and k^{cova} (which define the number of dimensions to use for neighbour search and covariance distance calculation), both parallel and baseline codes can achieve faster results, but with lower accuracy values (compared against larger values of control variables). It will be a final user's decision if he or she can tolerate lower levels of accuracy. In case of LVA-based SGS (Fig. 12), the accuracy loss comparing executions with $k^* = 1000$ versus $k^* = 32$ is 71%, measured as the average relative error at node level . In case of LVA-based SISIM (Fig. 13), the accuracy loss comparing executions with $k^* = 1344$ versus $k^* = 42$ is 15%, measured as the average categorical mismatch at node level. Nonetheless, the proposed parallel codes deliver the same results as the baseline for any fixed value of the control variables. Note that using the new parallel codes, simulations are executed much faster. Therefore, performing computations for low values of the control variables in search of a reduction of the execution time becomes meaningless when the accuracy loss is high.

6.2. Efficiency

In terms of the achieved efficiency by the proposed parallelization, in Tables 6 and 7 we can observe a detailed profile of the refactored codes using 16 OpenMP threads in *sgsim* and *sisim*, and 20 OpenMP threads in *swiss-roll* and *escondida* scenarios, all executions using 48 maximum neighbours for simulation, and $k^* = 1000$ and $k^* = 1344$ respectively on each LVA-based scenario. It is worth mentioning that these tables are very different from the initial baseline profiling in Tables 1 and 2, where the most relevant part was the neighbour calculation plus simulation as a whole. Now, the relevant parts for non LVA-based scenarios are neighbours calculations for SGSIM and again simulation for SISIM. For LVA-based scenarios, the relevant parts are distance calculation, neighbours calculation and simulation (embedding building is not relevant after applying optimizations in matrix operations and memory accesses). On non LVA-based scenarios, both results have efficiencies of 62% and 63% respectively, measured as the percentage of the theoretical maximum speedup achieved. On LVA-based scenarios, both results have efficiencies of 42% and 44% respectively. These results are acceptable for these applications since the baseline code and algorithms were not designed originally to run on

Table 3
Default parameters for *sgsim* and *sisim*.

Parameter	*sgsim*	*sisim*
Domain $nx \times ny \times nz$	$400 \times 400 \times 120$	$210 \times 600 \times 400$
Max nodes for simulation	$\{16, 32, 48, 64\}$	$\{16, 32, 48, 64\}$
Kriging	OK	OK
Number of structures (type)	3 (spherical, exponential, gaussian)	10 (sphericals)

Table 4
Default parameters for *swiss-roll* and *escondida*.

Parameter	*swiss-roll*	*escondida*
Domain $nx \times ny \times nz$	$120 \times 120 \times 120$	$148 \times 220 \times 52$
LVA field $nx \times ny \times nz$	same as domain	same as domain
Graph connectivity (offset)	1	1
Landmarks $nx \times ny \times nz$	$10 \times 10 \times 10$	$8 \times 12 \times 14$
k^{cova}	$\{32, 64, 125, 250, 500, 1000\}$	$\{42, 84, 168, 336, 672, 1344\}$
k^{search}	$\{32, 64, 125, 250, 500, 1000\}$	$\{42, 84, 168, 336, 672, 1344\}$
Max nodes for simulation	48	48
Kriging	SK	SK
Number of structures (type)	1 (exponential)	4 (exponentials)

Table 5
Contribution to speedup of neighbour search (NS) acceleration and parallelization on the *swiss-roll* and *escondida* scenarios. The initial parallel code did not include parallel neighbour search, only calculation of distance matrix, embedding and simulation were parallelized. Additionally, for LVA-based SISIM, KDTree was adapted and included for execution.

Version	*swiss-roll* $k^* = 1000$ time [seconds]	*swiss-roll* $k^* = 1000$ speedup	*escondida* $k^* = 1344$ time [seconds]	*escondida* $k^* = 1344$ speedup
Baseline	45060	1×	1833020	1×
Parallel except NS, 1 thread	12132	3.71×	16438	111.51×
Parallel except NS, 20 threads	8615	5.23×	8596	213.24×
Parallel except NS + KDTree optimized, 20 threads	1682	26.77×	1648	1111.91×
Parallel including NS + KDTree optimized, 20 threads	792	56.89×	1006	1822.08×

Table 6
Profiling of executions [% of elapsed time] with parallel refactored non-LVA codes using 16 threads. Left: *sgsim* scenario using accelerated non LVA-based SGSIM with 50×10^6 domain points, 48 maximum neighbours for kriging, total elapsed time was 6 min and 10 s. Right: *sisim* scenario using accelerated non LVA-based SISIM with 50×10^6 domain points, 48 maximum neighbours for kriging, total elapsed time was 21 min and 26 s.

Execution step	$\%t_{total}$ (*sgsim*)	$\%t_{total}$ (*sisim*)
Read params	1.807	0.101
Neighbours calculation	46.240	15.111
Simulation	43.325	84.395
Write out	8.695	0.391

Table 7
Profiling of executions [% of elapsed time] with parallel refactored LVA codes using 20 OpenMP threads. Left: *swiss-roll* scenario using accelerated LVA-based SGS with 1.7×10^6 domain points, 48 maximum neighbours for kriging and 1000 landmarks, total elapsed time was 1 h 47 min. Right: *escondida* scenario using accelerated LVA-based SISIM with 1.7×10^6 domain points, 48 maximum neighbours for kriging and 1344 landmarks, total elapsed time was 2 h 37 min.

Execution step	$\%t_{total}$ $k^{cova} = k^{search} = 1000$ (*swiss-roll*)	$\%t_{total}$ $k^{cova} = k^{search} = 1344$ (*escondida*)
Read params	0.03	0.01
Connectivity graph building	0.09	0.06
Distance matrix building	61.81	51.40
Embedding building	3.99	4.59
Neighbours calculation	15.20	10.22
Simulation	18.84	33.71
Write out	0.05	0.01

parallel architectures. Additionally, Figs. 14 and 15 contain efficiency percentages for non LVA-based scenarios for values of $P \leq 16$ and LVA-based scenarios for values of $P \leq 20$.

The achieved speedup and efficiency obtained on both scenarios can be explained mostly by three factors: 1) intrinsic efficiency of external libraries (C++ Boost and Intel MKL), 2) different sizes in the initial conditioning datasets, and 3) amount of work performed in the simulation step.

Factor 1 impacts only on LVA-based scenarios, and depends explicitly on the performance delivered by the external libraries. Even though further optimizations can be done in these libraries, it is left out of the scope of this work. In any case, on both scenarios we obtained similar performance since these executions only depend on the number of domain points, number of landmarks and LVA additional parameters, which remain on the same orders of magnitude across scenarios.

Regarding factor 2, also impacting only on LVA-based scenarios, in Fig. 16 we can observe the number of points assigned to each level, using a maximum of 48 neighbour points per simulation for both scenarios. In the *swiss-roll* scenario, 396 initial conditioning points are used, which results in a larger point-level curve, and translates into more overhead (resulting in less speedup and efficiency) due to multi-thread contingency and context switching from level to level. On the contrary, in the *escondida* scenario, 2313 initial conditioning points are used, which results in a shorter curve, and consequently less overhead (resulting in more speedup and efficiency). We can infer that a large number of initial conditioning data will generate a short point-level curve with better speedup and efficiency at lower execution time. In Fig. 17 we can observe different point-level curves for the same scenario with equal parameters, except the number of initial conditioning data points. Curves using 48 maximum neighbours and 4% and 0.25% of initial conditioning data (continuous blue and orange curves) exemplify this phenomenon.

Regarding factor 3, which impacts on both non LVA and LVA-based scenarios, we can identify two cases: keeping constant versus increasing the maximum number of neighbours to use for simulation. Assuming the maximum number of neighbours does not change, the amount of work in the simulation step can be increased by assembling and solving more kriging linear systems per simulation point. This situation occurs on non LVA-based and LVA-based SISIM implementations, since for each category, a kriging linear system should be assembled and solved per each simulation point. In *sisim* scenario, 10 categories are simulated, which result in $10\times$ more work per simulation compared against *sgsim* scenario. In *escondida* scenario, 4 categories are simulated, which results in $4\times$ more work per simulation point compared against *swiss-roll* scenario. The increment of work per simulation point does not change the form of the point-level curve, but it will impact on performance by delivering better speedup and efficiency values, at higher execution time. Multi-threaded execution becoming more efficient as the problem becomes larger (more computing needed) is a well-known behaviour denoted *weak scalability* (the reader can refer to the definition of Gustafson's law from Hennessy and Patterson (2012) or the original reference from Gustafson (1988)). On the other hand, if the maximum number of neighbours increases, the amount of work in the simulation step will be increased automatically, since the size of the kriging linear systems will increase accordingly. Fig. 18 shows the speedups obtained using 32 and 64 neighbours as the maximum values for *sgsim* and *sisim* scenarios. Similarly, Fig. 19 shows the equivalent using 48 and 96 as the maximum for the *swiss-roll* and *escondida* scenarios. On all figures we can observe an increment of the speedup values when higher number of neighbours are used. A slight decline in the speedup trend can be observed only for *escondida* using 96 neighbours, which is caused by overhead in the execution due to multicore contingency and higher memory usage overall.

7. Conclusions and future work

The proposed implementation is able to speed-up the execution using code optimizations and allowing parallel execution using OpenMP. Compared against the sequential baseline codes, using 16 OpenMP threads it delivers speedups of $12\times$ and $50\times$ for non LVA-based SGSIM and SISIM respectively, and using 20 OpenMP threads it delivers speedups of $56\times$ and $1822\times$ for LVA-based SGS and SISIM respectively. Results for SISIM codes were obtained by also refactoring the baseline code neighbour search to allow KDTree-based search. The code of KDTree search was also optimized to reduce overhead due to large number of conditional evaluations and branching in the CPU. On LVA-based codes, by decreasing the parameters k^{cova} and k^{search}, the execution speed can be increased, with a trade-off in the accuracy obtained. In any case, the proposed implementation obtains the same results as the baseline implementation, regardless of the number of parallel threads used in the execution. This level of accuracy is possible since the same random path of simulation is used on both implementations, preserving neighbours calculations and the order of simulated values (assuming the same pseudo-random number generator seeds and neighbour search method). This is exactly the main purpose of using an exact path-level approach for parallelization, as discussed in previous sections.

We expect that this implementation will allow many researchers and practitioners to improve their tasks in stochastic resource simulations. Since the code will be released publicly, every aspect can be modified and improved by the community of interested users and institutions.

In terms of future work and next steps, two aspects will be explored: performance and usability improvements. In the first aspect, approximate computing approaches will be explored in order to reduce the execution time keeping statistically similar results. Additionally, improved techniques to accelerate the distance matrix building on LVA scenarios. In the second aspect, usability improvement can be achieved by allowing Python or R wrappers to the main execution parts of the code.

8. Source code

The current version of the code is available in the following links
https://github.com/operedo/parallel-sgs-lva
https://github.com/operedo/parallel-sisim-lva

CRediT authorship contribution statement

Oscar F. Peredo: Study, Conception, Design, Acquisition/generation of data, Analysis and interpretation of results, Parallel design, Writing – original draft, Writing – review & editing. **José R. Herrero:** Parallel design, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors acknowledge the donated resources from project PID2019-107255GB of the Spanish *Ministerio de Economía y Competitividad*, and project 2017-SGR-1414 from the Generalitat de Catalunya, Spain.

References

Boisvert, J.B., 2010. Geostatistics with locally varying anisotropy. (Ph.D. thesis). University of Alberta.
Boisvert, J.B., Deutsch, C.V., 2011. Programs for kriging and sequential Gaussian simulation with locally varying anisotropy using non-euclidean distances. Comput. Geosci. 37 (4), 495–510.
Boost.org, 2012. Boost C++ libraries. URL http://www.boost.org.
Chilès, J.-P., Delfiner, P., 1999. Geostatistics : Modeling Spatial Uncertainty. In: Wiley series in probability and statistics, Wiley, New York, URL http://opac.inria.fr/record=b1098313A Wiley-Interscience publication.
Curriero, F.C., 2006. On the use of non-euclidean distance measures in geostatistics. Math. Geol. 38 (8), 907–926.
Deutsch, C., Journel, A., 1998. GSLIB: Geostatistical Software Library and User's Guide, second ed. In: Applied geostatistics series, Oxford Univ. Press, New York, NY.
Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numer. Math. 1, 269–271.
Graham, S.L., Kessler, P.B., McKusick, M.K., 2004. Gprof: A call graph execution profiler. SIGPLAN Not. 39 (4), 49–57. http://dx.doi.org/10.1145/989393.989401.
Gustafson, J.L., 1988. Reevaluating Amdahl's law. Commun. ACM 31 (5), 532–533.
Gutierrez, R., Ortiz, J., 2019. Sequential indicator simulation with locally varying anisotropy – simulating mineralized units in a porphyry copper deposit. J. Min. Eng. Res. 1 (1), 1–7. http://dx.doi.org/10.35624/jminer2019.01.01.

Hennessy, J.L., Patterson, D.A., 2012. Computer Architecture: A Quantitative Approach, fifth ed. Morgan Kaufmann, Amsterdam.

Huo, X., Ni, X.S., Smith, A.K., 2004. A survey of manifold-based learning methods. In: Recent Advances in Data Mining of Enterprise Data: Algorithms and Applications. pp. 691–745. http://dx.doi.org/10.1142/9789812779861_0015.

Intel, 2020. Math kernel library. URL https://software.intel.com/en-us/mkl.

Isaaks, E.H., Srivastava, R.M., 1990. an Introduction to Applied Geostatistics. Oxford University Press, USA.

Kennel, M.B., 2004. KDTREE 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional euclidean space. arXiv:physics/0408067.

Nunes, R., Almeida, J.A., 2010. Parallelization of sequential Gaussian, indicator and direct simulation algorithms. Comput. Geosci. 36 (8), 1042–1052.

Nussbaumer, R., Mariethoz, G., Gravey, M., Gloaguen, E., Holliger, K., 2018. Accelerating sequential Gaussian simulation with a constant path. Comput. Geosci. 112, 121–132. http://dx.doi.org/10.1016/j.cageo.2017.12.006.

OpenMP Architecture Review Board, 2008. OpenMP application program interface version 3.0. URL http://www.openmp.org/mp-documents/spec30.pdf.

Peredo, O., Baeza, D., Ortiz, J.M., Herrero, J.R., 2018. A path-level exact parallelization strategy for sequential simulation. Comput. Geosci. 110, 10–22.

Peredo, O., Ortiz, J.M., Herrero, J.R., 2015. Acceleration of the geostatistical software library (GSLIB) by code optimization and hybrid parallel programming. Comput. Geosci. 85, 210–233, Part A.

Rasera, L.G., Machado, P.L., Costa, J.F.C., 2015. A conflict-free, path-level parallelization approach for sequential simulation algorithms. Comput. Geosci. 80, 49–61.

Tenenbaum, J.B., de Silva, V., Langford, J.C., 2000. A global geometric framework for nonlinear dimensionality reduction. Science 290 (5500), 2319.

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