Computers & Geosciences 64 (2014) 1–6

Contents lists available at ScienceDirect

Computers & Geosciences

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

Using general-purpose computing on graphics processing units (GPGPU) to accelerate the ordinary kriging algorithm

E. Gutiérrez de Ravé *, F.J. Jiménez-Hornero, A.B. Ariza-Villaverde, J.M. Gómez-López

Department of Graphic Engineering, University of Córdoba, Gregor Mendel Building, Campus Rabanales, 14071 Córdoba, Spain

ARTICLE INFO

Article history:
Received 15 May 2013
Received in revised form
25 October 2013
Accepted 30 November 2013
Available online 6 December 2013

Keywords:
Graphic processing units (GPU)
General-purpose computation with GPUs (GPGPU)
Compute unified device architecture (CUDA)
Parallelization
Kriging

ABSTRACT

Spatial interpolation methods have been applied to many disciplines, the ordinary kriging interpolation being one of the methods most frequently used. However, kriging comprises a computational cost that scales as the cube of the number of data points. Therefore, one most pressing problems in geostatistical simulations is that of developing methods that can reduce the computational time. Weights calculation and then the estimate for each unknown point is the most time-consuming step in ordinary kriging. This work investigates the potential reduction in execution time by selecting the suitable operations involved in this step to be parallelized by using general-purpose computing on graphics processing units (GPGPU) and Compute Unified Device Architecture (CUDA). This study has been performed by taking into account comparative studies between graphic and central processing units on two different machines, a personal computer (GPU, GeForce 9500, and CPU, AMD Athlon X2 4600) and a server (GPU, Tesla C1060, and CPU, Xeon 5600). In addition, two data types (float and double) have been considered in the executions. The experimental results indicate that parallel implementation of matrix inverse by using GPGPU and CUDA will be enough to reduce the execution time of weights calculation and estimation for each unknown point and, as a result, the global performance time of ordinary kriging. In addition, suitable array dimensions for using the available parallelized code have been determined for each case. Thus, it is possible to obtain relevant saved times compared to those resulting from considering wider parallelized extension. This fact demonstrates the convenience of carrying out this kind of study in other interpolation calculation methodologies using matrices.

1. Introduction

In most cases, the data of one part of the region are unavailable, and only sparsely and unevenly scattered point samples have been collected. Therefore, spatial interpolation techniques are essential for estimating environmental variables for the un-sampled locations. Spatial interpolation methods have been applied to many disciplines (Zhou et al., 2007), and play a significant role in environmental science. Environmental managers often require spatial continuous data over a region of interest to make effective and informed decisions. In addition, scientists need accurate data which are well-distributed across a region to make justified interpretations (Li and Heap, 2008, 2011). In this context, saving the time required for decision making is important (i.e. predictions in risk situations, short-term meteorological forecast). It is needed to generate results as fast as possible in order to be useful for prediction purposes (Pesquer et al., 2011).

Spatial analysis methods handle a large amount of information, and CPU-based hardware development is not progressing as quickly as would be required by the models. One alternative to overcome this drawback is graphics processing units (GPUs). They are ubiquitous (i.e. present in personal computers) and can be seen as very cheap platforms for carrying out parallel computing (Fatone et al., 2012).

Li and Heap (2011) assessed 53 comparative studies and analyzed the performance of 72 methods/sub-methods. According to these authors, inverse distance weighting, ordinary kriging (OK) and ordinary co-kriging are those most frequently used. In general, kriging methods perform better than non-geostatistical ones (Li and Heap, 2011). Kriging (Isaaks and Srivastava, 1989; Kitanidis, 1997; Goovaerts, 1997a, 1997b) is an example of a method whose computational cost greatly increases when it is applied to large amounts of data. Kriging, a discipline originated by Matheron (1963), was originally developed in geostatistics by a South African mining engineer called Krige. Although this method is frequently used as a spatial interpolation option (Kleijnen, 2009) it is included in many Geographic Information Systems (GIS) (Burrough and McDonnell, 1998), statistical software and surface analysis and representation. Several authors have confirmed its high computational cost (e.g. Kerry and Hawick, 1998; Lloyd, 2006; O'Sullivan and Unwin, 2002).

* Corresponding author. Tel.: +34 957 218 371; fax: +34 957 218 455.
E-mail address: eduardo@uco.es (E. Gutiérrez de Ravé).

0098-3004/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cageo.2013.11.004

Several strategies have already been proposed for reducing the required heavy computations associated with geostatistical simulations. Thus, Tahmasebi et al. (2012) proposed to address the problem of the computational cost of geostatistical simulations by using a parallel computational strategy. This strategy has been applied with success to universal kriging by using OpenMP (Cheng et al., 2010) and graphics processing units, GPUs, (Cheng, 2013). In this paper, the research has been focused on the most time-consuming step in kriging that corresponds to calculating the weights and estimating each unknown point (Cheng, 2013). The use of graphical processors for nongraphical purposes, known as general-purpose computing on graphics processing units (GPGPU), has been explored here in order to reduce the execution time of the operations involved in this stage of the ordinary kriging algorithm. This research includes several comparative studies of Compute Unified Device Architecture (CUDA), developed by Nvidia (2008), GPGPU codes vs. conventional CPU processing considering different types of data (float and double). The suitable array dimensions to use GPGPU in the simulations can be determined with the help of the codes to parallelizing only the time-expensive operations in the algorithm step mentioned above (matrix sum and multiplication and matrix inverse). This circumstance constitutes the main novelty of this work because GPGPU computing is promising in terms of cost-efficiency and peak performance, although the performance is not always better than that of CPU computing depending on the size of the simulation (Li et al., 2013). This aspect is very relevant when dealing with geostatiscal simulations. Tahmasebi et al. (2012) stated that if the simulation grid is too small, it would be more efficient not to use the GPGPU instead of CPU codes based on the high computational cost of conflict management. Regarding the universal kriging algorithm, the research performed by Cheng (2013) is the first attempt to determine the time saved when using GPGPU and CPU codes in two situations: (i) different input data set sizes while the estimated data size is fixed and (ii) different estimated data size while the input data set size is fixed. The obtained results are relevant although the tested input and estimated data set sizes were limited. In addition, the simulations were referred to the major steps in the universal Kriging algorithm. The research described in this work is focused on the matrix operations related with the weights calculation and then the estimate for each unknown point, the most time-consuming step in ordinary kriging. Thus, the level of detail of this study is increased compared to previous works as well as the amount of data set sizes tested.

2. Methodology

2.1. Kriging interpolation

When a variable is distributed in space, it is said to be regionalized, and the geostatistical theory is based on the observation that the variability in all regionalized variables has a particular structure: (i) it is continuous but not adjustable mathematically, (ii) it has random local variation and (iii) it has regional nonrandom variation.

Kriging, a generic name for a family of generalized least-squares regression algorithms, is a popular interpolation method used in Geostatistics (Kleijnen, 2009; Journel and Huijbregts, 1978; Goovaerts, 1997a, 1997b). Geostatistics includes several methods that use kriging algorithms for estimating continuous attributes. In this work, we studied the OK, meaning that it was assumed that the random variable was stationary; the mean constant, but unknown.

Kriging estimates the value of the variable in a location from a weighted average of known values (Oliver and Webster, 1990), a procedure similar to that used in weighted moving average interpolation. In this case, the weights are derived from a geostatistical analysis (Burrough and McDonnell, 1998). The estimate is based on knowledge of the covariance (variogram) of the values taken at the observation points. Eq. (1) shows the predicted value as a weight average of the n values

$$Z(\vec{x}_0) = \sum_{i=1}^{n} \lambda_i Z(\vec{x}_i) \tag{1}$$

with $\sum_{i=1}^{n} \lambda_i = 1$. The weights λ_i are chosen so that the estimate $Z(\vec{x}_0)$ is unbiased, thus minimizing the variance of the variable by modeling its variogram. The minimum variance $Z(\vec{x}_0)$ is given by

$$\sigma_e^2 = \sum_{i=1}^{n} \lambda_i \gamma(x_i, x_0) + \phi \tag{2}$$

and is obtained when

$$\sum_{i=1}^{n} \lambda_i \gamma(\vec{x}_i, \vec{x}_j) + \phi = \gamma(\vec{x}_j, \vec{x}_0) \quad \forall j \tag{3}$$

being $\gamma(\vec{x}_i, \vec{x}_j)$ and $\gamma(\vec{x}_j, \vec{x}_0)$ the semivariances of z between the sampling points x_i, x_j and x_j, x_0. Both these quantities are obtained from the fitted variogram Eq. (5). The quantity ϕ is a Lagrange multiplier required for the minimization.

$$\gamma(h) = \frac{1}{2} E[\{Z(x+h) - Z(x)\}^2] \quad \forall h \tag{4}$$

This method is known as OK. Depending on the semivariance, the OK system is expressed in the matrix $AV = B$, where A is the matrix of the semivariance between points, V is the matrix of weights and B stands for the variogram (Eq. 4):

$$\begin{bmatrix} \gamma_{11} & \gamma_{12} & \cdots & \gamma_{1n} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma_{n1} & \gamma_{n2} & \cdots & \gamma_{nn} & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \\ \phi \end{bmatrix} = \begin{bmatrix} \gamma_{10} \\ \vdots \\ \gamma_{n0} \\ 1 \end{bmatrix} \tag{5}$$

The solution vector is: $V = A^{-1}B$

Several studies have shown that geostatistical methods are generally superior when there are sufficient data to estimate a variogram (Li and Heap, 2008; Burrough and McDonnell, 1998). The computational analysis of the kriging interpolation algorithm shows that calculating the solution of Eq. (5) consumes a large part of the execution time, from approximately 96.0% to 99.8% (Pesquer et al., 2011). This kriging interpolation step involves several matrix operations (matrix sum and multiplication and matrix inverse). Parallel computing reduces the algorithmic complexity and improves the performance of these operations. Therefore, the use of the GPUs benefits is proposed as an alternative to reduce the execution time spent in the step represented by Eq. (5).

2.2. Graphics processing units

GPUs are highly parallel, multi-threaded, multi-core processors with a tremendous computational power and high memory bandwidth. GPUs have hardware components specifically designed for being applied to parallel computation techniques, with the initial purpose of accelerating video data processing. The recent advances in the GPU design is a consequence of the pressure of the games industry, because many floating point calculations per frame are needed. This is achieved thanks to the architecture of GPUs that incorporates many processing units, a large amount of memory (VRAM) and a large data transfer, due a high bandwidth, (more than 10 CPUs) to move data between devices.

Simulations can benefit from using CUDA with GPU in parallel computing. When performing a simulation, both the CPU, for the execution of the sequential code, and the GPU, used to run data-intensive computations in parallel, must be involved. For this

reason, CUDA can be regarded as an alternative to support the joint implementation of the two types of codes within the same application. CUDA is available for a variety of programming languages. To implement an efficient GPU-based Kriging interpolation it would be necessary to redesign the basic algorithms. All the methods involved in the OK algorithm were implemented with the two programming paradigms in order to compare the potential improvement achieved by the use of GPUs compared to the method of architecture-based programming CPU execution. The OK algorithm is mainly divided into matrix, geometric, statistical and data processing operations. Matrix calculations include inversion, multiplication and sum. As geometric operations, the distance and the differences in attributes between each pair of points on a 2D map have to be calculated. Finally, the data set statistics calculations have to be performed.

C/C++ programming language has been selected to perform the simulation in combination with NVIDIA-CUDA technology through its associated SDK libraries. With the aim of adapting the code to the CPU and GPU executions, a conditional compilation is provided with the preprocessor directives to be able to commute between parallel and serial simulations. The execution times for both kinds of computing, serial and parallel, were determined. For this purpose, algorithms and mechanisms to measure computation time, to ensure that the execution is completely equivalent in function calls, allocating and freeing memory (Harris, 2005) have been developed. The execution times for those functions related to matrix calculations (matrix sum and multiplication and matrix inverse) involved in Eq. (5) were studied. All the tests were carried out for different array dimensions.

2.3. Parallel algorithm

The algorithms needed to obtain the result from Eq. (5) were implemented for single CPU thread to multi-threaded algorithms GPU, the latter being included in the CUBLAS library. According to the CUDA Toolkit documentation (http://docs.nvidia.com/cuda/cublas/), the CUBLAS library is an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA®CUDA™ runtime. When using this library, the application must allocate the required matrices in the GPU memory space, fill them with data, call the sequence of desired CUBLAS functions, and then upload the results from the GPU memory space back to the host. The key points of parallelized algorithms are set out below.

2.3.1. Matrix sum

The single thread algorithm executed by a CPU for summing and subtracting matrices has an $O(n^2)$ complexity. Nevertheless, the parallelized algorithm exhibits a complexity of $O(1)$ that can be reached only in the theory because when the matrices exceed a certain size, it will be necessary to execute a series procedure in the blocks on the GPU, increasing the complexity to $O(n)$.

2.3.2. Matrix multiplication

The classical matrix multiplication algorithm is computationally expensive in CPU as a consequence of having an $O(n^3)$ complexity. However, the complexity of the matrix multiplication algorithm in GPU is $O(n^2)$.

2.3.3. Matrix inverse algorithm

The matrix inverse algorithm implemented to exploit the parallel computing architecture provided by CUDA technology uses the Gauss–Jordan reduction method for the resolution of the inverse of a given array. The resulting complexity of the algorithm is $O(n)$, compared to $O(n^2\log(n))$ obtained in the single thread CPU version (Srinivasan et al., 2010).

3. Results

Fig. 1 shows the initial dataset used to perform the test throughout this research. The size of the input dataset was 191. These data correspond to accumulated reference evapotranspiration (RET or ET_0) values, expressed in mm, recorded in Andalusia (Southern Spain). The interpolated output data, consisting of 9604 values can be visualized in the same figure. The mean error was lower than 0.012 mm. This investigation has been done on two different computers, one PC and a server specifically designed for a calculation using CPU and GPU. The characteristics of the PC were AMD Athlon X2 4600+, 512 kB L2, 2 GB DDR, GPU: GeForce 9500 GT, 1 GB DDR2, and PCI−e 2.0. The server features were Processor 2× Xeon 5600 (8 cores per processor), 4 MB L2, 16 GB DDR2, GPU: 4× Tesla C1060, and PCI−e 2.0. Both GPU designed by Nvidia.

Tests on PC including array calculations were carried out with the aim of comparing a single GPU with a single processor core performance. These tests on the PC have been performed with floating point numbers of a single precision (float in C++). This situation is a result of the limitation shown by GeForce 9500 GT card that cannot process double precision numbers (double in C++) because of CUDA computing capability 1.1. Tests on server calculation have been made with float and double precision due to CUDA compute capability 1.3 of Tesla C1060 GPU.

The performance of ordinary kriging by using a PC (GPU, GeForce 9500, and CPU, AMD Athlon X2 4600) and a server (GPU, Tesla C1060, and CPU, Xeon 5600) is illustrated. The array dimensions considered for the tests run on the PC (float data type) and the server (float and double data types) were, respectively, 10×10 to 1500×1500 and 100×100 to 3000×3000.

3.1. Matrix multiplication

As Fig. 2 shows, the difference between the processing times for the matrix multiplication tests on GPU (GeForce 9500) and CPU (AMD Athlon X2 4600) is not relevant up to 450×450 dimensions according to the saved time (%), $(CPU_{time}-GPU_{time})/CPU_{time}$, curve included in the same figure. For larger dimensions, the computational time on the CPU increases rapidly. The corresponding saved time is 97.3%. When the matrix multiplication test is performed in the server, it can be verified for data type float that significant differences between the processing times on GPU (Tesla C1060) and CPU (Xeon 5600) appear for larger array dimensions than 1500×1500 (Fig. 3) with 99.8% of saved time. When data type double is considered in the test matrix multiplication performed in

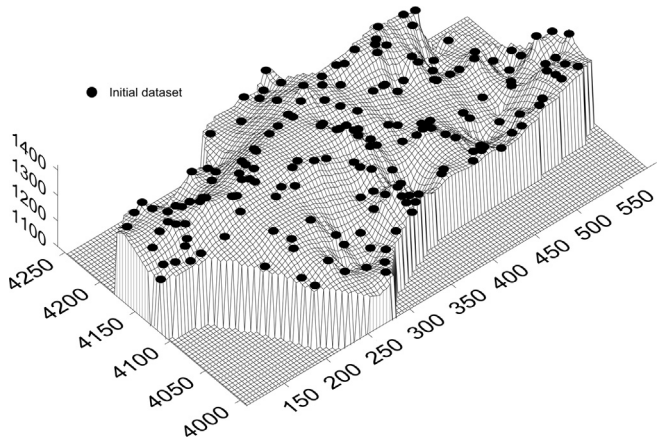

Fig. 1. Initial and interpolated dataset corresponding to accumulated reference evapotranspiration values recorded in Andalusia (southern Spain).

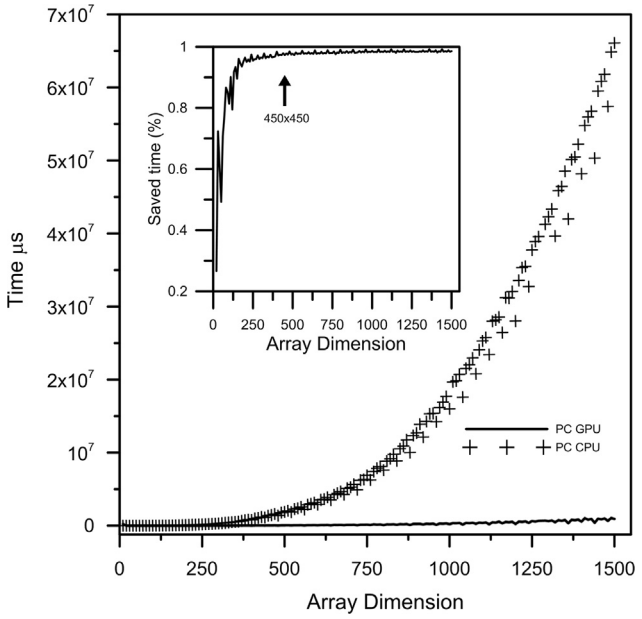

Fig. 2. Performance of multiplying matrices on PC (GPU: GeForce 9500 vs. CPU: AMD Athlon X2). The inner graphic shows the saved time (%) curve with the corresponding array dimension for saturation, 450×450 in this case.

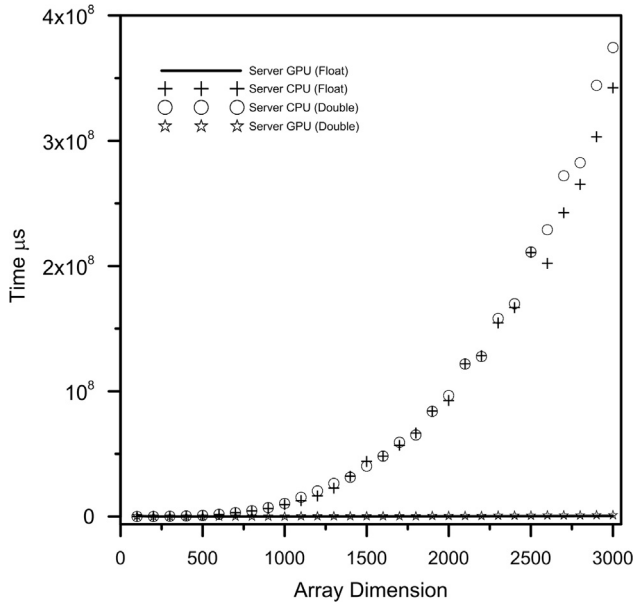

Fig. 3. Performance of multiplying matrices on server (GPU: Tesla C1060 vs. CPU: Xeon 5600) with float and double data types.

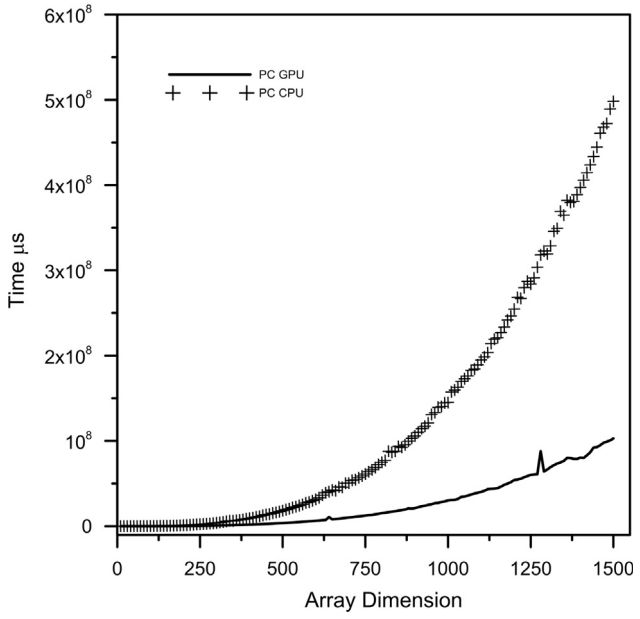

Fig. 4. Performance of inverse matrix on PC (GPU: GeForce 9500 vs. CPU: AMD Athlon X2).

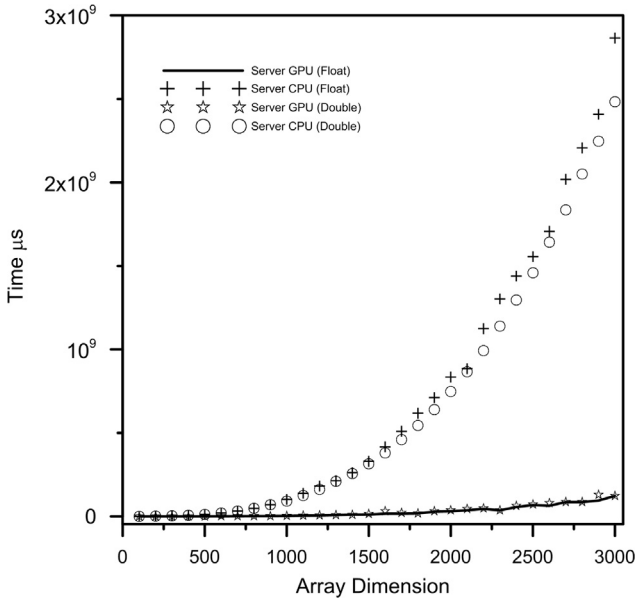

Fig. 5. Performance of inverse matrix on server (GPU: Tesla C1060 vs. CPU: Xeon 5600) with float double data types.

the server, relevant differences between the processing time have been detected for array dimensions above 1300×1300 (Fig. 3) with 99.6% of saved time. In all the cases, the CPU time increases rapidly, based on the algorithmic O(n^3) complexity vs. time on the GPU that is closer to an O($n \times \log(n)$) complexity. The time saved in the test is especially relevant for the data type float in the server (43.8 s)

3.2. Matrix inverse

As can be seen in Figs. 4 and 5, the array dimensions (300×300–500×500) that exhibit significant differences in running time are smaller than those determined for the matrix multiplication. In addition, the values for these array dimensions are similar unlike the situation found for matrix multiplication. It should be noted that the percentage of saved time calculated for GPU (GeForce 9500) and CPU (AMD Athlon X2 4600) is much lower (79.5%) than those reported for GPU (Tesla C1060) and CPU (Xeon 5600), considering both float and double data types ($\approx 97\%$). According to Figs. 4 and 5 it is shown that the computational time on the CPU increases rapidly due to the algorithmic O(n^3) complexity vs. time on the GPU that is closer to an O($n^2 \times \log(n)$) complexity. This fact may explain the smaller array dimensions found here compared to the rest of operations involved in OK that are of a lesser complexity and the higher saved times for server.

3.3. Matrix sum

It should be noted in Figs. 6 and 7, that significant differences between the processing times on GPU (Tesla C1060) and CPU (Xeon 5600) appear for array dimensions that are dissimilar when considering float (600×600) and double (1200×1200) data types. In addition, the saved time values are the lowest (65–80%) mainly due to the simplicity of this operation, which requires a low computational cost compared to the rest of operation but needs the same volume of data transfer between CPU and GPU. Figs. 6 and 7 reflect how the CPU time grows according to the algorithmic $O(n^2)$ complexity vs. computational time on the GPU that is closer to an $O(n \times \log(n))$ complexity.

There is interesting information about the time saved in the simulations when the larger array dimensions (1500×1500 for PC and 3000×3000 for server) are considered. It can be seen that multiplication and inverse matrix operations reduce their processing times, by factors of 5 and 6, respectively, when they are performed on the server compared to the result obtained for the PC. In addition, the running times corresponding to float and double data types on the server are similar. As a result, multiplication and matrix inverse operations are potential candidates to be performed on GPUs with the aim of accelerating the step corresponding to Eq. (5) in OK interpolation. However, the total running time for this step is more influenced by inverse matrix operation according to the difference found when comparing the performances of GPU-PC and GPU-server for array dimensions of 3000×3000. The time saved by using the GPU-server was 92 s against 0.9 s obtained for GPU-PC.

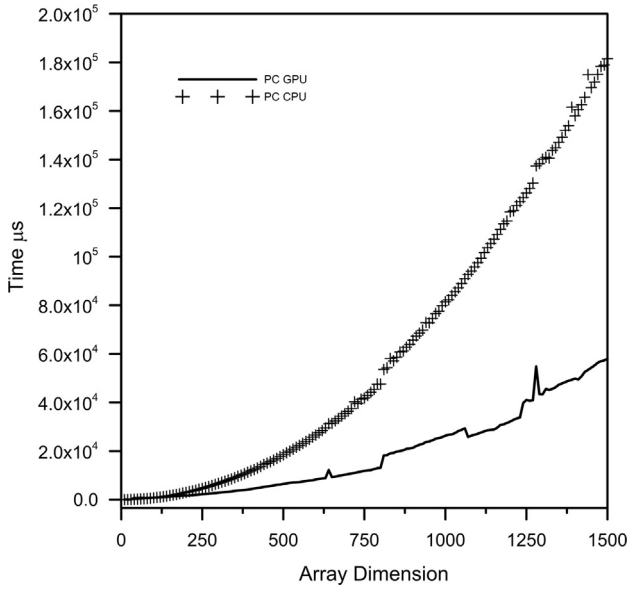

Fig. 6. Performance of summing matrices on PC (GPU: GeForce 9500 vs. CPU: AMD Athlon X2).

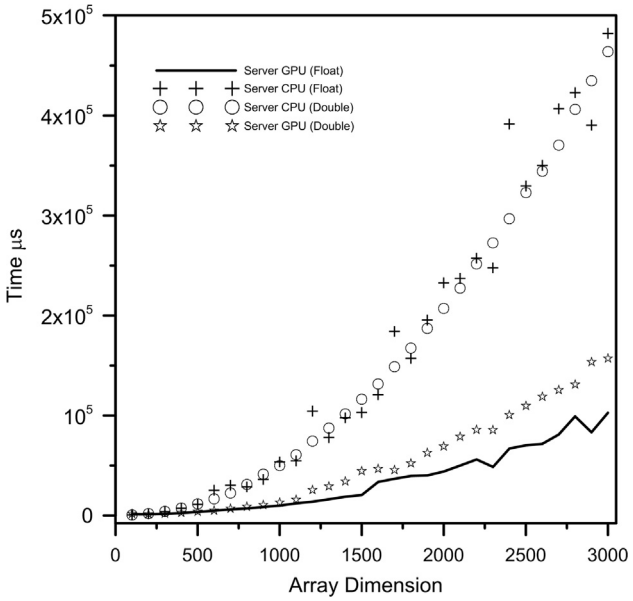

Fig. 7. Performance of summing matrices on server (GPU: Tesla C1060 vs. CPU: Xeon 5600) with float and double data types.

As it can be checked in Figs. 4 and 6, there are two GPU time peaks corresponding to array dimensions of 640×640 and 1280×1280, both being multiple of 16 which is the number of banks for CUDA devices 1.X is 16. All threads of a warp (execution unit that divides a block of threads in a multiprocessor) can access the shared memory with similar speed access to a record of processor. However, sometimes there are conflicts between threads accessing to different memory banks and, as a consequence, the access velocity decreases. With the aim of increasing the bandwidth, the shared memory of each multiprocessor is divided into memory modules that are equal in size called banks. They can be accessed in a parallel way by different threads of the block. Thus, if read or write accesses to n addresses are required, and each of these addresses are located in different memory banks, the bandwidth is n times the width of an individual bank. This situation occurs when two threads do not simultaneously access to the same memory bank. In case of conflict, the access is serialized, decreasing the bandwidth.

4. Conclusions

In this work, an analysis of running times for matrix operations involved in weights calculation and estimation for each unknown point, the most time-consuming step in ordinary kriging, was carried out on PC (GPU, GeForce 9500, and CPU, AMD Athlon X2 4600) and server (GPU, Tesla C1060, and CPU, Xeon 5600). The study at this level of detail, not taken into account in previous works, constitutes the main novelty of this research. It has been demonstrated that matrix inverse is the most relevant operation in terms of time performance for this step. As a consequence, the parallel implementation of matrix inverse will be enough to reduce the global execution time of ordinary kriging. In addition, it is possible to determine suitable array dimensions for parallelizing this operation, considering several data types at the same time. This fact leads to a significant improvement in computational performance when GPU are used, dismissing the effect of computational cost of conflict management.

The analysis framework shown here is exportable to other interpolation calculation methodologies using matrices. The low cost of GPUs and their increasing presence in research centers due to their portability, ensure the use of GPGPU computing in spatial interpolation methods. In this way, some limitations of parallelization methods designed for multi-CPU machines (i.e. trade-off between the speed and the array dimension, which is expensive) may be overcome. However, some further research needs to be done in order to provide solutions to some of the limitations detected in GPUs (i.e. the limited speed of data transfer between GPU and CPU and the relatively small amount of memory).

Acknowledgment

The “sequence-determines-credit” approach has been applied for the authors' order. The authors gratefully acknowledge support from the Consejería de Innovación, Ciencia y Empresa (Junta de Andalucía), ERDF, and ESF Project P08-RNM-3989.

References

Burrough, P.A., McDonnell, R.A., 1998. Principles of Geographical Information Systems. Oxford University Press, New York, USA (333pp).

Cheng, T., Li, D., Wang, Q., 2010. On parallelizing universal Kriging interpolation based on OpenMP. In: Proceedings of the 9th International Symposium on Distributed Computing and Applications to Business Engineering and Science (DCABES'10), IEEE, pp. 36–39.

Cheng, T., 2013. Accelerating universal Kriging interpolation algorithm using CUDA-enabled GPU. Comput. Geosci. 54, 178–183.

Fatone, L., Giacinti, M., Mariani, F., Recchioni, M.C., Zirilli, F., 2012. Parallel option pricing on GPU: barrier options and realized variance options. J. Supercomput. 62, 1480–1501.

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

Goovaerts, P., 1997b. Geostatistics for Natural Resources Evaluation. Oxford University Press, New York. (483pp).

Harris, M., 2005. Mapping computational concepts to GPUs. In: Fujii, J. (Ed.), ACM SIGGRAPH 2005 Courses, Los Angeles, California, 31 July–4 August 2005, SIGGRAPH'05. ACM Press, New York, NY, p. 50.

Isaaks, E.H., Srivastava, R.M., 1989. An Introduction to Applied Geostatistics. Oxford University Press, New York, USA.

Journel, A.G., Huijbregts, Ch.J., 1978. Mining Geostatistics. Academic Press, San Diego, USA.

Kerry, K.E., Hawick, K.A., 1998. Kriging Interpolation on High-Performance Computers, Technical Report DHPC-035. Department of Computer Science, University of Adelaide, Australia.

Kitanidis, P.K., 1997. Introduction to Geostatistics Applications in Hydrogeology. Cambridge University Press, New York, USA.

Kleijnen, J.P.K., 2009. Kriging metamodeling in simulation: a review. Eur. J. Operat. Res. 192, 707–716.

Li, J., Heap, A., 2008. A Review of Spatial Interpolation Methods for Environmental Scientists No. Record 2008/23. Geoscience Australia, Canberra.

Li, J., Heap, A., 2011. A review of comparative studies of spatial interpolation methods in environmental science: performance and impact factors. Ecol. Inf. 6, 228–241.

Li, J., Jiang, Y., Yang, C., Huang, Q., Rice, M., 2013. Visualizing 3D/4D environmental data using many-core graphics processing units (GPUs) and multi-core central processing units (CPUs). Comput. Geosci. 59, 78–89.

Lloyd, C.D., 2006. Local Models for Spatial Analysis. CRC Press, Belfast(244pp).

Matheron, G., 1963. Principles of geostatistics. Econ. Geol. 58, 1246–1266.

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

O'Sullivan, D., Unwin, D., 2002. Geographic Information Analysis. JohnWiley & Sons, Hoboken NewJersey(436pp).

Oliver, M.A., Webster, R., 1990. Kriging: a method of interpolation for geographical information systems. Int. J. Geogr. Inf. Sci. 4 (3), 313–332.

Pesquer, L., Cortes, A., Pons, X., 2011. Parallel ordinary kriging interpolation incorporating automatic variogram fitting. Comput. Geosci. 37, 464–473.

Srinivasan B.V., Duraiswami R., Murtugudde R., 2010. Efficient kriging for real-time spatio-temporal interpolation. In: Proceedings of the 20th Conference on Probability and Statistics in the Atmospheric Sciences, pp. 228–235.

Tahmasebi, P., Sahimi, M., Mariethoz, G., Hezarkhani, A., 2012. Accelerating geostatistical simulations using graphics processing units (GPU). Comput. Geosci. 46, 51–52.

Zhou, F., Guo, H.-C., Ho, Y.-S., Wu, C.-Z., 2007. Scientometric analysis of geostatistics using multivariate methods. Scientometrics 73, 265–279.