An out-of-core GPU approach for accelerating geostatistical interpolation

Victor Allombert, David Michéa, Fabrice Dupros, Christian Bellier, Bernard Bourgine, Hideo Aochi, Sylvain Jubertie

Procedia Computer Science

Volume 29, 2014, Pages 888–896

ICCS 2014. 14th International Conference on Computational Science

An out-of-core GPU approach for accelerating geostatistical interpolation

Victor Allombert[1,2], David Michea[2], Fabrice Dupros[2], Christian Bellier[2], Bernard Bourgine[2], Hideo Aochi[2], and Sylvain Jubertie[3]

[1] LACL, Universit de Paris-Est, France
[2] BRGM, BP 6009, 45060 Orlans Cedex 2, France
[3] LIFO, Universit d'Orlans, France

Abstract

Geostatistical methods provide a powerful tool to understand the complexity of data arising from Earth sciences. Since the mid 70's, this numerical approach is widely used to understand the spatial variation of natural phenomena in various domains like Oil and Gas, Mining or Environmental Industries. Considering the huge amount of data available, standard implementations of these numerical methods are not efficient enough to tackle current challenges in geosciences. Moreover, most of the software packages available for geostatisticians are designed for a usage on a desktop computer due to the trial and error procedure used during the interpolation. The Geological Data Management (*GDM*) software package developed by the French geological survey (BRGM) is widely used to build reliable three-dimensional geological models that require a large amount of memory and computing resources. Considering the most time-consuming phase of kriging methodology, we introduce an efficient out-of-core algorithm that fully benefits from graphics cards acceleration on desktop computer. This way we are able to accelerate kriging on GPU with data 4 times bigger than a classical in-core GPU algorithm, with a limited loss of performances.

Keywords: Geostatistics, Graphic processing units (GPU), Linear algebra, Out-of-core, Geological Data Management (GDM)

1 Introduction

Spatial interpolation plays a significant role in various domains of Earth sciences. One of the main objectives of geostatistics is to help researchers to build reliable model that take into account the spatial variation of the phenomena under study. This method offers many advantages such as the spatial structure of the phenomenon, the correlation between many variables, and a measure of uncertainty. A survey describing geostatistical interpolation can be found in [5, 10]. Lying at the heart of this numerical method, several costly linear algebra operations slow down the computation. Indeed, a linear system of size $(N * V)^2$ (with N the

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

doi:10.1016/j.procs.2014.05.080

number of data and V the number of correlated variables in the interpolation) must be solved which is an important limitation for large problems. Moreover, as the problem under study is three-dimensional, the amount of data required to build the interpolation and the limited amount of memory available to process them constitute another bottleneck.

Indeed, the majority of the software packages used for spatial interpolation and three-dimensional geological modeling are strongly limited by the resources available on standard desktop machines. This is mainly due to the way geostatistical models are built. The overall procedure relies on a trial and error strategy that strongly involves the researcher in the overall treatment. This induces a huge constraint on the ressources both in terms of memory available but also in terms of computing power. For instance, the *GDM*[1] software package that is developed by the French geological survey (BRGM) runs on desktop machines in order to process large scale three-dimensional domains.

Several references underline that GPU represents a promising way for kriging and very good speedups are described for instance in [4, 1]. Moreover, these computing resources of several Teraflops are available at the desktop computer level. Unfortunately, one of the bottlenecks remains, as a trade-off needs to be found between the opportunity to speed up the computation on the GPU and the limited amount of memory available on such accelerators.

In this paper, we design an algorithm to reduce the complexity of kriging methods on big data sets with respect to the constraint on the memory and computing ressources available on a desktop computer. In the first part, we will detail the standard geostatistical method and introduce the particular decomposition we have used in order to exhibit simple linear algebra operations. In the second part, we will call back the Cholesky factorization, then we will describe our approach on the out-of-core GPU Cholesky algorithm used to tackle the previous decomposition. We will mainly focus on the efficiency of this approach in terms of trade-off between the memory consumption and the overall speedup.

2 Kriging method

2.1 Numerical approach

In this paper we consider universal kriging, referenced as the best linear unbiased estimator, that has been formalized by Georges Matheron in 1963 [5]. As detailed in [9], this method allows to compute the estimation variance, which is a characterization of estimation error, a fundamental concept in geostatistics.

The variable of interest is denoted Z and its value at the point x is denoted $Z(x)$. It is considered as the sum of the mathematical expectation $m(x)$ and a residual $Y(x)$ that is a random function on the covariance $\sigma(x, x^{'})$. Thus, we obtain the following equation: $Z(x) = m(x) + Y(x)$. In our case, we are interested in the kriging of a unique point. Let's call $Z^*(x_0)$ the estimator of this point of interest. The mathematical expression of kriging a system is:

$$Z^*(x_0) = \sum_{\alpha=1}^{N} \lambda_\alpha Z(x_\alpha) \tag{1}$$

Where $Z(x_\alpha)$ are values for each points x_α, λ_α are unknown weights of points x_α and N the number of data.

[1] Geological Data Management : `http://gdm.brgm.fr`

Minimizing the estimation variance results in the following kriging system:

$$\begin{cases} \sum\limits_{\beta} \lambda_\beta \, \sigma_{\alpha\beta} + \sum_l \, \mu_l \, f^l(x_\alpha) = \sigma_{\alpha 0} & \text{for } \alpha = 1, \ldots, N \\ \\ \sum\limits_{\beta} \lambda_\beta \, f^l(x_\beta) = f^l(x_0) & \text{for } l = 0, \ldots, L \end{cases} \tag{2}$$

With $f^l(x_\alpha)$ representing the value of the base point function x_α (1 in this case), μ_l the Lagrange parameters and $\sigma_{\alpha\beta}$ the covariance between $Z(x_\alpha)$ and $Z(x_\beta)$.

2.2 Matrix expression of kriging

An interesting property of the kriging algorithm is the easiness of transforming it into matrix operations [2] :

$$\begin{bmatrix} \Sigma & F \\ F' & 0 \end{bmatrix} . \begin{bmatrix} \lambda \\ \mu \end{bmatrix} = \begin{bmatrix} \sigma_0 \\ f_0 \end{bmatrix} \tag{3}$$

With Σ the covariance matrix, F the base function matrix, λ the weights, μ the Lagrange parameters, σ_0 the vector of covariance between data and kriged points and f_0 the vector of base functions of the kriged point. This leads to solve a system of linear equations:

$$\big[A\big] . \big[X\big] = \big[B\big] \tag{4}$$

It is well known that the time complexity of solving such a system of linear equations using a regular Gaussian elimination is $\mathcal{O}(n^3)$. To avoid the direct resolution, we choose to split the system of equations into two sub-systems in order to reduce the size of the system to solve. The original system (3) can be split using the following transformations:

$$\big[\Sigma\big] . \big[\lambda\big] + \big[F\big] . \big[\mu\big] = \big[\sigma_0\big] \tag{A}$$

$$\big[F'\big] . \big[\lambda\big] = \big[f_0\big] \tag{B}$$

By transforming (A) such as: $\big[\lambda\big] = \big[\Sigma^{-1}\big] . \big[\sigma_0\big] - \big[\Sigma^{-1}\big] . \big[F\big] . \big[\mu\big]$.

Let: $\big[\lambda_{KS}\big] = \big[\Sigma^{-1}\big] . \big[\sigma_0\big]$ and $\big[w_1\big] = \big[\Sigma^{-1}\big] . \big[F\big]$.

So, (A) can be written: $\big[\lambda\big] = \big[\lambda_{KS}\big] - \big[w_1\big] . \big[\mu\big]$.

Then, we need to substitute the result $\big[\lambda\big]$ in (B) in order to obtain:

$$\big[F'\big] . \big[\lambda_{KS}\big] - \big[F'\big] . \big[w_1\big] . \big[\mu\big] = \big[f_0\big].$$

Let: $\big[R\big] = \big[F'\big] . \big[\lambda_{KS}\big] - \big[f_0\big]$ and $\big[Q\big] = \big[F'\big] . \big[w_1\big]$.

Then, equation (B) became: $\big[Q\big] . \big[\mu\big] = \big[R\big]$.

Resolving the last system allows us to obtain $[\mu]$ and then compute $\big[\lambda\big] = \big[\lambda_{KS}\big] - \big[w_1\big] . \big[\mu\big]$.

2.3 Resolution method

The covariance matrix Σ is composed by coefficients related to the distance between points and a coefficient defined by a variogram. Since $d(x_a, x_b) = d(x_b, x_a)$ and since these distances are positive, Σ is a positive symmetric definite matrix.

$$\begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} & \dots & \sigma_{1n} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} & \dots & \sigma_{2n} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} & \dots & \sigma_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{n1} & \sigma_{n2} & \sigma_{n3} & \dots & \sigma_{nn} \end{bmatrix} \tag{5}$$

Therefore, we are able to solve this particular system using Cholesky factorization, which is a optimized method for this kind of symmetric matrices [11], to resolve $\big[\Sigma\big] \,.\, \big[\lambda_{KS}\big] = \big[\sigma_0\big]$ and $\big[\Sigma\big] \,.\, \big[w_1\big] = \big[F\big]$. As explain in [6], the numerical stability of this method does not introduce errors in our interpolation. During the kriging, the solving of this particular operation is the most expensive part of the process in terms of time and memory needs. This is the reason why we will only focus on the Cholesky factorization, even if others GPU accelerated functions are used during the kriging algorithm.

3 GPU algorithm

The main idea of this algorithm is to use as much as possible the computing capacities of GPUs, even in their low-end versions. In order to allow users to use this GPU optimized algorithm, we chose CUDA compatible GPUs that offer to use the CuBLAS library[2]. The main idea is to define an out-of-core GPU algorithm able to run even if the whole data can not fit into the available memory.

3.1 Cholesky Algorithm

$$\begin{cases} L_{j,j} = \sqrt{A_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2} \\ L_{i,j} = \dfrac{1}{L_{j,j}} \left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k}\right) \end{cases} \tag{6}$$

We discuss the Cholesky factorization such as: $A = L\,L^T$ with A a symmetric positive definite matrix and L a lower triangular matrix. The algorithm can be described by a formula and resolved using the "line by line" Cholesky-Crout algorithm. We can also achieve the factorization using the well-known blocked algorithm [3] which is implemented into many linear algebra libraries. We select a pivoted Cholesky algorithm [8] that is more efficient in terms of GPU computations and memory bandwidth. Considering A, a block partitioned matrix, this algorithm will compute the current column using previously computed blocks with no needs of updating the trailing matrix.

[2] `https://developer.nvidia.com/cuBLAS`

So, at the k^{th} step, we have: $\begin{bmatrix} L_{11}^{(k-1)} & L_{12}^{(0)} & L_{13}^{(0)} \\ L_{21}^{(k-1)} & L_{22}^{(0)} & L_{23}^{(0)} \\ L_{31}^{(k-1)} & L_{32}^{(0)} & L_{33}^{(0)} \end{bmatrix}$

With $L_{ij}^0 = A_{ij}$ and L_{ij}^k is the of value L at the k^{th} iteration. So, according to the blocked numeration, for each step k we have:

$$L_{22}^{(k)} = L_{22}^{(0)} - L_{21}^{(k-1)} L_{21}^{(k-1)^T} \tag{7}$$

$$L_{22}^{(k)} = \text{Cholesky}(L_{22}^{(k)}) \tag{8}$$

$$L_{32}^{(k)} = L_{32}^{(0)} - L_{31}^{(k-1)} L_{21}^{(k-1)^T} \tag{9}$$

$$L_{32}^{(k)} = \text{Resolve}(X.L_{22}^{(k)} = L_{32}^{(k)}) \tag{10}$$

Obviously, these computations can be translated into some basic linear algebra subroutines such as *DSYRK* (rank-k update), *DPOTR2* (Cholesky factorization of a single block), *DGEMM* (standard matrix multiplication) and *DTRSM* (solver of matrix equations). In terms of complexity, this algorithm is similar to the original blocked version which runs in $\mathcal{O}(n^3)$ as explained in [11].

3.2 Out-of-core GPU algorithm

Most of the generic linear algebra subroutines are provided by NVIDIA in the CuBLAS library. Using them in such an algorithm is a good choice in terms of performance and easiness of implementation. We only need to implement *DPOTR2* on GPU in order to compute the Cholesky factorization of a single block.

The main idea of this algorithm is to load the maximum amount of data in the GPU's memory, to perform the computation, then to unload results. By using a tiling method we can iterate through the data and achieve the whole computation by splitting the given matrix into major sub-matrices. To achieve the best performance, we need to maximize the memory usage in order to minimize the load/unload steps.
According to the pivoted algorithm, we will define a work tile to specify the amount of data needed for each step. Thus, we are able to determine the size of each step and also the number of tiles which will fit into the GPU memory. In order to obtain the best performance let $BlockDim$, the size of a block, be constraint by the quantity of shared memory available on the GPU. Let $BlockSize$ be the number of elements of a block ($BlockDim^2$), N the dimension of the given matrix, and $NbBlocks$ the total number of blocks ($NbBlocks = N/BlocDim$) then $BlocSize * (k+1) * (NbBlocs - k)$ corresponds to the quantity of elements needed at the k^{th} step. So, a work tile consists of the maximal number of steps that can be loaded on GPU's memory. As kshown in Figure 1, the computation is split into several work tiles which are successively computed at each step.

The main constraint of this algorithm is given by the biggest step of computation. The computation that requires the more data is reached when k equals $(NbBlocks - 1)/2$. Indeed, we must be able to store this greedy step on the GPU in order to compute the factorization. Otherwise, the entry is too big and we can not go any further. It is possible to exceed this limitation by using an out-of-core GPU matrix multiplication, but this issue is out of the scope of this paper.

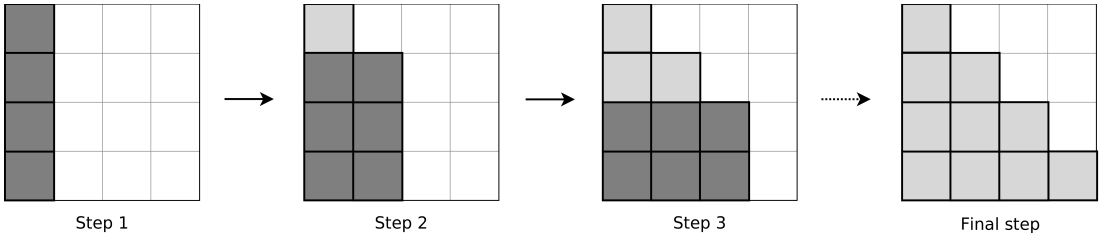

Figure 1: Work tiles (Grey) over several steps

The pseudo-code of the proposed out-of-core GPU pivoted Cholesky algorithm could be described as follows:

Algorithm 1: Out-of-core GPU Cholesky factorization

```
Data: A : Input matrix
Result: L : Cholesky factorization of A
while Computation not finished do
    Define a work tile
    Load data on GPU
    foreach column do
        DSYRK
        DPOTF2
        DGEMM
        DTRSM
    end
    Unload data from GPU
end
```

We can observe that this algorithm is not using a streaming method in order to overlap memory transfers and computations. As this algorithm has many data dependencies, using smaller data blocks and reordering them with asynchronous streams may not be as fast as the synchronous version. Thanks to its out-of-core capability, this algorithm allows us to run a GPU accelerated version of kriging on various configuration of memory or power without manual tuning and regardless of the amount of data.

4 Experimental results

We detail the results obtained with our out-of-core algorithm in this section. We focus on memory footprint reduction as the speedups obtained with respect to standard GPU implementations have already been extensively discussed in several others related works [1, 4, 7]. We consider several matrices arising from various application domains. The matrices are in double-precision and positive-definite. Table 1 summarizes their characteristics with N representing the number of lines for each square matrix. A standard NVIDIA Quadro *K4000* graphics card has been used to run the experiments. This card represents the mid-range graphics card for a desktop machine dedicated to 3D processing. The NVIDIA Quadro *K4000* has 768 cores and 3 gigabytes of DDR5 memory. The bandwidth for the memory transfers is 135 gigabytes per second.
We artificially reduce the memory available on the graphics card (from 2.5 GB to 800 MB) in

Name	N	size (GB)
Matrix_1	14016	1.46
Matrix_2	15520	1.79
Matrix_3	20032	2.98
Matrix_4	23040	3.95
Matrix_5	24522	4.48
Matrix_6	26048	5.05
Matrix_7	27552	5.65

Table 1: Characteristics of the matrices used for the experiments.

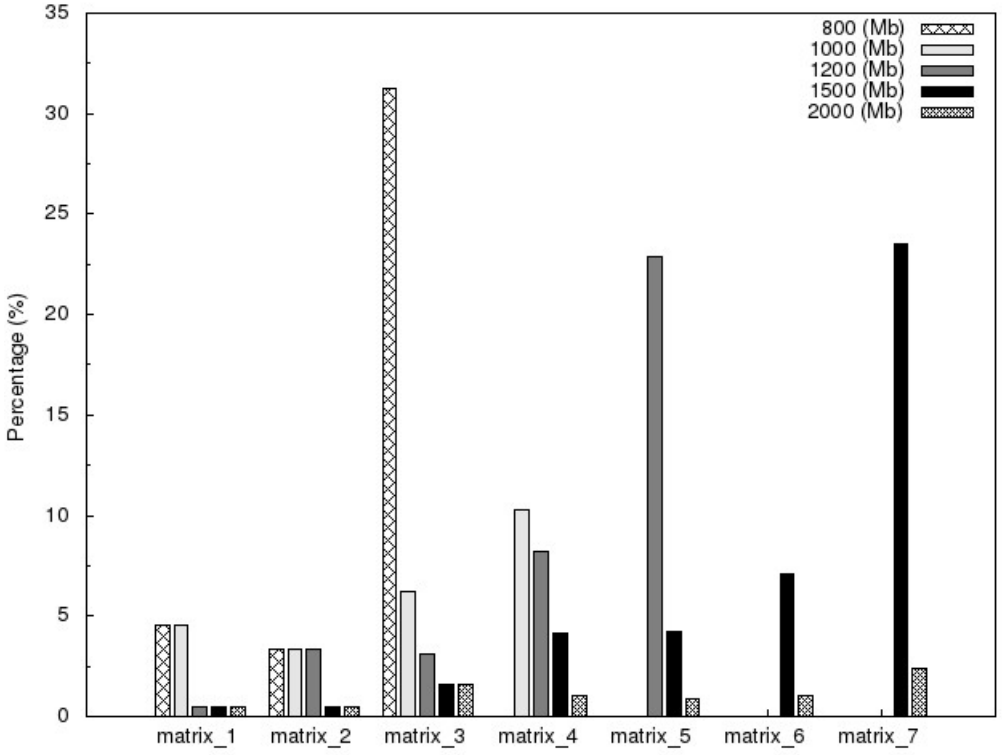

Figure 2: Performance of the out-of-core GPU algorithm. We represent the performance loss (%) in comparison with the use of all the memory available on graphics card.

order to evaluate the efficiency of our out-of-core algorithm. Figure 2 represents the performance loss (%) in comparison with the use of all the memory available on the graphics card. For example: with Matrix_3, we can see that computations are respectively 32%, 6% and 3% slower with 800, 1000 and 1200 Mb of memory, compared to the reference time reached when all the data can fit into the memory. The Cholesky decomposition could only be performed for the first three matrices for all memory configurations (until 2.98 GB of memory consumption considering a minimum of 800 MB of allocatable space on the graphics device). For larger matrices, the remaining memory space is not large enough to store the minimum amount of blocks required during the out-of-core procedure. In terms of memory consumption reduction,

the best result correspond to the larger matrix (Matrix_7) that could be decomposed with a reduction of a factor 3.78 of the standard amount of memory required for such a factorization. At this stage, it is important to remind that our algorithm relies on a tradeoff between the reduction of the memory consumption and a limited increase of the total elapsed time. Indeed, as we reduce the amount of memory available, the elapsed time is increasing for all problem sizes. This loss of performance is rather limited, the worst result is obtained with a matrix of size 2.98 GB that is decomposed using 800 MB of memory. In this case, the loss of performance is close to 34%. The average loss measured during the experiments with various problem sizes is 10%. These results appear pretty good in comparison with the tremendous gain coming from the use of GPU as accelerators for computational geostatistics on desktop machines.

5 Conclusion

Geological Data Management (*GDM*) is a commercial software package stemming from the French Geological Survey (BRGM) know-how in three-dimensional geological modeling. Limitations at the algorithmic level prevent the use of *GDM* for large scale problems arising in Earth sciences. The modification of the kriging method in order to obtain two sub-systems allows us to decrease the complexity of the algorithm. We also adapt the Cholesky algorithm on NVIDIA graphics cards to accelerate the computations during the kriging. This version of the algorithm also allows us to check the error of estimation during the computation: an essential issue for geostatistical interpolation. Finally, thanks to a controlled out-of-core strategy, we are able to tackle 4 times larger problems with a limited loss of performance. This approach relies on NVIDIA CUDA architecture but can be easily extended to the OpenCL programming model. We are also interested in approximated methods, like the incomplete Cholesky factorization, to reduce the system size in order to display real-time interpolations. These features will be added in the next release of *GDM*.

Acknowledgements

This work is partially funded by the European FP7 IRSES project HPC-GA (High-Performance Computing for Geophysics Applications). The authors thank Franois Courteille, Senior Solution Architect at NVIDIA, for advices on the implementation.

References

[1] Tangpei Cheng. Accelerating universal kriging interpolation algorithm using CUDA-enabled GPU. *Computers and Geosciences*, 54:178–183, 2013.

[2] Jean Paul Chils and Pierre Delfiner. *Geostatistics: Modeling Spatial Uncertainty*. Wiley, 2 edition, 2012.

[3] Jaeyoung Choi, Jack J. Dongarra, L. Susan Ostrouchovn, Antoine Petitet, David Walker, and Clint Whaley. Design and implementation of the scalapack LU, QR an Cholesky factorization routines. *Scientific Programming*, 1994.

[4] E. Gutirrez de Rav, F.J. Jimnez-Hornero, A.B. Ariza-Villaverde, and J.M. Gmez-Lpez. Using general-purpose computing on graphics processing units (GPGPU) to accelerate the ordinary kriging algorithm. *Computers and Geosciences*, 2014.

[5] Matheron Georges. *Principles of geostatistics*, volume 58. Economic geology, 1963.

[6] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, 2002.

[7] Hatem Ltaief, Stanimire Tomov, Rajib Nath, and Jack Dongarra. Hybrid multicore cholesky factorization with multiple GPU accelerators. *IEEE Transaction on Parallel and Distributed Computing*, 2010.

[8] Craig Lucas. Lapack-style codes for level 2 and 3 pivoted cholesky factorizations. *Numerical Analysis Report*, 2004.

[9] Georges Matheron. *The Theory of Regionalized Variables and Its Applications*. cole national suprieure des mines, 1971.

[10] M.A. Oliver and R. Webster. A tutorial guide to geostatistics: Computing and modelling variograms and kriging. *CATENA*, pages 56 – 69, 2014.

[11] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, 1997.