A Path-Level Exact Parallelization Strategy for Sequential Simulation

Oscar F. Peredo[a,d], Daniel Baeza[b], Julián M. Ortiz[c], José R. Herrero[a]

[a] *Computer Architecture Department, UPC-BarcelonaTech, Spain*
[b] *Advanced Laboratory for Geostatistical Supercomputing, University of Chile, Chile*
[c] *The Robert M. Bucham Department of Mining, Queen's University, Canada*
[d] *Telefónica I+D, Chile*

Abstract

Sequential Simulation is a well known method in geostatistical modelling. Following the Bayesian approach for simulation of conditionally dependent random events, Sequential Indicator Simulation (SIS) method draws simulated values for K categories (categorical case) or classes defined by K different thresholds (continuous case). Similarly, Sequential Gaussian Simulation (SGS) method draws simulated values from a multivariate Gaussian field. In this work, a path-level approach to parallelize SIS and SGS methods is presented. A first stage of rearrangement of the simulation path is performed, followed by a second stage of parallel simulation for non-conflicting nodes. A key advantage of the proposed parallelization method is to generate identical realizations as with the original non-parallelized methods. Case studies are presented using two sequential simulation codes from GSLIB: SISIM and SGSIM. Execution time and speedup results are shown for large-scale domains, with many categories and maximum kriging neighbours in each case, achieving high speedup results in the best scenarios using 16 threads of execution in a single machine.

1. Introduction

Geostatistical simulation provides an approach to quantify uncertainty over spatially distributed variables. Several methods are available depending on the properties of the random function considered.

For continuous variables, most methods are based on a multiGaussian assumption, reducing the inference problem to finding the mean and covariance function for pairs of points. Uncertainty quantification is easily solved with kriging (Journel & Alabert, 1989; Deutsch & Journel, 1998; Chilès & Delfiner, 1999), which linearly infers the conditional expectation and conditional variance of the Gaussian random variables, requiring the two-point spatial covariance function, which in turn can be inferred from available data. In the same context, Sequential Gaussian Simulation (SGS) (Alabert, 1987; Isaaks, 1990) is one the most straightforward methods for generating stochastic realizations of multivariate Gaussian random fields.

An alternative approach to multiGaussian methods is offered by sequential indicator simulation (Alabert, 1987), where spatial correlation can be tailored to show different behaviors for different thresholds and also provides a flexible framework to integrate secondary variables and trends (Zhu & Journel, 1993). This provides flexibility, but also brings additional challenges related to order relations, spatial correlation inside bins defined by the thresholds and spatial correlation between simulated values from different bins (Machuca-Mory et al., 2008).

The case of categorical variables is particularly suited for high variability deposits where transitions between facies show low correlation. Alternative methods based on truncation of Gaussian random fields, namely Truncated Gaussian and PluriGaussian simulation (Matheron et al., 1987), offer more flexibility to reproduce these transitions Deutsch (2006), but are not as flexible when dealing with secondary variables and trends (Yarus et al., 2012). SIS has been applied to the geological modelling of ore deposits (Dimitrakopoulos, 1998; Dimitrakopoulos & Dagbert, 1993; Journel & Isaaks, 1984; de Souza & Costa, 2013) and oil reservoirs (Dubrule & Damsleth, 2001; Pan, 1997; dell'Arciprete et al., 2012; de Almeida, 2010), as well as in other fields such as rock fractures modelling (Dowd et al., 2007), imaging (van der Meer, 1994), and soil science (Bierkens & Burrough, 1993; Goovaerts, 2001), to name a few. The use of two-point statistics is clearly a limitation that can be overcome through multiple-point

simulation (MPS) (Daly & Caers, 2010; Bastante et al., 2008; Ortiz & Deutsch, 2004; Caers, 2005) to improve the reproduction of the transition between classes defined by a set of thresholds and, when dealing with categorical variables, the connectivity of the geobodies. However, MPS methods require a training image to infer these patterns statistics. These methods have a historical importance in terms of large-scale simulations and novel implementations, as will be reviewed in subsequent paragraphs.

In terms of performance, parallel and distributed computational techniques can decrease the execution time of the methods by increasing the number of operations per cycle through multi-thread or multi-process execution. In this context, using the taxonomy defined by Mariethoz (2010), three levels of parallelization can be developed for sequential simulation codes: realization-level, path-level and node-level. The straightforward approach is the realization-level, where each realization is performed by different operating system processes or threads by changing appropriately the pseudo-random seed or other structural parameters in each run. Peredo et al. (2015) and Navarro et al. (2014) applied this approach to the SISIM and SGSIM routines from GSLIB (Deutsch & Journel, 1998). Path-level parallelization is based on the partition of the domain into zones that can be handled by different processes or threads. Rasera et al. (2015), based on the strategy proposed by Vargas et al. (2007), developed a novel path-level conflict-free parallelization for the SGS method with promising future results in the SIS context. In the same context, Mariethoz (2010) and posteriorly Tahmasebi et al. (2012) developed a master-slave strategy that distributes the grid nodes among several processors, using a multi-core cluster or a graphical unit processor (GPU). Regarding the node-level parallelization, where the computation of each grid node is parallelized, two-point (Nunes & Almeida, 2010) and multi-point statistics (Straubhaar et al., 2011; Peredo & Ortiz, 2011, 2012; Peredo et al., 2014) strategies have been proposed.

In this work, a path-level strategy of parallelization is presented. The main idea of the strategy is to group all nodes having no conflicting neighborhood following the original path, such that, all nodes of the same group can be simu-

lated simultaneously. In the next section the non-parallel sequential simulation algorithm is described, using as base the implementation of SGSIM and SISIM from GSLIB depicted in Deutsch & Journel (1998). In Section 3 the path-level parallelization is described. In Section 4 case studies are presented using SISIM and SGSIM codes, with their time execution and speedup results. Finally in Section 5 the limitations and advantages of the parallelization are presented, with some ideas to improve the strategy in the future.

2. Non-parallel algorithm

The main concept of the sequential simulation family of algorithms is based on the Bayes postulate applied to a joint probability distribution of several dependent variables (Devroye, 1986; Johnson, 1987). The successive application of Bayes postulate to the joint probability leads to a sequential backward inference of marginals and posterior distributions, monotonically increasing the size of the prior data set as different grid nodes are randomly visited and simulated.

In Algorithm 1 the main steps of the method can be viewed. These steps are synthesized from SGSIM and SISIM codes from GSLIB. For each realization `isim`, a simulation path $\mathcal{P}$ visiting all nodes of the domain Ω is randomly generated (pseudo-routine `create_random_path`). Following the simulation path, each node `index` of the domain is simulated. Both simulation methods use different inner routines to draw the simulated value, for simplicity we refer to these steps as `searchNeighbours` and `simulate`. Some of the most important parameters of the simulation steps are related with the local neighbourhood to use for interpolation defined globally by the parameter κ. In this work, we consider that this parameter contains the maximum and minimum number of neighbours for interpolation, the number of sample data values and previously simulated values, the size of search lookup window, and neighbour offset indices, according to the specific search startegy selected. The neighbourhood search strategy, such as super-block or spiral search, as described in (Deutsch & Journel, 1998), will be highly affected by the previous parameters, affecting the overall performance of

the simulation algorithm. In this work we consider only the spiral search, since the alternative method implemented in the GSLIB simulation routines is computationally expensive (two-step search running first a super-block search for nearby data and then execute the spiral search on previously simulated nodes). In line 6 of Algorithm 1 the pseudo-routine `spiral_search_neighbours` is executed, running the original GSLIB implementation of the spiral search. The local neighbourhood information is stored in the structure **LocalNeighbourhood**, consisting in the coordinates, indices, variable values and other information of the neighbours inside the search lookup window. In line 7 the simulation is performed in the pseudo-routine `simulate`, by calculating one or more local interpolations centered in the grid node $\mathcal{P}_{ixyz}$ using the local neighbourhood information obtained in the previous step.

Algorithm 1 Pseudo-code sequential simulation program (single-thread algorithm)

Input: $(\mathbf{V}, \Omega)$: sample database values $\mathbf{V}$ defined in a 3D domain Ω; κ: local interpolation parameters; τ: seed for pseudo-random number generator; N: number of generated simulations; `output.txt`: output file name

Output: N stochastic simulations stored in file `output.txt`

1: **for** `isim` $\in \{1, \ldots, N\}$ **do**
2: $\quad \mathcal{P} \leftarrow$ `create_random_path`(Ω, τ)
3: $\quad \mathbf{V}^{tmp} \leftarrow$ `zeros`$(|\Omega| \times 1)$
4: $\quad \mathbf{V}^{tmp} \leftarrow$ `assign`$(\mathbf{V})$ *//Sample data assignment*
5: $\quad$ **for** `ixyz` $\in \{1, \ldots, |\Omega|\}$ **do**
6: $\quad\quad$ **LocalNeighbours** $\leftarrow$ `spiral_search_neighbours`$(\mathcal{P}_{\texttt{ixyz}}, \kappa, \tau)$
7: $\quad\quad \mathbf{V}^{tmp}(\mathcal{P}_{\texttt{ixyz}}) \leftarrow$ `simulate`$(\mathcal{P}_{\texttt{ixyz}}, \mathbf{LocalNeighbours})$
8: $\quad$ **end for**
9: $\quad$ `write`$(\texttt{output.txt}, \mathbf{V}^{tmp})$
10: **end for**

3. Parallel algorithm

3.1. Main algorithm

The parallel version of the method is presented in Algorithm 2. It is based in two stages, the first one related to node tagging in order to group all nodes with non conflicting neighbourhoods. The second stage is the actual simulation, similarly to the single-thread algorithm, with a different node loop formulation but the same neighbourhood data and simulation method. The pseudo-routines with their steps detailed are `spiral_search_neighbours_push` (Algorithm 3), `build_level` (Algorithm 4), `order_nodes_by_level` (Algorithm 5) and `spiral_search_neighbours_pop` (Algorithm 6).

Regarding the first stage, in steps 7 and 8 of Algorithm 2 two arrays are defined, **Level** and **Neighbours**, which will store the level tags and neighbours information (local and global indices). A first pass through all nodes is performed between steps 9 and 12. The simulation path is walked sequentially, scanning for neighbours around the current node and storing basic information about them in the pseudo-routine `spiral_search_neighbours_push` (Algorithm 3). This routine is essentially the same as the original spiral search from GSLIB, with the only difference that, instead of actually calculating the coordinates and other information about the neighbours, it only stores the neighbours indices by *pushing* (copying) them into the array **Neighbours**. After the neighbours have been calculated, a level tag to the current node is assigned according to the pseudo-routine `build_level` which scans for the maximum of all neighbours level tags and adds 1 to that value (Algorithm 4). Initially all nodes with conditioning data are assigned with level tag 0 and non informed nodes are assigned with level tag -1. With this initial assignment, nodes with some conditioning data inside their search lookup window are assigned with level tag 1, nodes with a level 1 neighbour are assigned with level tag 2, and so on. The last part of the first stage is at step 13 of Algorithm 2, where the pseudo-routine `order_nodes_by_level` performs a rearrangement procedure, storing the indices of the new order in the array **IndexSort**, and the number of nodes and initial

index per level in the arrays **LevelCount** and **LevelStart** (Algorithm 5).

In Figure 1 an example of the level assignment is presented using a search lookup window of size 3×3. Initially the conditioning data nodes are placed in the locations 6, 13, 15, 18, 24 and 25 of the random path (value in the top-right of each grid cell). The level tag for those conditioning nodes is zero. Starting the assignment, the node in location 1 is visited resulting in a level tag assignment of 1, since in its search lookup window of 3×3 there are only nodes with level tags of 0 (neighbours in locations 13, 15 and 24). Similarly, nodes in locations 2, 3 and 4 are assigned with level tag 1. Node in location 5 is assigned with level tag 2, since in its search lookup window a neighbour with level tag 1 is located (neighbour in location 4). Node in location 7 is assigned with level tag 1 (neighbour in locations 6 and 25 with level 0), and node in location 8 is assigned with level tag 3 (neighbour in location 5 with level 2), and so on.

The second stage of Algorithm 2 involves the simulation in parallel of all nodes in the same level, since no data dependencies arise between those nodes. For each level, as shown in step 15, the initial and final indices are calculated, `lbegin` and `lend` respectively in steps 16 and 17. The index of the node to be simulated is obtained in step 19 using the re-ordered array **IndexSort**. In step 20 the pseudo-routine `spiral_search_neighbours_pop` (Algorithm 6) is called, which essentially is a query to extract local neighbour indices from the array **Neighbours**, previously stored by using `spiral_search_neighbours_push` in step 10 (Algorithm 3). With the local neighbour indices, the coordinates are computed for each neighbour, and the simulation can be performed in step 21 with the pseudo-routine `simulate`, as the single-thread original algorithm.

In this work, OpenMP directives (OpenMP Architecture Review Board, 2008) are included into the modified code. A synchronization method must be used in order to keep the order of the levels being processed, since threads can spend different time in the simulation of their assigned nodes, causing race conditions when accessing neighbour values that are being simulated or not simulated yet. A first alternative is to use the implicit OpenMP barrier declared at the end of a parallel loop region. Since this barrier adds a major overhead to

Algorithm 2 Pseudo-code sequential simulation program (multi-thread algorithm)

Input: $(\mathbf{V}, \Omega)$: sample database values defined in a 3D domain; κ: local interpolation parameters; τ: seed for pseudo-random number generator; N: number of generated simulations; `output.txt`: output file name; T: number of threads

Output: N stochastic simulations stored in file `output.txt`

1: $n \leftarrow$ `obtain_max_neighbour_number`(κ)
2: **for** `isim` $\in \{1, \ldots, N\}$ **do**
3: $\quad \mathcal{P} \leftarrow$ `create_random_path`(Ω, τ)
4: $\quad \mathbf{V}^{tmp} \leftarrow$ `zeros`$(|\Omega| \times 1)$
5: $\quad \mathbf{V}^{tmp} \leftarrow$ `assign`$(\mathbf{V})$ *//Sample data assignment*
6: $\quad$ *//Stage 1*
7: $\quad$ **Level** $\leftarrow$ `zeros`$(|\Omega| \times 1)$
8: $\quad$ **Neighbours** $\leftarrow$ `zeros`$(|\Omega| \times n \times 2)$ *//store the neighbours local and global index*
9: $\quad$ **for** `ixyz` $\in \{1, \ldots, |\Omega|\}$ **do**
10: $\quad\quad$ **Neighbours**$(\mathcal{P}_{\texttt{ixyz}}) \leftarrow$ `spiral_search_neighbours_push`$(\mathcal{P}_{\texttt{ixyz}}, \kappa)$
11: $\quad\quad$ **Level**$(\mathcal{P}_{\texttt{ixyz}}) \leftarrow$ `build_level`$(\mathcal{P}_{\texttt{ixyz}}, \kappa,$ **Neighbours**$)$
12: $\quad$ **end for**
13: $\quad$ **IndexSort**, **LevelCount**, **LevelStart** $\leftarrow$ `order_nodes_by_level`$($**Level**$)$
14: $\quad$ *//Stage 2*
15: $\quad$ **for** `lev` $\in \{1, \ldots,$ `max`$($**Level**$)\}$ **do**
16: $\quad\quad$ `lbegin` $\leftarrow$ **LevelStart**$($`lev`$)$
17: $\quad\quad$ `lend` $\leftarrow$ **LevelStart**$($`lev`$)$ + **LevelCount**$($`lev`$) - 1$
18: $\quad\quad$ **for** `ixyz` $\in \{$`lbegin`$, \ldots,$ `lend`$\}$ **in parallel do**
19: $\quad\quad\quad$ `index` $\leftarrow$ **IndexSort**$(\mathcal{P}_{\texttt{ixyz}})$
20: $\quad\quad\quad$ **LocalNeighbours** $\leftarrow$ `spiral_search_neighbours_pop`$($`index`$, \kappa,$ **Neighbours**$)$
21: $\quad\quad\quad$ $\mathbf{V}^{tmp}(\mathcal{P}_{\texttt{ixyz}}) \leftarrow$ `simulate`$(\mathcal{P}_{\texttt{ixyz}},$ **LocalNeighbours**$)$
22: $\quad\quad$ **end for**
23: $\quad$ **end for**
24: $\quad$ `write`$($`output.txt`$, \mathbf{V}^{tmp})$
25: **end for**

the parallelization, a second alternative was chosen based on lock variables that control when all neighbour nodes of a node being simulated are available (have a defined value). A pseudo-code of this strategy is depicted in Algorithm 7, using an extra shared array **Lock** with size $|\Omega|$ and values 1 or 0 indicating if the corresponding grid node has been simulated or not. As the neighbour node indices

are collected, each thread waits until all neighbours have lock value **Lock**(`i`)=1, in order to get out of the waiting loop and continue with the simulation steps.

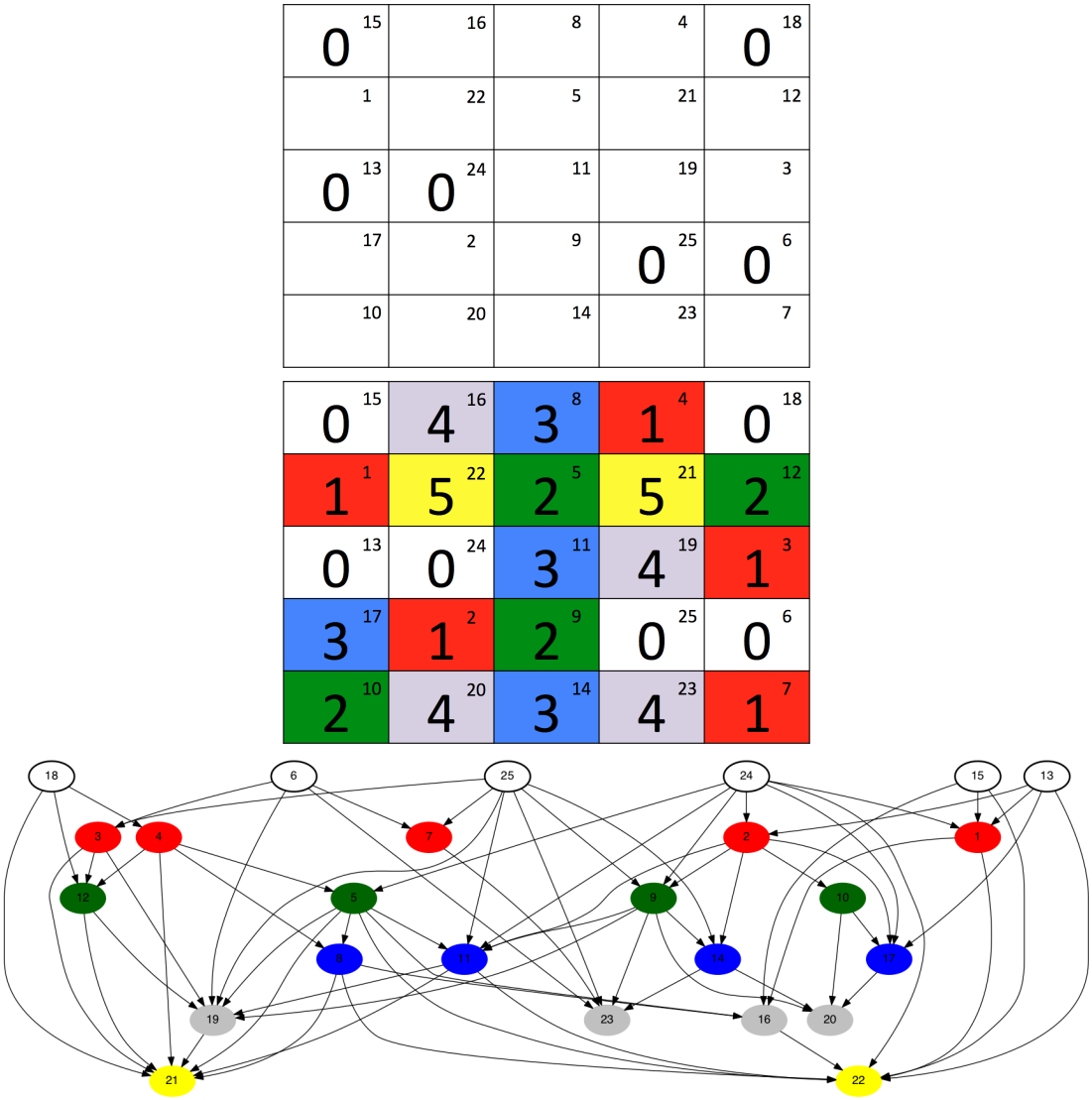

Figure 1: Top: Random path index (top-right corner or each cell) and initial assignment of level tags (only zeros for nodes with conditioning data). Middle: Final assignment of level tags, with different color for different levels. The search lookup window in this example is a 3×3 square centered in the node of interest. By walking through the random path and scanning the max level tag in each window, adding 1 to it, the final assignment of levels can be obtained. Bottom: Data dependency graph associated with the level tags and neighbour relationships.

Algorithm 3 Pseudo-routine `spiral_search_neighbours_push`

Input: `index`: grid node index; κ: local interpolation parameters.

Output: **Neighbours**(`index`): array with neighbour indices for grid node `index`.

$m \leftarrow$ `obtain_search_window_size`(κ)
$n \leftarrow$ `obtain_max_neighbour_number`(κ)
Offset $\leftarrow$ `obtain_spiral_search_offsets`(κ)
`numberOfLocalNeighbours` $\leftarrow 0$
for `ind` $\in \{1, \ldots, m\}$ **do**
  If `numberOfLocalNeighbours` $\geq n$ **then Return**
  $(i, j, k) \leftarrow (\texttt{index}_x, \texttt{index}_y, \texttt{index}_z) +$ **Offset**(ind) // *spiral node visiting centered in node* `index`
  if (i, j, k) is inside the domain **then**
    `indexglobal` $\leftarrow i + (j-1) * nx + (k-1) * nx * ny$
    if Node `indexglobal` has been previously simulated **then**
      Neighbours(`index`) $\leftarrow$ {`ind`, `indexglobal`}) //*local and global indexes*
      `numberOfLocalNeighbours` $\leftarrow$ `numberOfLocalNeighbours` $+ 1$
    end if
  end if
end for

Algorithm 4 Pseudo-routine `build_level`

Input: `index`: grid node index; κ: local interpolation parameters; **Neighbours**: array with neighbour indices of all domain nodes; **Level**: array with level tags of all domain nodes.

Output: **Level**(`index`): level assigned to grid node `index`.

// *Obtain the number of valid neighbours for node* `index`
`numberOfLocalNeighbours` $\leftarrow$ `obtain_total_neighbour_number`(**Neighbours**(`index`))
`maxLevel` $\leftarrow -1$
Level(`index`) $= 0$
for `ind` $\in \{1, \ldots,$ `numberOfLocalNeighbours`$\}$ **do**
  if Level(**Neighbours**(`index`, `ind`, 1)) $>$ `maxLevel` **then**
    `maxLevel` $\leftarrow$ **Level**(**Neighbours**(`index`, `ind`, 1)) //*obtain level of neighbour using global index*
  end if
end for
Level(`index`) $=$ `maxLevel` $+ 1$

Algorithm 5 Pseudo-routine `order_nodes_by_level`

Input: **Level**: array with level tags of all domain nodes.

Output: **IndexSort**, **LevelCount**, **LevelStart**: arrays with the reordering of node visits using the level tag as grouping identifier.

```
numberOfLevels ← max(Level)
count ← 0
lastCount ← 0
LevelCount ← zeros((numberOfLevels + 1) × 1)
LevelStart ← zeros((numberOfLevels + 1) × 1)
for lev ∈ {0, ..., numberOfLevels} do
    LevelStart(lev + 1) ← count + 1
    LevelCount(lev + 1) ← 0
    for ixyz ∈ {1, ..., |Ω|} do
        if Level(ixyz) == lev then
            count ← count + 1
            IndexSort(count) ← ixyz
            LevelCount(lev + 1) ← LevelCount(lev + 1) + 1
        end if
    end for
end for
```

Algorithm 6 Pseudo-routine `spiral_search_neighbours_pop`

Input: `index`: grid node index; κ: local interpolation parameters; **Neighbours**: array with neighbour indices.

Output: **LocalNeighbours**: neighbour coordinates of grid node `index`.

```
// Obtain the number of valid neighbours for node index
numberOfLocalNeighbours ← obtain_total_neighbour_number(Neighbours(index))
for ind ∈ {1, ..., numberOfLocalNeighbours} do
    indexglobal ← Neighbours(index, ind, 0) //indexglobal is neighbour global index
    LocalNeighbours(index, ind) ← get_coordinates(indexglobal)
end for
```

Algorithm 7 Pseudo-code for thread synchronization using locks.

for `lev` $\in \{1, \ldots,$ `max`(**Level**)$\}$ **do**
  `lbegin` $\leftarrow$ **LevelStart**(`lev`)
  `lend` $\leftarrow$ **LevelStart**(`lev`) + **LevelCount**(`lev`) $- 1$
  for `ixyz` $\in \{$`lbegin`$, \ldots,$ `lend`$\}$ **in parallel do**
    LocalNeighbours $\leftarrow$ `spiral_search_neighbours_pop`($\mathcal{P}_{\texttt{ixyz}}, \kappa,$ **Neighbours**)
    // Wait until all local neighbours are ready with a defined value, ***Lock*** *is shared array*
    `ilock` $\leftarrow 0$
    while `ilock` $== 0$ **do**
      `ilock` $\leftarrow 1$
      for $i \in \{1, \ldots,$ `numberOfLocalNeighbours`$\}$ **do**
        `ilock` $\leftarrow$ `ilock` $*$ **Lock**(**Neighbours**($\mathcal{P}_{\texttt{ixyz}}, i, 1$)) //global index
      end for
    end while
    // Proceed to simulate the current node
    $\mathbf{V}^{tmp}(\mathcal{P}_{\texttt{ixyz}}) \leftarrow$ `simulate`($\mathcal{P}_{\texttt{ixyz}},$ **LocalNeighbours**)
    Lock($\mathcal{P}_{\texttt{ixyz}}$) $= 1$ *//Mark this node as unlocked for all other nodes waiting for it*
  end for
end for

4. Results

In this section, two cases are presented, including their execution times, speedup and maximum theoretical speedup. The base codes are SGSIM and SISIM from GSLIB, developed by Deutsch & Journel (1998), and posteriorly code-optimized by Peredo et al. (2015). On both codes, a code optimization was applied on the routine that performs neighbourhood search, pseudo-routine `spiral_search_neighbours_push`. The source code can be viewed in the corresponding link of Section 6.

All runs were executed in a single-node machine with Ubuntu 14.04.5 LTS with 2×8-cores Intel(R) Xeon(R) CPU E5-2673 v3 at frequency 2.40GHz, and a memory hierarchy of 116GB RAM, 30MB L3 cache, 256KB L2 cache and 32KB/32KB L1d/L1i cache. All programs were compiled using GCC `gfortran` version 4.8.4 supporting OpenMP version 3.1, with the options `-O2 -march=native -ffast-math -ftree-vectorize` in all cases and `-fopenmp` in the multi-thread executions. All results are the average value of 5 runs, in order to reduce external factors in the measurement.

The serial and parallel parts of the code can be time measured (t_{ser} and t_{par}) in all runs. With the percentage of serial time, an estimate of the maximum theoretical speedup can be obtained using the following formula:

$$\begin{aligned} speedup(P) &= \frac{t_{total}}{t_{ser} + \frac{t_{par}}{P}} \\ &= \frac{1}{f + \frac{1-f}{P}} \text{ , with } f = \frac{t_{serial}}{t_{total}} \text{ (serial fraction)} \\ max_{speedup} &= \lim_{P \to \infty} \frac{1}{f + \frac{1-f}{P}} \\ &= \frac{1}{f} \end{aligned}$$

where P is the number of running processes or threads. The efficiency of a parallelization using P running processes or threads is defined as

$$efficiency(P) = \frac{speedup(P)}{P} \quad (1)$$

If the efficiency is small, the obtained speedup is not optimal, since the usage of the P processes or threads is not achieving the peak performance ($efficiency(P) \approx 1$).

4.1. SGSIM

The case study for the parallel SGSIM code uses a real mining 3D dataset of 2376 diamond drill-hole samples with information of copper grade composites. In Figure 2 a realization sample is depicted, with standardized values simulating the copper grades. Table 1 contains all relevant parameters, such as grid sizes, search lookup windows and variographic parameters. The local interpolation is ordinary kriging.

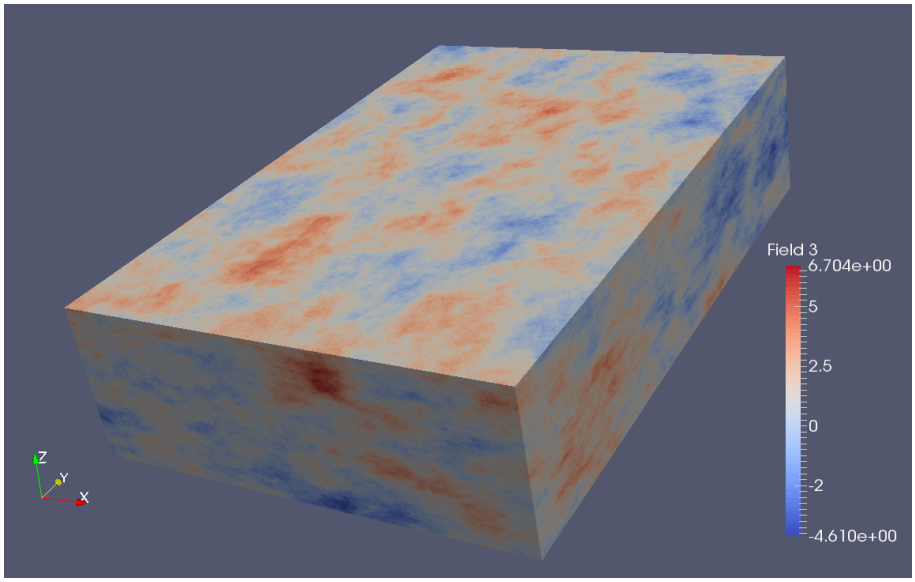

Figure 2: Realization sample of the SGSIM case study.

Parameter	Values
$nx \times\ ny \times nz$	$\{800 \times 800 \times 160, 400 \times 800 \times 160\}$
$xsiz, ysiz, zsiz$	$0.5, 0.75, 0.9$
max data for kriging	$\{16, 32, 64, 128\}$
max search radii	$300, 300, 300$
size of covariance lookup table	$201 \times 201 \times 201$
number of structures and type	3,{spherical,exponential,gaussian}

Table 1: Parameters for SGSIM case study: grid sizes, search lookup window and variography for all categories. For a description of each parameter, see Deutsch & Journel (1998) section V.7.2.

The results are depicted in Tables 3, 4, 5 and 6 for the larger grid size with $800 \times 800 \times 160$ nodes ($102,400,000$ nodes), and 7, 8, 9 and 10 for the smaller grid size with $400 \times 800 \times 160$ nodes ($51,200,000$ nodes). Figure 3 depicts the tendency of the achieved speedup according to the number of maximum number of neighbours for kriging and grid size. We can observe that as the number of maximum kriging neighbours increases, the achieved speedup using 16 threads also increases since the percentage of serial time (f) decreases proportionally. Using both grid sizes, the percentage of serial time in each case remains similar, with approximate values of 39%, 19%, 7% and 2% respectively. These percentages imply that the approximate theoretical maximum speedup that can be reached with 16 threads is 2.1×, 3.6×, 6.4× and 11.5× for each maximum number of neighbours for kriging, which are values far from being optimal in terms of parallel efficiency (Equation 1).

4.2. SISIM

A minor modification must be done in the base code SISIM, in order to run simulations on large domains ($> 2^{24} = 16,777,216$ nodes). The array that stores the random path visiting order, denoted **order**, is defined originally as a `real` structure. Since `real` is a single-precision floating point representation, the maximum integer value that can be represented with this data type is 2^{24}, since the size of the significant precision bits is 24 (IEEE, 2008). By changing

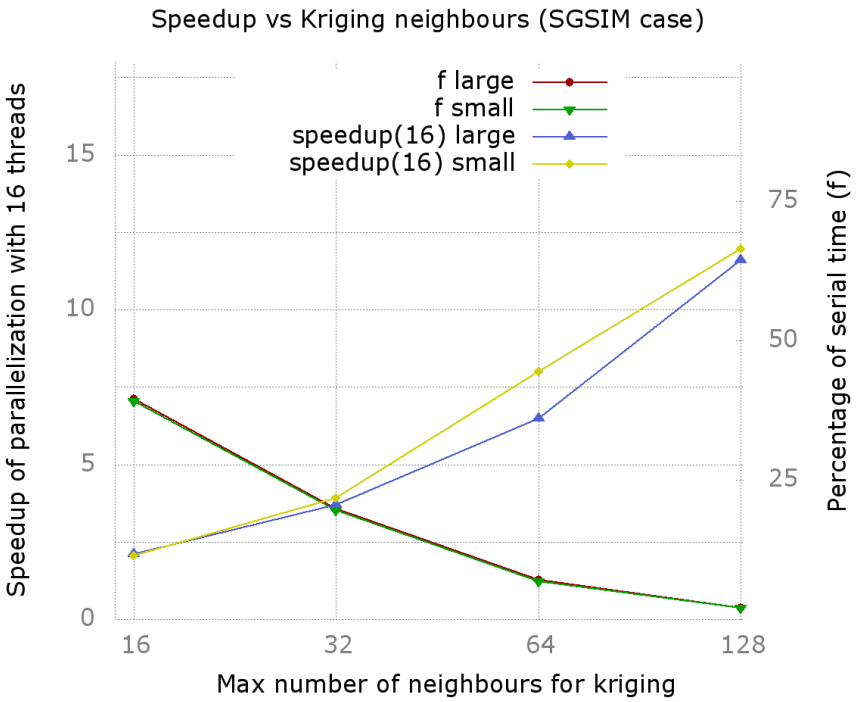

Figure 3: Speedup and percentage of serial time of SGSIM case using 16 threads and different maximum number of neighbours for kriging with a large (102,400,000 nodes) and small grids (51,200,000 nodes).

the data type of **order** to `integer`, a maximum of $2^{32} - 1 = 2,147,483,647$ nodes can be achieved.

The case study for the parallel SISIM code uses a synthetic 3D dataset of 3000 random samples with 10 categories generated by truncation of a convoluted Gaussian kernel with a white noise random field according to the procedure described by Peredo et al. (2016) (Figure 4 shows a realization using three categories). Table 2 contains all relevant parameters, such as grid sizes, search lookup windows and variographic parameters. In all cases the method of local interpolation was simple kriging, with the option *full indicator kriging* active.

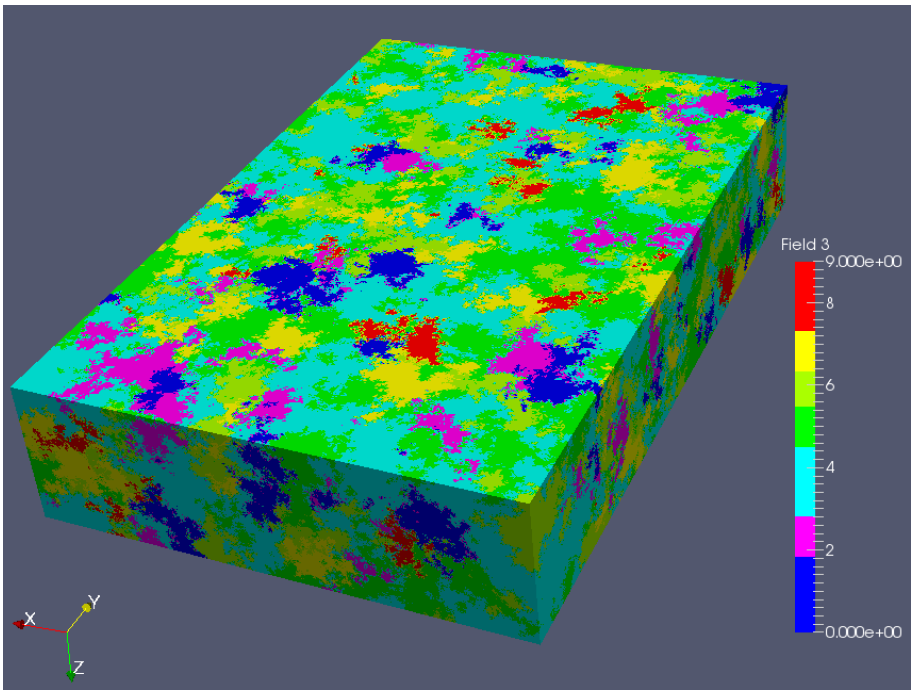

Figure 4: Realization sample of the SISIM case study.

Parameter	Values
$nx \times \ ny \times nz$	$\{420 \times 600 \times 400, 210 \times 600 \times 400\}$
$xsiz, ysiz, zsiz$	$9.5, 10.0, 3.0$
max data for kriging	$\{16, 32, 64, 128\}$
max search radii	∞, ∞, ∞
size of covariance lookup table	$51 \times 51 \times 166$
number of categories	10
number of structures and type	10,{spherical}

Table 2: Parameters for SISIM case study: grid sizes, search lookup window and variography for all categories. For a description of each parameter, see Deutsch & Journel (1998) section V.8.1.

The results are depicted in Tables 11, 12, 13 and 14 for the larger grid size with $420 \times 600 \times 400$ nodes ($100,800,000$ nodes), and 15, 16, 17 and 18 for the

smaller grid size with $210 \times 600 \times 400$ nodes ($50,400,000$ nodes). Figure 5 depicts the tendency of the maximum achieved speedup according to the maximum number of neighbours for kriging and grid size. Similarly to the SGSIM case, as the maximum number of neighbours for kriging increases, the achieved speedup using 16 threads also increases since the percentage of serial time decreases proportionally. The percentages of serial time using a large and small grid are similar, with approximate values of 8%, 3%, 1% and 0.2% respectively. Since these values are considerably lower that the percentages of the SGSIM case, the maximum theoretical speedup that can be achieved with 16 threads is higher, with approximate values of 7.0×, 10.7×, 13.8× and 15.5× respectively.

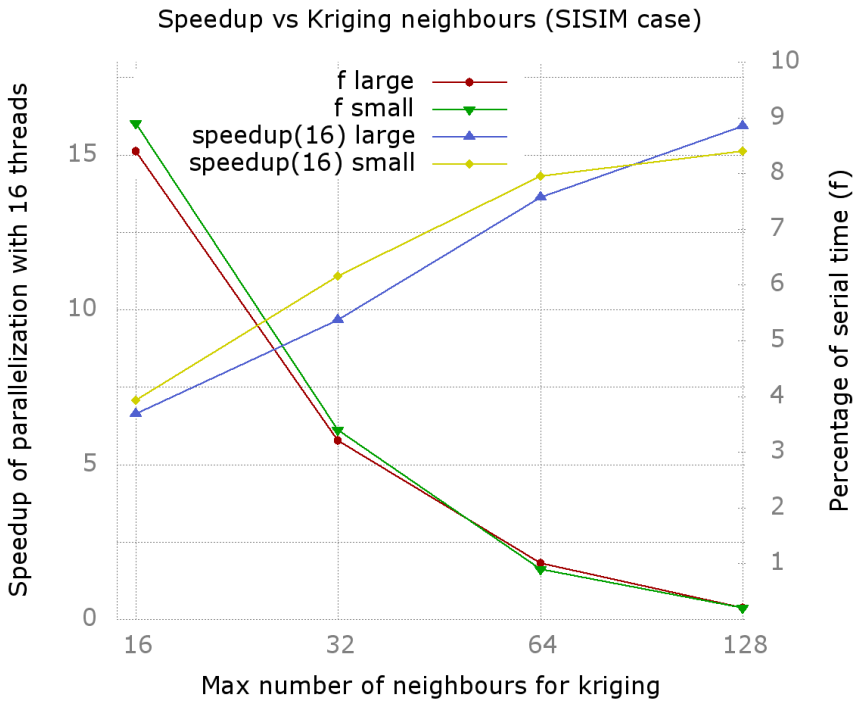

Figure 5: Speedup and percentage of serial time of SISIM case using 16 threads and different maximum number of neighbours for kriging with a large (100,800,000 nodes) and small grids (50,400,000 nodes).

4.3. Analysis

Figure 6 shows the relationship between efficiency of the parallelization and the maximum number of neighbours for kriging, according to the previous results for SGSIM and SISIM using 16 threads from Tables 3-10 and 11-18. As mentioned before, as the number of maximum kriging neighbours increases, the efficiency increases as well. The lower efficiency obtained in the overall SGSIM results can be explained in part by the relative small amount of computation involved in the execution of these cases, compared against the SISIM case. The number of kriging computations per node is exactly one, in contrast to SISIM where ten interpolations must be solved (ten categories to simulate). As shown in Figure 7, a small number of grid nodes are simulated in parallel in the first levels, which adds a large amount of overhead to thread initialization, such as shared/private variables setup. The best result in terms of efficiency for SGSIM is obtained using the larger maximum number of neighbours, 128, which is directly related with higher number of computations in the local interpolations. The efficiency obtained in all SISIM cases is higher than the SGSIM cases and can be explained by the higher amount of computation involved in the parallel step while the serial part is kept identical. As mentioned before, by using ten categories for simulation, ten local interpolation systems must be solved for each grid node. Regarding the number of grid nodes per level, since a larger number of levels contain sufficiently large number of grid nodes (Figure 8), high parallel efficiency values are obtained with more than 90% in almost all cases. The best result for SISIM is obtained using the larger maximum number of neighbours, 128, for the same reasons as the best SGSIM case.

Regarding numerical precision of the results, in SGSIM only small errors with absolute value less than 1.0^{-6} are present, as a result of non-commuting floating-point operations using the different order of simulation. As a reference, the results returned by SGSIM are single-precision floats with 6 to 9 significant decimal digits (IEEE, 2008). To obtain the error values a simple node by node substraction is calculated between the simulated values using the original SGSIM non-parallel code and the values obtained using the parallel version,

and then the histogram of errors is calculated. In the case of SISIM, the results are exactly the same since integer values are rounded for all categories, without small errors in lower decimal digits as SGSIM.

In comparison with other reported parallelization strategies, particularly Rasera et al. (2015), the efficiency obtained is comparable only in the larger cases of SISIM with 64 or 128 maximum neighbours. However, the results reported must not be compared directly, since different base codes and parameters were used. Since the proposed method of this work aims to generate the exact same results as the non-parallel versions of the simulation algorithms, the serial part of node reordering adds a major bottleneck if small domains or small maximum neighbour number are used in the configuration parameters. However, in some applications the exactness property can be particularly useful, like audited practices in mineral and ore reserves estimation (JORC, 2012).

In terms of computational resources, the parallelization strategy uses a large amount of memory to perform the level and neighbourhood storage in the current implementation version. The reason of this requirement is that many additional shared arrays with the same dimension of the simulation array must be allocated, and also additional space is needed by the neighbour information array **Neighbours**, extracted in the push stage of the spiral search (Algorithm 3).

In the largest cases, with approximately 100 million nodes and 128 maximum kriging neighbours, around 96GB of memory where needed. This size comes largely from the array **Neighbours** which stores approximate $100,000,000 \times 2 \times 128$ 4-byte integers. With 16, 32 and 64 maximum kriging neighbours, the memory usage is around 12GB, 24GB and 48GB respectively. Since several cloud computing providers offer computational services at affordable prices, these memory usage values are not prohibitive given the current technological trends. For instance, a Linux virtual machine with 16 CPU-cores, 112GB RAM and 800GB of disk can be rented by 1 dollar per hour (Microsoft Azure, 2017).

To sum up, SGSIM shows a significant efficiency in the largest scenarios and under performs in the smaller scenarios, decreasing approximately $12\times$ the

GSLIB baseline execution time in the best case with efficiency of 74% using 16 threads. SISIM shows higher speedup and efficiency, decreasing approximately 15× the baseline time in the best case with efficiency of 97% using 16 threads, thanks to the lower serial fraction which results from an increase of work in the simulation loop (more kriging system solving in each node). Considering that no additional libraries or external tools were used in the parallelization (with exception of OpenMP), further gains can be achieved by reducing the serial time.

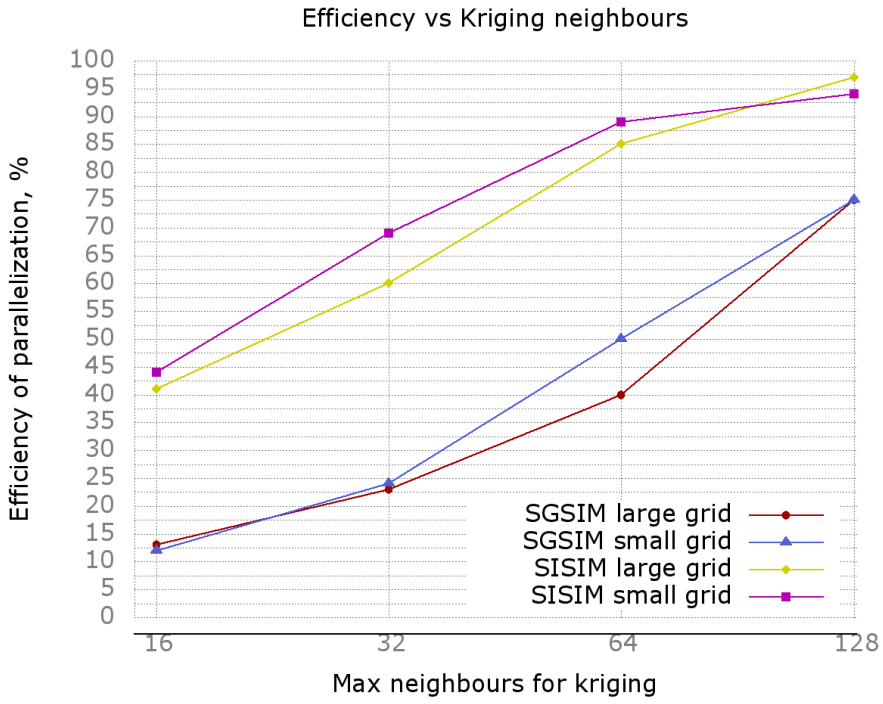

Figure 6: Relationship between efficiency of the parallelization and kriging neighbours using 16 threads in all cases.

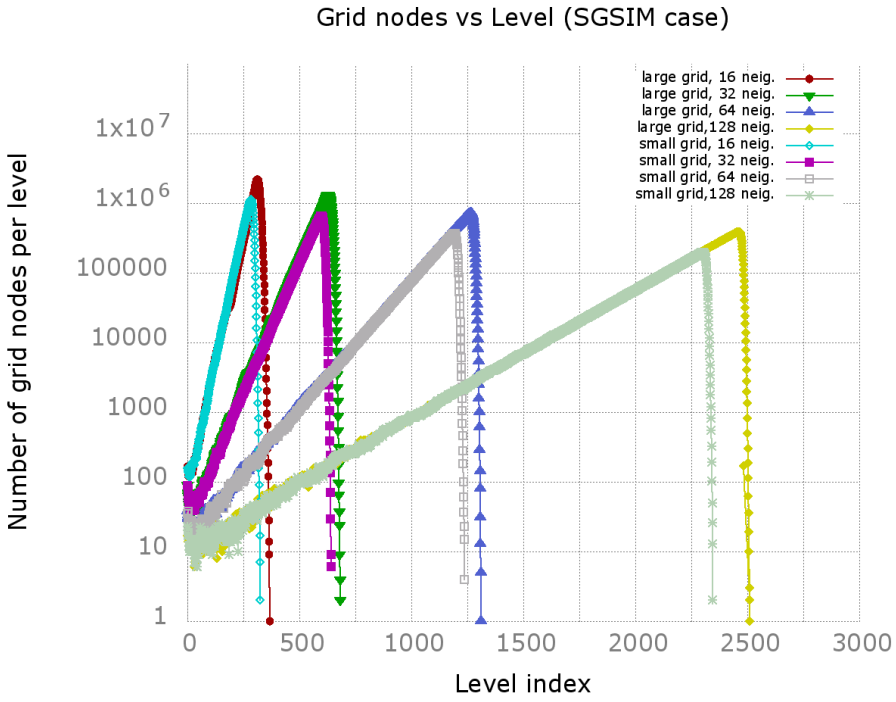

Figure 7: Number of grid nodes per level in SGSIM case.

5. Conclusions and future work

We have shown a path-level parallelization of the sequential simulation, using as base code SISIM from GSLIB. Our parallelization delivers the exact same results as the original routine. The proposed parallelization strategy groups the original unmodified simulation path, by assigning a level to each grid node and, subsequently, performing parallel processing in all nodes of each level, one level at a time in ascending order. The strategy is straightforward to implement using OpenMP directives in a well proven Fortran base code, without using external libraries or packages, and keeping the same user interface from the classic GSLIB.

The achieved speedup in the first case study, using the SGSIM code, is reasonable using large number of maximum neighbours, but not optimal in

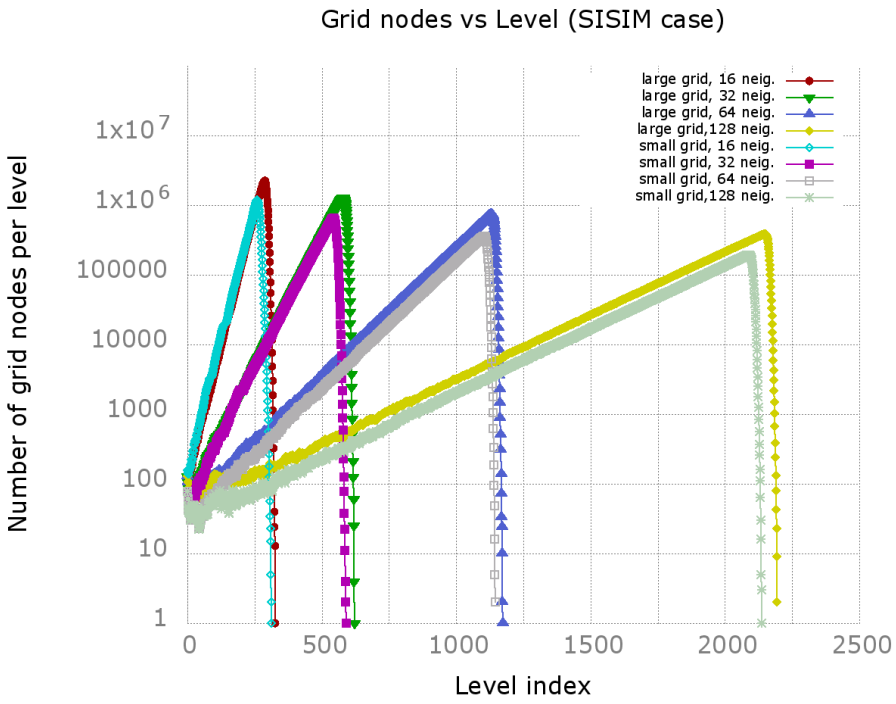

Figure 8: Number of grid nodes per level in SISIM case.

smaller tests, reaching efficiencies below 50% using 16 threads for 16 and 32 neighbours, and 50% and 74% using 16 threads for 64 and 128 neighbours. On the other hand, in the second case study using the SISIM code with ten categories to simulate, the speedup obtained was considerably better than the previous case, closer to the optimal and reaching an efficiency larger than 90% using 16 threads in almost all tests. Applying a modification in the data types of the base code (particularly the array that stores the random path), larger domains can be simulated without further efforts (increased from 2^{24} to $2^{32} - 1$ maximum grid nodes).

In both cases, the serial part of the execution is the main bottleneck of performance and efficiency of the parallelization. A possible strategy to decrease the serial part is to aggressively optimize the internal routine that searches for

neighbours using the covariance lookup table dimensions (Algorithm 3).

Regarding multiple realizations of the simulation, distributed executions can be performed in different compute-nodes, using all threads in each node to run one parallel execution. Future integration with the code-optimized versions of SGSIM and SISIM from Peredo et al. (2015) is being planned.

6. Source code

The current version of the code can be downloaded from

`https://github.com/operedo/sisim-levels`

`https://github.com/operedo/sgsim-levels`

7. Acknowledgements

The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the Barcelona Supercomputing Center - Centro Nacional de Supercomputación (Spain) which supports the Marenostrum supercomputer, the National Laboratory for High Performance Computing (Chile), which supports the Leftraru supercomputer, and the IT services from Department of Computer Architecture at UPC-BarcelonaTech which supports the Arvei computer cluster. Additionally, the authors acknowledge the donated resources from Telefónica Chile, public Chilean government initiative CORFO 13CEE2-21592 (2013-21592-1-INNOVA_PRODUCCION2013-21592-1), project TIN2015-65316-P of the Spanish *Ministerio de Economía y Competitividad*, and project 2014-SGR-1051 from the Generalitat de Catalunya.

Special thanks to Pablo Garcia Briosso and the CID Telefónica Staff for the invested resources and constant encouragement, and the two anonymous reviewers which contribute with insightful comments and suggestions.

References

Alabert, F. (1987). *Stochastic Imaging of Spatial Distributions using Hard and Soft Information*. Master's thesis Stanford University.

de Almeida, J. A. (2010). Stochastic simulation methods for characterization of lithoclasses in carbonate reservoirs. *Earth-Science Reviews*, *101*, 250 – 270.

Bastante, F., Ordonez, C., Taboada, J., & Matías, J. (2008). Comparison of indicator kriging, conditional indicator simulation and multiple-point statistics used to model slate deposits. *Engineering Geology*, *98*, 50–59.

Bierkens, M. F. P., & Burrough, P. A. (1993). The indicator approach to categorical soil data. *Journal of Soil Science*, *44*, 361–368.

Caers, J. (2005). *Petroleum Geostatistics*. An interdisciplinary approach to topics in petroleum engineering and geosciences. Society of Petroleum Engineers.

Chilès, J.-P., & Delfiner, P. (1999). *Geostatistics : modeling spatial uncertainty*. Wiley series in probability and statistics. New York: Wiley. URL: `http://opac.inria.fr/record=b1098313` a Wiley-Interscience publication.

Daly, C., & Caers, J. (2010). Multi-point geostatistics - an introductory overview. *First Break*, *28*, 39–47.

dell'Arciprete, D., Bersezio, R., Felletti, F., Giudici, M., Comunian, A., & Renard, P. (2012). Comparison of three geostatistical methods for hydrofacies simulation: a test on alluvial sediments. *Hydrogeology Journal*, *20*, 299–311.

Deutsch, C. V. (2006). A sequential indicator simulation program for categorical variables with point and block data: Blocksis. *Computers & Geosciences*, *32*, 1669 – 1681.

Deutsch, C. V., & Journel, A. G. (1998). *GSLIB: Geostatistical Software Library and user's guide*. Applied geostatistics series (2nd ed.). New York, NY: Oxford Univ. Press.

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. (1st ed.). Springer.

Dimitrakopoulos, R. (1998). Conditional simulation algorithms for modelling orebody uncertainty in open pit optimization. *International Journal of Surface Mining, Reclamation and Environment*, *12*, 173–179.

Dimitrakopoulos, R., & Dagbert, M. (1993). Sequential modelling of relative indicator variables: Dealing with multiple lithology types. In A. Soares (Ed.), *Geostatistics Tróia '92: Volume 1* (pp. 413–424). Dordrecht: Springer Netherlands.

Dowd, P. A., Xu, C., Mardia, K. V., & Fowell, R. J. (2007). A comparison of methods for the stochastic simulation of rock fractures. *Mathematical Geology*, *39*, 697–714.

Dubrule, O., & Damsleth, E. (2001). Achievements and challenges in petroleum geostatistics. *Petroleum Geoscience*, *7*, S1–S7.

Goovaerts, P. (2001). Geostatistical modelling of uncertainty in soil science. *Geoderma*, *103*, 3 – 26. Estimating uncertainty in soil models.

IEEE (2008). IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, (pp. 1–70).

Isaaks, E. (1990). *The Application of Monte Carlo Methods to the Analysis of Spatially Correlated Data*. Master's thesis.

Johnson, M. E. (1987). *Multivariate Statistical Simulation*. New York, NY: John Wiley.

JORC (2012). Australasian code for reporting of exploration results, mineral resources and ore reserves (the JORC Code, 2012 Edition): Report prepared by the Joint Ore Reserve Committee of the Australasian Institute of Mining and Metallurgy, Australian Institute of Geoscientists and Minerals Council of Australia. URL: `http://www.jorc.org/`.

Journel, A., & Alabert, F. (1989). Non-gaussian data expansion in the earth sciences. *Terra Nova*, *1*, 123 134.

Journel, A. G., & Isaaks, E. H. (1984). Conditional Indicator Simulation: Application to a Saskatchewan uranium deposit. *Journal of the International Association for Mathematical Geology*, *16*, 685–718.

Machuca-Mory, D. F., Ortiz, J. M., & Deutsch, C. V. (2008). On the challenge of using sequential indicator simulation for the estimation of recoverable reserves. *International Journal of Mining, Reclamation and Environment*, *22*, 285–299.

Mariethoz, G. (2010). A general parallelization strategy for random path based geostatistical simulation methods. *Computers & Geosciences*, *36*, 953 – 958. URL: http://dx.doi.org/10.1016/j.cageo.2009.11.001. doi:10.1016/j.cageo.2009.11.001.

Matheron, G., Beucher, H., de Fouquet, C., Galli, A., Guerillot, D., & Ravenne, C. (1987). Conditional simulation of the geometry of fluvio-deltaic reservoirs. In *62nd Annual Technical Conference and Exhibition of the Society of Petroleum Engineers, Dallas, Texas* (pp. 123–131). Society of Petroleum Engineers.

van der Meer, F. (1994). Sequential indicator conditional simulation and indicator kriging applied to discrimination of dolomitization in ger 63-channel imaging spectrometer data. *Nonrenewable Resources*, *3*, 146–164.

Microsoft Azure (2017). Linux Virtual Machines Pricing, D14 v2 Promo specifications. URL: https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/.

Navarro, F., González, C., Peredo, Ó., Morales, G., Egaña, Á., & Ortiz, J. M. (2014). A flexible strategy for distributed and parallel execution of a monolithic large-scale sequential application. In G. Hernández, C. J. Barrios Hernández, G. Díaz, C. García Garino, S. Nesmachnow, T. Pérez-Acle, M. Storti, & M. Vázquez (Eds.), *High Performance Computing: First HPCLATAM - CLCAR Latin American Joint Conference, CARLA 2014, Val-*

paraiso, Chile, October 20-22, 2014. Proceedings (pp. 54–67). Berlin, Heidelberg: Springer Berlin Heidelberg.

Nunes, R., & Almeida, J. A. (2010). Parallelization of sequential gaussian, indicator and direct simulation algorithms. *Computers & Geosciences*, *36*, 1042 – 1052.

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

Ortiz, J. M., & Deutsch, C. V. (2004). Indicator simulation accounting for multiple-point statistics. *Mathematical Geology*, *36*, 545–565.

Pan, G. (1997). Conditional simulation as a tool for measuring uncertainties in petroleum exploration. *Nonrenewable Resources*, *6*, 285–293.

Peredo, O., & Ortiz, J. M. (2011). Parallel implementation of simulated annealing to reproduce multiple-point statistics. *Computers & geosciences*, *37*, 1110–1121.

Peredo, O., & Ortiz, J. M. (2012). Multiple-point geostatistical simulation based on genetic algorithms implemented in a shared-memory supercomputer. In *Geostatistics Oslo 2012* (pp. 103–114). Springer Netherlands.

Peredo, O., Ortiz, J. M., & Herrero, J. R. (2015). Acceleration of the Geostatistical Software Library (GSLIB) by code optimization and hybrid parallel programming. *Computers & Geosciences*, *85, Part A*, 210 – 233.

Peredo, O., Ortiz, J. M., Herrero, J. R., & Samaniego, C. (2014). Tuning and hybrid parallelization of a genetic-based multi-point statistics simulation code. *Parallel Computing*, *40*, 144–158.

Peredo, O., Ortiz, J. M., & Leuangthong, O. (2016). Inverse modeling of moving average isotropic kernels for non-parametric three-dimensional gaussian simulation. *Mathematical Geosciences*, *48*, 559–579.

Rasera, L. G., Machado, P. L., & Costa, J. F. C. (2015). A conflict-free, path-level parallelization approach for sequential simulation algorithms. *Computers & Geosciences*, *80*, 49 – 61.

de Souza, L. E., & Costa, J. F. C. (2013). Sample weighted variograms on the sequential indicator simulation of coal deposits. *International Journal of Coal Geology*, *112*, 154 – 163. Special issue on geostatistical and spatiotemporal modeling of coal resources.

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.

Tahmasebi, P., Sahimi, M., Mariethoz, G., & Hezarkhani, A. (2012). Accelerating geostatistical simulations using graphics processing units (gpu). *Computers & Geosciences*, *46*, 51 – 59.

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.

Yarus, J. M., Chambers, R. L., Maucec, M., & Shi, G. (2012). Facies simulation in practice: Lithotype proportion mapping and plurigaussian simulation, a powerful combination. In *Geostatistics Oslo 2012*. Springer Netherlands.

Zhu, H., & Journel, A. G. (1993). Formatting and Integrating Soft Data: Stochastic Imaging via the Markov-Bayes Algorithm. In A. Soares (Ed.), *Geostatistics Tróia '92: Volume 1* (pp. 1–12). Dordrecht: Springer Netherlands.

Annex: Tables with numerical results

SGSIM execution time and speedup

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	766.574	1			
1 (omp)	811.090	0.945	320.949	0.395	1
2 (omp)	588.970	1.301	325.524	0.552	1.432
4 (omp)	522.970	1.465	311.224	0.595	1.828
8 (omp)	381.586	2.008	310.503	0.813	2.122
16 (omp)	362.666	2.113	325.929	0.898	2.306

Table 3: Time/Speedup of SGSIM, serial and parallel fractions: 102,400,000 grid nodes and maximum of 16 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	2743.532	1			
1 (omp)	3036.122	0.903	601.266	0.198	1
2 (omp)	1823.014	1.504	597.452	0.327	1.669
4 (omp)	1217.604	2.253	591.161	0.485	2.509
8 (omp)	918.545	2.986	602.056	0.655	3.352
16 (omp)	743.966	3.687	585.737	0.787	4.029

Table 4: Time/Speedup of SGSIM, serial and parallel fractions: 102,400,000 grid nodes and maximum of 32 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	14795.288	1			
1 (omp)	16337.288	0.905	1152.972	0.070	1
2 (omp)	8719.349	1.696	1181.455	0.135	1.868
4 (omp)	5116.514	2.891	1184.814	0.231	3.301
8 (omp)	3235.145	4.573	1189.570	0.367	5.354
16 (omp)	2278.863	6.492	1181.039	0.518	7.772

Table 5: Time/Speedup of SGSIM, serial and parallel fractions: 102,400,000 grid nodes and maximum of 64 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	106393.643	1			
1 (omp)	109554.268	0.971	2420.665	0.022	1
2 (omp)	51280.557	2.074	2413.071	0.047	1.956
4 (omp)	27401.376	3.882	2408.623	0.087	3.752
8 (omp)	15561.031	6.837	2446.570	0.157	6.932
16 (omp)	9175.207	11.595	2423.585	0.264	12.030

Table 6: Time/Speedup of SGSIM, serial and parallel fractions: 102,400,000 grid nodes and maximum of 128 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	393.929	1			
1 (omp)	413.479	0.952	161.854	0.391	1
2 (omp)	299.263	1.316	165.900	0.554	1.437
4 (omp)	235.154	1.675	165.991	0.705	1.839
8 (omp)	204.316	1.928	167.452	0.819	2.138
16 (omp)	190.589	2.066	166.153	0.871	2.328

Table 7: Time/Speedup of SGSIM, serial and parallel fractions: 51,200,000 grid nodes and maximum of 16 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	1453.179	1			
1 (omp)	1497.249	0.970	293.959	0.196	1
2 (omp)	888.965	1.634	288.895	0.324	1.671
4 (omp)	600.941	2.418	288.974	0.480	2.517
8 (omp)	448.859	3.237	291.008	0.648	3.369
16 (omp)	371.572	3.910	290.456	0.781	4.055

Table 8: Time/Speedup of SGSIM, serial and parallel fractions: 51,200,000 grid nodes and maximum of 32 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	7921.930	1			
1 (omp)	7908.930	1.001	541.059	0.068	1
2 (omp)	4148.421	1.909	532.623	0.128	1.871
4 (omp)	2382.522	3.325	532.860	0.223	3.318
8 (omp)	1474.447	5.372	531.122	0.360	5.409
16 (omp)	989.462	8.006	515.919	0.521	7.896

Table 9: Time/Speedup of SGSIM, serial and parallel fractions: 51,200,000 grid nodes and maximum of 64 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	51726.258	1			
1 (omp)	48939.258	1.056	1070.864	0.021	1
2 (omp)	26576.200	1.946	1088.212	0.040	1.957
4 (omp)	14121.606	3.662	1105.343	0.078	3.753
8 (omp)	7574.541	6.828	1098.133	0.144	6.937
16 (omp)	4328.232	11.950	1096.891	0.253	12.046

Table 10: Time/Speedup of SGSIM, serial and parallel fractions: 51,200,000 grid nodes and maximum of 128 neighbours to infer conditional probability.

SISIM execution time and speedup

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	4755.103	1			
1 (omp)	4522.103	1.051	380.150	0.084	1
2 (omp)	2533.220	1.877	380.019	0.150	1.844
4 (omp)	1501.155	3.167	379.732	0.252	3.194
8 (omp)	931.031	5.107	378.360	0.406	5.036
16 (omp)	718.128	6.621	379.683	0.528	7.076

Table 11: Time/Speedup of SISIM, serial and parallel fractions: 100,800,000 grid nodes, 10 categories and maximum of 16 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	22560.543	1			
1 (omp)	21325.543	1.057	696.359	0.032	1
2 (omp)	11251.314	2.005	663.583	0.058	1.936
4 (omp)	6221.798	3.626	662.079	0.106	3.643
8 (omp)	3399.888	6.635	660.492	0.194	6.511
16 (omp)	2332.582	9.671	661.245	0.283	10.739

Table 12: Time/Speedup of SISIM, serial and parallel fractions: 100,800,000 grid nodes, 10 categories and maximum of 32 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	140548	1			
1 (omp)	137425.264	1.022	1446.443	0.010	1
2 (omp)	71605.673	1.962	1386.161	0.019	1.979
4 (omp)	36470.465	3.853	1255.436	0.034	3.877
8 (omp)	18590.755	7.560	1261.528	0.067	7.451
16 (omp)	10310.647	13.63	1385.712	0.134	13.818

Table 13: Time/Speedup of SISIM, serial and parallel fractions: 100,800,000 grid nodes, 10 categories and maximum of 64 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	1017870	1			
1 (omp)	1001197.2	1.016	2211.853	0.002	1
2 (omp)	511434.417	1.990	2309.101	0.004	1.996
4 (omp)	252730.204	4.027	2294.369	0.009	3.976
8 (omp)	125209.547	8.129	2349.094	0.018	7.889
16 (omp)	63913.105	15.925	2257.121	0.035	15.533

Table 14: Time/Speedup of SISIM, serial and parallel fractions: 100,800,000 grid nodes, 10 categories and maximum of 128 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	2566.105	1			
1 (omp)	2456.625	1.044	219.108	0.089	1
2 (omp)	1328.791	1.931	219.435	0.165	1.836
4 (omp)	793.696	3.233	217.062	0.273	3.155
8 (omp)	505.963	5.071	215.699	0.426	4.925
16 (omp)	362.476	7.079	217.765	0.600	6.843

Table 15: Time/Speedup of SISIM, serial and parallel fractions: 50,400,000 grid nodes, 10 categories and maximum of 16 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	12093.603	1			
1 (omp)	11347.603	1.065	391.077	0.034	1
2 (omp)	5850.835	2.066	389.079	0.066	1.933
4 (omp)	3191.038	3.789	391.645	0.122	3.625
8 (omp)	1824.460	6.628	404.207	0.221	6.445
16 (omp)	1091.702	11.077	383.317	0.351	10.547

Table 16: Time/Speedup of SISIM, serial and parallel fractions: 50,400,000 grid nodes, 10 categories and maximum of 32 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	73424.527	1			
1 (omp)	70250.527	1.045	671.820	0.009	1
2 (omp)	34813.426	2.109	668.964	0.019	1.981
4 (omp)	18327.576	4.006	669.623	0.036	3.888
8 (omp)	9629.237	7.625	673.144	0.069	7.498
16 (omp)	5132.127	14.306	672.849	0.131	13.992

Table 17: Time/Speedup of SISIM, serial and parallel fractions: 50,400,000 grid nodes, 10 categories and maximum of 64 neighbours to infer conditional probability.

# Threads	$t_{total} = t_{ser} + t_{par}$ [s]	Speedup	t_{ser} [s]	$f = t_{ser}/t_{total}$	Max Speedup
1 (gslib)	490844	1			
1 (omp)	487311.691	1.007	1293.773	0.002	1
2 (omp)	255689.720	1.919	1284.359	0.005	1.994
4 (omp)	128435.340	3.821	1201.493	0.009	3.968
8 (omp)	64432.234	7.617	1301.860	0.020	7.854
16 (omp)	32454.387	15.124	1311.743	0.040	15.387

Table 18: Time/Speedup of SISIM, serial and parallel fractions: 50,400,000 grid nodes, 10 categories and maximum of 128 neighbours to infer conditional probability.