

GPGPU-based parallel variogram calculation and automatic model fitting

David Valenzuela U. ^{1*}, Daniel Pola C.¹ and Alejandro Cáceres S.²

1. *GeoInnova Consultores, Chile*
2. *Department of Mining Engineering / Advanced Mining Technology Center, University of Chile, Chile*

ABSTRACT

Variogram analysis is a necessary step in geostatistics and mineral resources evaluation to describe the spatial dependence of continuous and categorical regionalized variables. It is an exploratory data tool, as well as a key input for all the prediction and simulation algorithms based on kriging.

Variogram analysis has two major steps. The first one is the calculation of experimental variograms by using possibly hundreds of thousands of sampling data. The high computational cost of this step is primarily given by the calculations of the distances between all the combinations of pairs of data locations, resulting in repetitive and independent operations. In this research, we propose a General-Purpose Graphic Processing Unit (GPGPU) algorithm for the multi-directional direct and cross variogram calculation, to parallelize the computation of the distances between pairs of data locations given the independence of the operations.

The second step is the fitting of a variogram model using a set of nested structures. Traditionally, this fitting is performed manually using three orthogonal directions. We propose a GPGPU Particle Swarm Optimization (PSO) fitting scheme that allows adjusting several directions and variables simultaneously under the linear model of correlogram. The fitting aims to reduce the mean squared error between the experimental variogram and variogram model, while ensuring the consistency of the model.

The results show a significant reduction in experimental variogram computation time, compared to the traditional sequential CPU algorithm. Both the GPGPU experimental calculation and GPGPU PSO fitting provide a suitable automatic method to analyze large datasets such as blasthole data or Landsat images. The approach is illustrated using a dense blasthole dataset to fit local variograms to be further used as an input in short-term resources models.

***Corresponding author:** Área de Innovación y Desarrollo I+D, GeoInnova Consultores, Ingeniero de Proyectos I+D, Antonio Bellet 444 Of. 1301. Phone: +56 2 22049082. Email: dvalenzuela@geoinnova.cl

INTRODUCTION

The experimental variogram calculation and the variogram model curve adjustment are highly required in the mining industry, being both computed many times for each considered data set. Improvements on the computation time of these algorithms result in a reduction of the total time required to complete the variogram calculation process, impacting directly on the mining operation practices.

The GPU parallel programming platforms and APIs development, and the widespread growth of GPU-based algorithms used by developers and researchers to solve diverse technological and scientific problems, offer a feasible alternative to implement highly parallelized algorithms. A GPU has hundreds more cores than a CPU, and consequently much more threads to parallelize, becoming an ideal and affordable hardware for repetitive independent computations over extensive data sets.

This paper shows the advantages of a dedicated hardware for repetitive computations in the variogram calculation and automatic model fitting, comparing the computation time of experimental variogram from drillholes and blastholes samples, using a CPU-based and a GPU-based algorithm.

Furthermore, we propose a parallel optimization algorithm that searches for the best fitting between the experimental variogram and the variogram model, which is a linear combination of spherical, exponential, cubic and gaussian models, minimizing the Mean Squared Error between both variograms. The proposed method considers a reduced number of parameters that need to be introduced by the users.

We propose a study case for experimental validation where several local experimental variograms are obtained around each centroid of the block model to have a more realistic texture and enhance the prediction model. The approach is applied in a real data set of a Chilean copper porphyry deposit in an open pit mine operation using blastholes and drillholes databases.

NOMENCLATURE

BH	Blastholes
CPU	Central Processing Unit.
CUDA	Compute Unified Device Architecture.
DH	Drillholes
gamv	Variogram calculation of scattered data (GSLIB program)
GPGPU	General-Purpose Computing on Graphics Processing Units.
GPU	Graphics Processing Units.
GSLIB	Geostatistical Software Library
LU	Lower Upper (decomposition)
MSE	Mean Squared Error
PSO	Particle Swarm Optimization.

METHODOLOGY

This section shows the implementations of the experimental variogram and its model fitting and the comparison between the CPU-based and GPU-based algorithms. The applications where GPU-based

algorithms show advantages are fundamentals to get better local information about the distribution of continuous variables in the space.

CPU Traditional Experimental Variogram Algorithm

The experimental semivariogram algorithm used as comparative base is a GSLIB (Geostatistical Software Library) CPU-based implementation. The traditional semivariogram is defined by:

$$\gamma(h) = \frac{1}{2 |N(h)|} \sum_{i=1}^{N(h)} (Z(x_i) - Z(x_i + h))^2 \quad (1),$$

where $Z(x_i)$ and $Z(x_i + h)$ are a pair of values of the variable used to compute the semivariogram, $N(h)$ is the number of data pairs that are separated at distance h , and x_i is the location associated to the pair i .

For the experimental variogram calculation several parameters are needed such as the lag distance, that represent the size of the bin in which we must divide the space to accumulate the distance values, and the *nlag*, that represents the number of total bins and determines the maximum possible range. *All_lag* is the array constructed with the *lag* and *nlag* parameters and determine the center of each bin. The azimuth, dip, azimuth tolerance and dip tolerance, determine the angles that define the data subset.

Figure 1 shows the pseudo-code of the CPU-based multivariable semivariogram, where i and j are the index of the samples selected from Data set, lag_{ij} is the euclidean distance between the samples pair (i, j) , *num_var* is the number of variables that participate in the co-variogram, ΔZ is the product between the differences in Z of a pair of samples of continuous variables, *lagAcum* and $\Delta ZAcum$, are accumulators of lag_{ij} and ΔZ respectively, $\Delta ZConta$ is the counter of each samples pair from a pair variables computed. Finally, the *distExp* is the experimental distance (x axis) computed by the division of *lagAcum* over $\Delta ZConta$, and *varExp* is the experimental semi-variogram values (y axis) obtained from the Equation 1.:

```

for i ∈ DataSet:
    for j ∈ DataSet:
        lagij = √Δx2 + Δy2 + Δz2
        for lagcurrent ∈ all_lag:
            for var ∈ (num_var * num_var) :
                ΔZ = (Zi,var1 - Zj,var1) * (Zi,var2 - Zj,var2)
                lagAcum[lagcurrent, var] += lagij
                ΔZAcum[lagcurrent, var] += ΔZ
                ΔZConta[lagcurrent, var] += 1
    for k ∈ (all_lag * num_var * num_var):
        distExp[k] = lagAcum[k] / ΔZConta[k]
        varExp[k] = ΔZAcum[k] / (2 * ΔZConta[k])

```

Figure 1 Pseudocode of CPU Traditional Experimental Variogram Algorithm

This semivariogram implementation results in high demand of computer resources, mainly given by four nested cycles as can be seen in Figure 1. The computation time increases quadratically ($O(n^2)$) where n is the data set size (usually blastholes (BH) or drillholes (DH) samples), therefore, the first two nested cycles (*for i, for j* in Figure 1) determine the algorithm complexity. The following two nested cycles depend on parameters such *lag*, *nlag*, angles, angular tolerances and how many pairs of continuous variables participate (co-variogram).

GPU Experimental Variogram Algorithm

Using the GPU for tasks traditionally managed by CPU (such as linear algebra computations), instead of computer graphics, is called General-purpose computing on graphics processing units (GPGPU). For GPGPU approach we use CUDA, that is the proprietary language of NVIDIA® that allows to code based in C language and take access to specific functions to use the multiple threads and GPU global memory (Lindholm et al., 2008). The CUDA threads are indexed using 1-Dimensional (x), 2-Dimensional (x, y), or 3-Dimensional (x, y, z) grid. These threads are organized in blocks, that also are indexed 1D, 2D or 3D. The maximum number of threads per block is given by the CUDA capability of the GPU hardware. The maximum number of active blocks is also limited by hardware. The maximum number of blocks that can be processed at the same time is related to the number of GPU multiprocessors (Li, Kecman & Salman, 2010).

The variogram algorithm parallelization is suitable because the computation the distance between each pair of samples of the data set can be done by multiple threads independently (Peredo, Ortiz & Herrero, 2015). The distances are stored in an array, where each register represents an individual lag with the respective tolerance. The GPU-based implementation is not memory intensive but is demanding in CUDA cores resources (number of blocks per grid, and threads per block).

In the Figure 2 we present the GPU experimental variogram pseudocode. The elements shown in the pseudocode GPU Figure 2 are similar to the elements presented in the pseudocode CPU Figure 1, however, the GPU Kernel 1 replaces the 2 main nested cycles that indexes the samples pair implemented in the CPU approach. The Kernel 1 is implemented with 2D blocks grid, to keep one thread for i index and the other thread for j index (Figure 2) independently of the samples pair. The Kernel 2 is implemented with 1D blocks grid, whose threads compute a simple arithmetic operation using the accumulated values from Kernel 1.

<p>CUDA Kernel 1: Accumulator</p> <pre> i = thread.X j = thread.Y lag_{ij} = $\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$ for lag_{current} ∈ all_lag: for var ∈ (num_var * num_var) : ΔZ = (Z_{i,var1} - Z_{j,var1}) * (Z_{i,var2} - Z_{j,var2}) lagAcum[lag_{current}, var] += lag_{ij} ΔZAcum[lag_{current}, var] += ΔZ ΔZConta[lag_{current}, var] += 1 </pre>
<p>CUDA Kernel 2: Variogram</p> <pre> k = thread.X distExp[k] = lagAcum[k] / ΔZConta[k] varExp[k] = ΔZAcum[k] / (2 * ΔZConta[k]) </pre>

Figure 2 Pseudocode GPU Traditional Experimental Variogram Algorithm

The Figure 3 shows the difference between the sequential and parallel algorithm. The first case uses only one thread (CPU) and the second case uses multiple threads (GPU). Additionally, the GPU algorithm takes advantage of the symmetry when computing the distance between the first sample and the second sample, therefore only requires completing the superior triangular matrix.

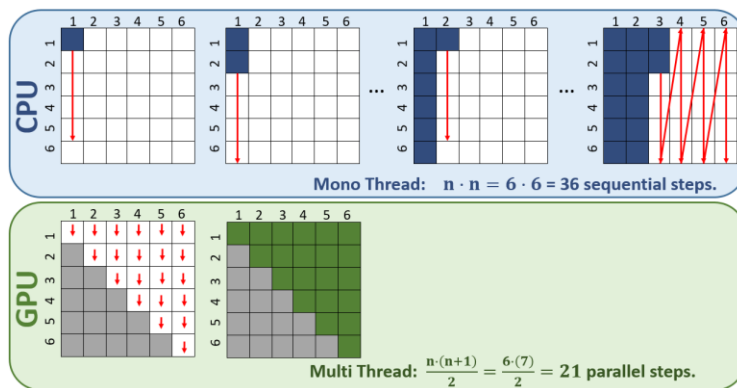


Figure 3 GPU vs CPU gamv algorithm

CPU Traditional Variogram Model Fitting Algorithm

The experimental variogram allows to quantify the spatial changes in a continuous variable based on the real samples data, but the objective is to use this information to create estimation results, i.e. get suitable data in the locations where the continuous variable values are unknown. The most used estimation algorithm is called cokriging, which has a restriction: The covariance matrix that represents the spatial relationship of the continuous variable must be invertible. To handle this restriction, the matrix is required to be positive semi-definite, which implies that the variogram at

least must be obtained from a linear combination of functions that keep this property. The variography model must be created with the linear combination of basic structures such as spherical, exponential, cubic, and gaussian models, keeping the invertibility propriety, and adjusting as much as possible to the experimental variogram.

In the industry is very common to use specialized software (as Isatis®) to get the variogram model fitting, but in general these algorithms are poorly automatized. The user must give several parameters, such as the angles or the number of nested structures in the model, among others. When the software gives a possible solution model, the user can modify parameter values to try to get a better fit manually, without any quantified evaluation. This situation is an opportunity to explore an alternative method to obtain fitted models.

GPU Variogram Model Fitting Algorithm

To reach the best fit between the experimental variogram and variogram model, based on Mean Squared Error (MSE) criterion, we use a method called Particle Swarm Optimization (PSO). As other optimization algorithms, PSO tries to minimize an objective function, in this case, the MSE between the experimental and the proposed variogram model. In the first step, this algorithm creates a random population of candidate solutions (particles), that move inside the search-space during the next steps. The movement of each particle is determined by a velocity vector, which is calculated with the best position known of each particle (individual learning), and the best position known among all particles (social or global learning). This produces the movement of the swarm to reach the best solution, inspired by insects that are looking for food in their habitat.

The proposed variogram models are randomly initialized. Each set of parameters represents the position of a particle in a n-dimensional space. For example, for a two-nested structure, we define a particle of 17 dimensions (see Figure 4), where each dimension corresponds to an individual parameter. The parameters that define the curve are: Model Type (Spherical, Exponential, Cubic, or Gaussian model), Range X, Range Y, Range Z, Angles X, Angles Y, Angles Z, Partial Sill, and finally the Nugget. The combination of partial sills from each nested structure, defines the full sill (see Figure 4).

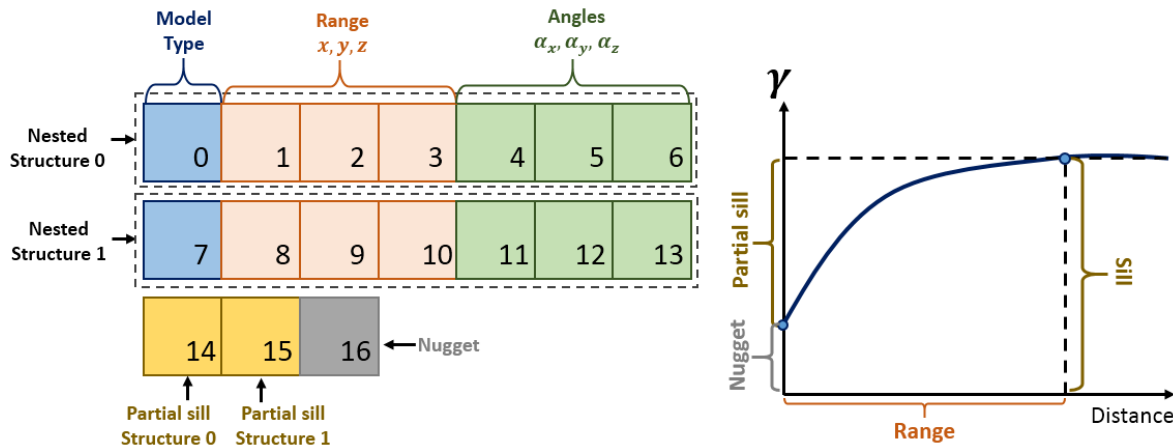


Figure 4 Example of particle used to generate a variogram model with two nested structures, through PSO algorithm

During the first step N random variogram model curves (see Figure 5) are initialized. Then, the objective function is computed by each particle, identifying the best particle in that iteration. Later, the velocity function is computed, and determines how the particles position change in the next iteration. The velocity function is defined by three sections: The global learning section, the individual learning section and a proportional random section. The global learning is a vector between the current particle and the best particle in the current iteration, and has the aim to force the current particle to follow the path of the best sample of the entire population. On the other hand, the individual learning section represents "the individual memory" of each particle, that keeps the best historical performance of each particle without being influenced by the others. This keeps the balance between greedy particles and exploratory particles. Greedy particles look for the best result as soon as possible with the undesirable consequence to only find a local optimum. The proportional random section is the responsible of a random value change, in some random selected group of particles in each iteration, to increase the variability of possible results candidates. The random effect is proportional because its effect increases with the iteration number, to avoid the accelerated convergence of the algorithm.

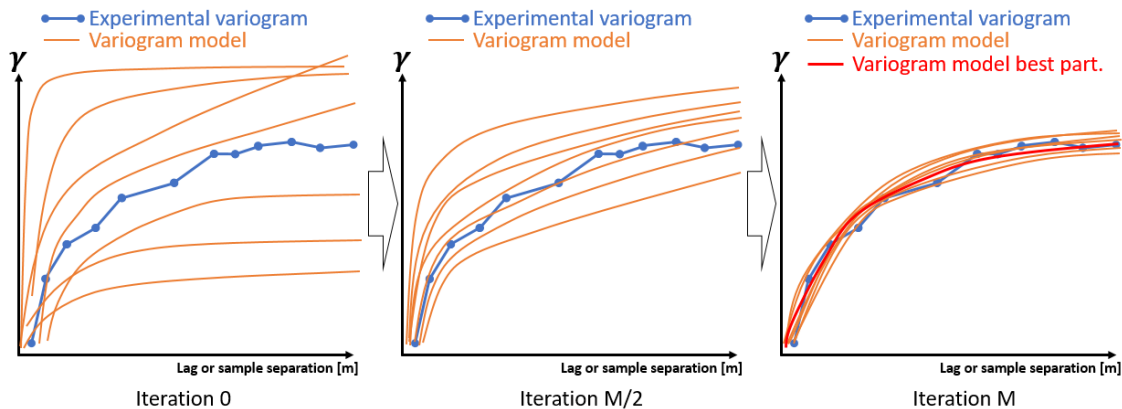


Figure 5 Particle Swarm Optimization for variogram model search

The Figure 5 shows how the particles population evolves in each iteration, changing their position (considering the n-dimensional space), which is traduced in a new candidate of variogram model per particle. As the number of iteration increases, particles tend to move closer to the best variogram model curve.

This variogram fitting algorithm is fully automated and it is not necessary to predetermine what type of models are used. The only two prerequisites are: (i) to determine the minimum and maximum values of each dimension of the particles constraining the feasible space solution, and (ii) define the number of maximum nested structures, that are necessary to calculate the dimension of the particle position vector.

Due to the nature of the PSO algorithm and the multi-dimensional optimization problem, the resulting model is not a unique solution. With K executions, it is possible to get K different solutions. Therefore, it is necessary to define a number of iterations and particles enough to ensure the convergence in a reasonable time.

In the implemented GPU version of the PSO (Papadakis & Bakrtzis, 2011), (Bali et al., 2015), (Hussain, Hattori & Fujimoto, 2016), all the particle computations have been parallelized, i.e., knowing the location of the best particle, all the particles update their position and obtain the objective function result at the same time (each thread computes the operations for one particle). Thus, it is possible to use a considerable number of particles (over 1000) and don't get a significative delay in the computational time, therefore it is possible use a small number of iterations, that implies a better fit in less time. It is worth to mention that the iterations over the population evolution remain sequentially computed.

A set of sintethic data with a known variogram was made through unconditional simulation using LU decomposition algoritmn. The data was used as an input to the PSO algorithm to check the output results against the known model and adjust internal algorithm parameters such as learning rates, number of particles and iterations.

Case of Study

The case of study is divided in global and local calculation of the experimental and fitting model variograms, using 28 999 drillholes samples and 87 305 blastholes samples from an open pit mine database. The total block model grid has 327 829 blocks.

For the global approach, the first test is computing the experimental variogram with all samples from drillhole dataset. The second test is computing the experimental variogram with the samples from blasthole dataset. Both, the first and the second test are computed by CPU and GPU in order to compare results and time performance. The third test is the model fitting using the PSO GPU approach for drillholes and blastholes experimental variogram.

Given the hardware capabilities and the parallelized algorithms (experimental variogram and variogram model fitting) we propose obtaining an experimental variogram and a variogram model in a local approach for several locations using each centroid of a deposit block model as the reference. To achieve this, we have two main steps:

The first step is the search of sample neighbors from each centroid of the deposit block model, obtaining the corresponding distance between each sample and the centroid that will be used in the weighted variogram. In the case of blastholes samples, we use a 40 meters radio neighborhood. In the case of drillholes samples, we use a 500 meters neighborhood. The second step is iterating over each file and get the experimental curve and then the variogram model curve. In the case of drillholes samples we use weighted variogram implementation and in the case of blastholes samples we use the non-weighted variogram. In both cases we use the same GPU implementation to obtain the variogram model curves.

The weighted GPU experimental variogram implementation assigns more relevance to samples that are closer to a fixed reference (Ec. 2). We use this weighted algorithm to obtain local variograms through samples from drillholes data base. The weights are obtained using (2)

$$\omega_{ij}^k = \omega_i^k \cdot \omega_j^k = \frac{1}{\epsilon + d_{k,i}^2} \cdot \frac{1}{\epsilon + d_{k,j}^2} \quad (2),$$

where ω_{ij}^k is the weight associated to centroid k for the samples pair ij , $\epsilon \approx 0.00001$, $d_{k,i}$ is the distance between centroid k and sample i , and $d_{k,j}$ is the distance between centroid k and sample j .

RESULTS AND DISCUSSION

All the following results are obtained from a computer with an Intel Core i7-5930K 3.5GHz processor, 32 Gb RAM, GPU NVIDIA Quadro M6000 24 GB, and a PCI-e SSD 480 GB. The data set used in the experimental validation is consists of 4 files: The drillholes samples, the grid of block model that supports the drillhole samples, the blastholes samples, and the grid of block model that supports the blastholes. Following we show the details from each file:

Table 1 Data bases used in experimental validation

	DH samples	DH grid	BH samples	BH grid
Number of registers	28 999	327 829	87 305	30 735

Benchmark Traditional Variogram CPU vs GPU

Figure 6 shows the comparison in performance time between CPU and GPU gamv algorithm (experimental variogram). The GPU shows an advantage of around 20 times less computational time than the same algorithm running in CPU hardware. Furthermore, the bigger the data set size the

better the computational time advantages of using a GPU-based version with respect to a CPU-based implementation.

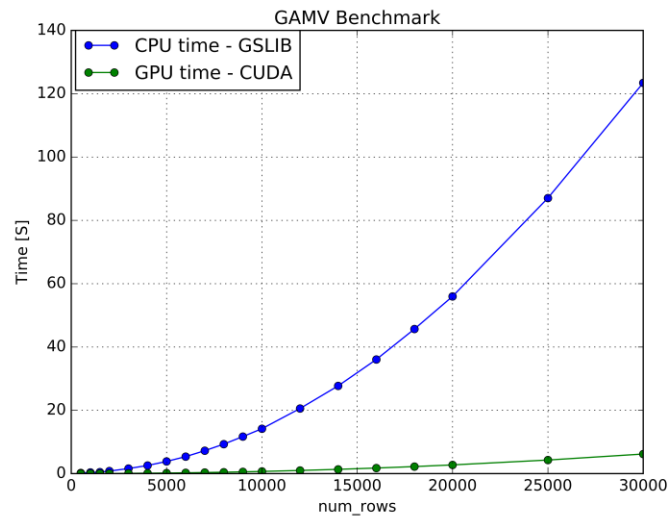


Figure 6 Experimental Variogram (gamv) Benchmark (CPU vs GPU)

Variogram Model curve from ideal experimental variogram

To test the gamv fitting algorithm (variogram model curve fitting) an "ideal" experimental curve was created, based on fixed parameters. The idea is to obtain the same parameters used to create the experimental curve at the end of the PSO algorithm. The result converges to the original values with a MSE of 0.000 002 (See Figure 7).

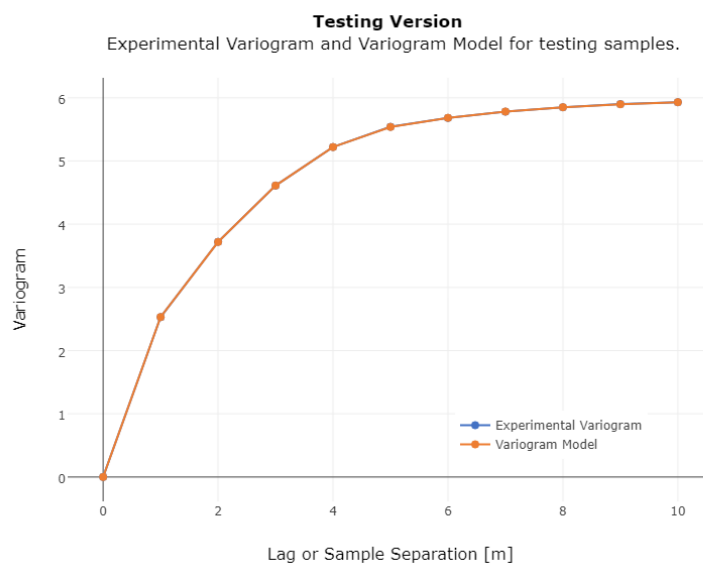


Figure 7 Experimental variogram and variogram model for testing samples.

Experimental variogram and variogram model from Blastholes samples

Figure 8 and Figure 9 present an irregular experimental variogram, due to lack of Blastholes samples neighbors (40 meters) around each centroid of the grid. The variogram model has a huge MSE due the irregularities in the experimental variogram. In this case, the experimental variogram do not consider weights.

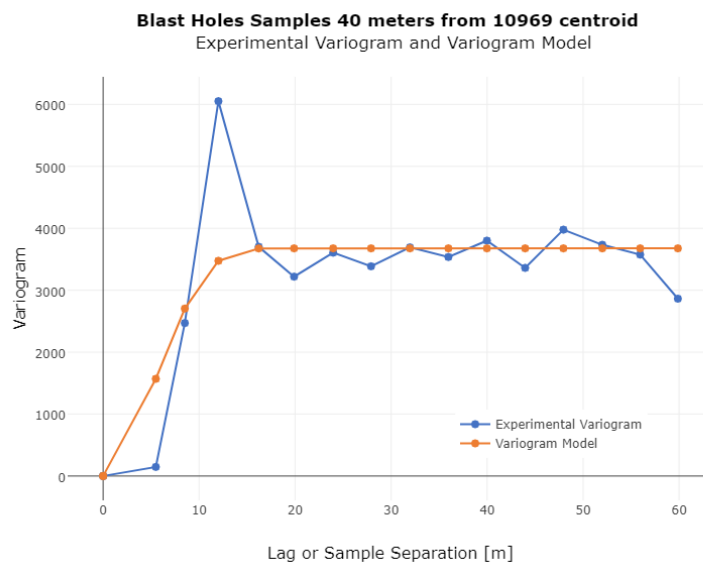


Figure 8 Experimental variogram and variogram model for Blastholes samples

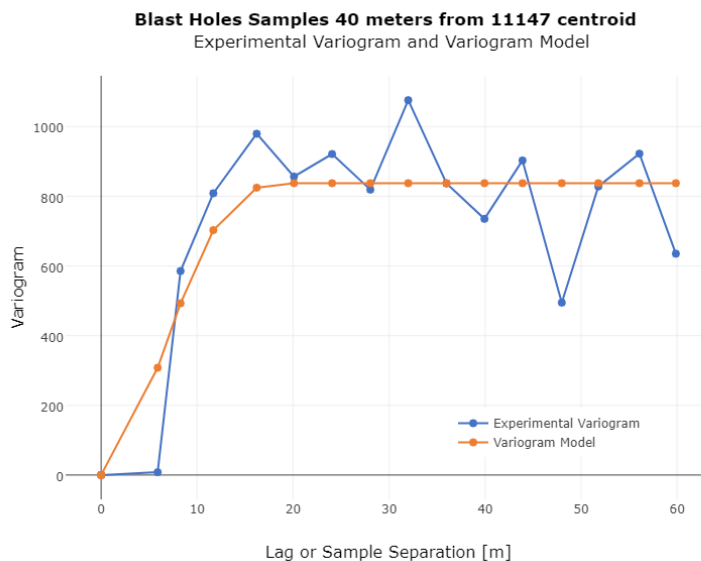


Figure 9 Experimental variogram and variogram model for Blastholes samples

Experimental variogram and variogram model from Drillholes samples

Figure 10 and Figure 11 present a better result compared with the blastholes testing, because the drillholes samples are more separated than the blastholes samples. In this case, the variogram was executed with the weighted version.

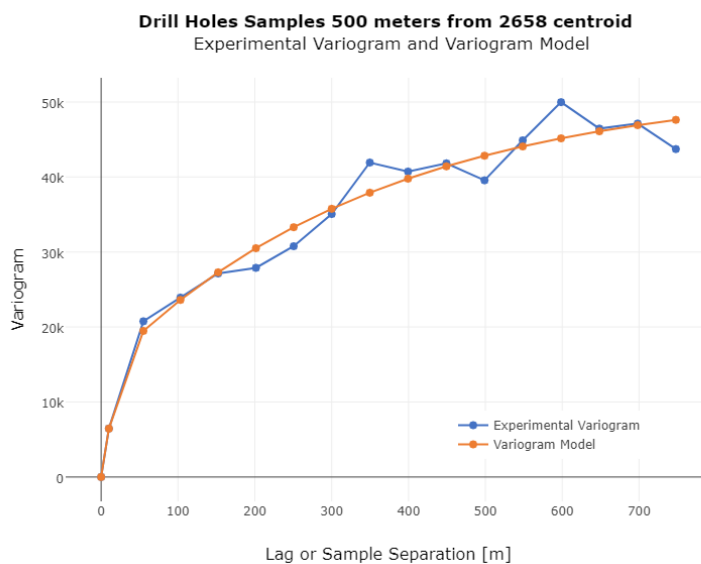


Figure 10 Experimental variogram and variogram model for Drillholes samples

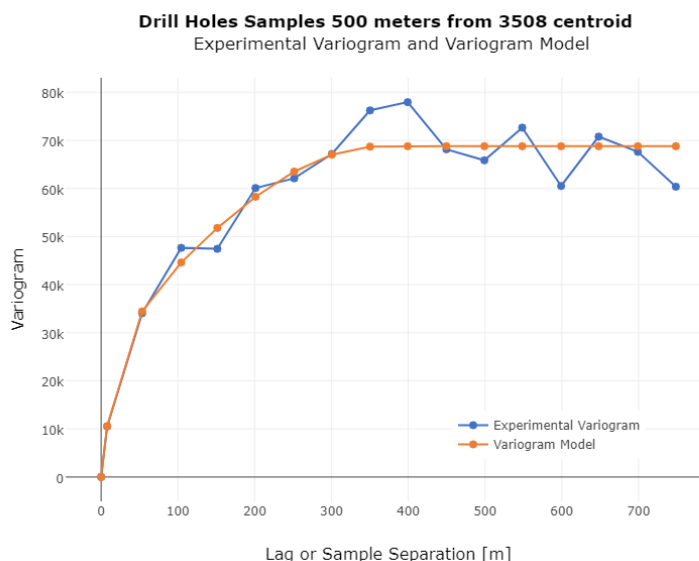


Figure 11 Experimental variogram and variogram model for Drillholes samples

Performance local experimental variograms GPU vs CPU

Table 2 shows the time performance of the experimental variogram obtained from the neighborhood of each centroid of block model, using the drillhole samples. The main time reduction is obtained when the total effective computation time is compared, which only includes the operations to get the results in memory, i.e. does not includes the time to read the data from file, transfer the data to memory (GPU or RAM respectively) neither the time necessary to write the results to a file. The GPU approach reduces the time in almost 98.55% only considering the computation of the variogram, while considering the read-write time, the GPU approach reduce in 42.49% the time of CPU approach. This analysis proves that it is very important to avoid read-write operations that can slow down considerably the total time. In Table 2 we show the Percentage Difference, that is the difference between gamv GPU time and gamv CPU time, divided by the average of the two values.

Table 2 Performance in Local variogram using Drillholes samples, using GPU and CPU approaches

	gamv GPU	gamv CPU	Percentage Difference
Local variograms	327 829	327 829	-
Mean samples per neighbor	1 164.55	1 164.55	-
Variables	1	1	-
Mean Effective computation time	0.004 467 92 sec.	0.307 539 sec.	194.27%

Mean Total Time (Includes read and write to file)	0.410 39 sec.	0.713 462 sec.	53.93%
TOTAL Effective computation time	24 min 25 sec.	1 day 4 H 0 min 21 sec.	194.27%

Figures 12 and 13 show the block model (transparent) around 4 202.5 m. elevation with the values obtained from PSO GPU based variogram fitting parameters. *AmpX0* color bar shows the first nested structure *xRange*, obtained with de Arsenic drillhole samples data set (*AS99* color bar). We can see that in the zones with less data set density, the range (*AmpX0*) is less than in the zones with more density, because with more data is possible to infer more continuity (Figure 13). Furthermore, in the zones with less data density, the nugget effect is higher than in the denser zones, therefore the range values obtained are lower.

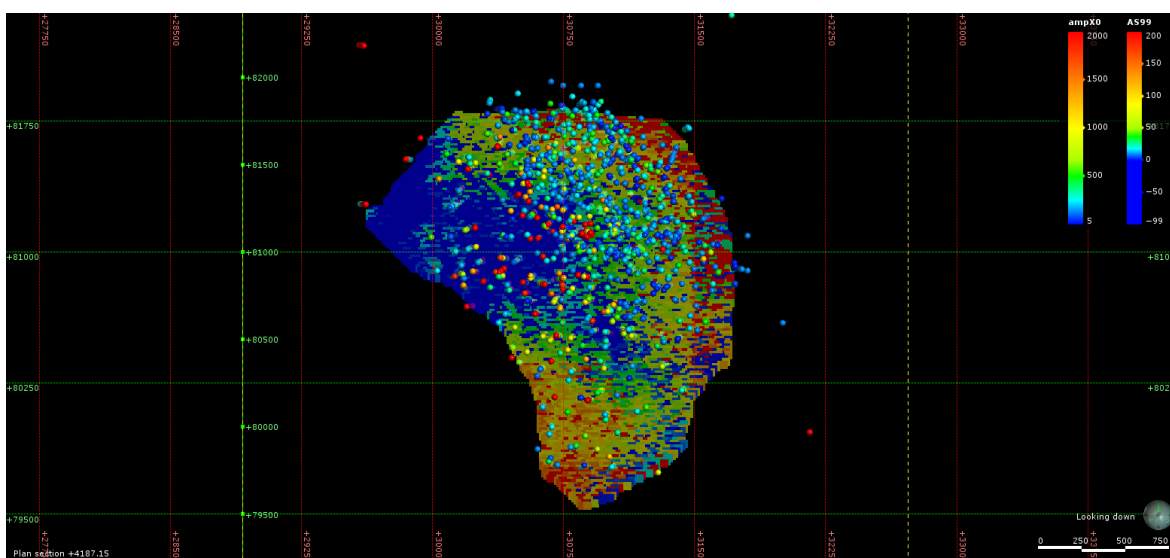


Figure 12 Parameter *xRange* and drillholes (*As*) samples from the first nested structure – variogram model output from PSO GPU based algorithm

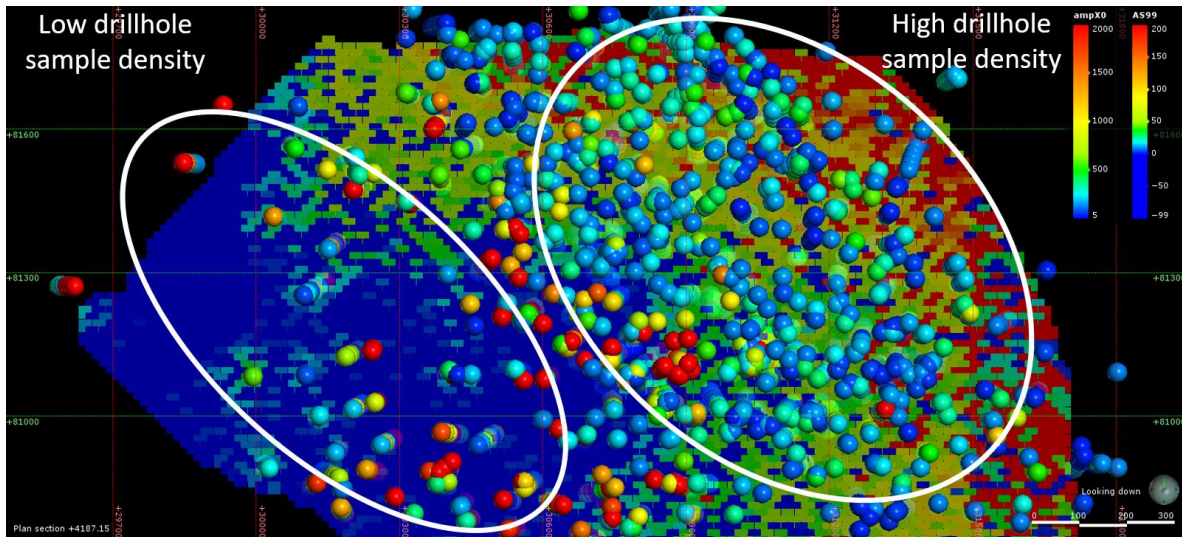


Figure 13 Parameter *xRange* and drillholes (*As*) samples from the first nested structure. Drillhole sample density

Performance local variogram models GPU vs CPU

It is not possible to compare exactly the performance between the GPU and CPU in the variogram adjust, because the algorithms are very different. In this case, not only the reduction of the computation time is important, but also the human decision parameters based in the experience are not more crucial for get satisfactory results.

CONCLUSION

The present paper achieves the goal of showing the advantages of a parallelized algorithm compared to the traditional sequential version. The GPU implementation not only increases the time performance, but also makes possible the local variogram analysis, enhancing the simulation details in a deposit.

The GPU takes considerable advantages over a CPU, when handles a large data set or a huge number of iterations, reducing around 20 times the computation time. By contrast, the development time increases considerably, given the physical design constraints in the GPU (memory, blocks per grid, thread per block, etc.).

The proposed future work includes implementing the search of nearest neighbors in the GPU, for each centroid of block model, then compute the experimental variogram, and finally obtaining the variogram model curve in the same execution without passing through files between each step. Observing the results, the bottleneck for GPU implementations is reading and writing times to Hard Disk Drives. These times delay increase the computational GPU time at least two orders of magnitude. Additionally, we propose making the local computations including all samples from Drillholes with the weighted version, instead of only considering the nearest neighbors at a certain distance.

ACKNOWLEDGEMENTS

This research was supported by “Corporación de Fomento de la Producción - CORFO” through the “PROGRAMA GESTIÓN DE LA INNOVACIÓN – PORTAFOLIO, código 16GPI-60687 Gestión de Innovación en Geoinnova”. We thank our colleagues from GeoInnova Consultores Ltda. whom provided all their expertise in geostatistics.

REFERENCES

Qi Li, Vojislav Kecman, Raied Salman, (2010), “A Chunking Method for Euclidean Distance Matrix Calculation on Large Dataset Using Multi-GPU”, 2010 Ninth International Conference on Machine Learning and Applications, (Conference Paper).

Oscar Peredo, Julián M. Ortiz, José R. Herrero, (2015), “Acceleration of the Geostatistical Software Library (GSLIB) by code optimization and hybrid parallel programming”, Computers & Geosciences, pp. 210–233, (Journal Paper).

Erik Lindholm, John Nickolls, Stuart Oberman, John Montrym, (2008), “NVIDIA Tesla: A Unified Graphics and Computing Architecture”, IEEE Computer Society, pp. 39–55, (Journal Paper).

S.E. Papadakis, A.G. Bakrtzis, (2011), “A GPU accelerated PSO with application to Economic Dispatch problem”, 2011 16th International Conference on Intelligent System Application to Power Systems (ISAP), (Conference Paper).

Olfa Bali, Walid Elloumi, Pavel Krömer, Adel M. Alimi, (2015) “GPU Particle Swarm Optimization Applied to Travelling Salesman Problem”, 2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoc), (Conference Paper).

Md. Maruf Hussain, Hiroshi Hattori, Noriyuki Fujimoto, (2016), “A CUDA Implementation of the Standard Particle Swarm Optimization”, 2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) , (Conference Paper).