Proceedings of the 11th International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2011 26–30 June 2011.

Solving Differential Riccati Equations on Multi-GPU Platforms

Peter Benner[1], Pablo Ezzatti[2], Hermann Mena[3], Enrique S. Quintana-Ortí[4] and Alfredo Remón[4]

[1] *Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstr. 1, 39106 Magdeburg, (Germany)*

[2] *Centro de Cálculo-Instituto de Computación, Universidad de la República, 11.300-Montevideo (Uruguay)*

[3] *Departamento de Matemática, Escuela Politécnica Nacional, EC1701 Quito (Ecuador)*

[4] *Dpto. de Ingeniería y Ciencia de Computadores, Universidad Jaime I, 12.071-Castellón (Spain)*

emails: benner@mpi-magdeburg.mpg.de, pezzatti@fing.edu.uy, hermann.mena@epn.edu.ec, quintana@icc.uji.es, remon@icc.uji.es

Abstract

Differential Riccati Equations (DREs) arise in many scientific and engineering applications. Particularly, they play an important role in control problems, where a finite-time horizon of integration is considered. In this paper, we present several high-performance implementations of the Rosenbrock method for multi-core and graphic processors (GPUs). The Rosenbrock method for solving DREs is an iterative technique that requires the solution of a Lyapunov equation per step, which in our approach is solved via the highly parallel sign function method. Mainly, this is an iterative procedure, where the most time-consuming operation is the computation of a matrix inverse per step. Hence, an efficient implementation of the Rosenbrock method can be obtained providing an efficient matrix inversion kernel. We analyze two different approaches for the matrix inversion: the traditional method based on the LU factorization and the Gauss-Jordan elimination method. Numerical experiments show that the execution time can be drastically reduced by off-loading part of the computations to one or more GPUs.

Key words: Differential Riccati equations, Rosenbrock methods, matrix sign function, graphics processors, multi-core processors.

 ISBN: 978-84-614-6167-7

1 Introduction

Consider the symmetric differential Riccati equation (DRE)

$$\begin{aligned} \dot{X}(t) &= Q + X(t)A + A^T X(t) - X(t)SX(t) \equiv F(X(t)), \\ X(t_0) &= X_0, \end{aligned} \tag{1}$$

where $t \in [a, b]$, $A \in \mathbb{R}^{n\times n}$, $Q \in \mathbb{R}^{m\times n}$, $S \in \mathbb{R}^{n\times m}$, and $X(t) \in \mathbb{R}^{m\times n}$. The solution of (1) exists and it is unique, e.g., [1, Thm. 4.1.6]. Symmetric DREs arise in linear-quadratic optimal control problems such as LQR and LQG design with finite-time horizon, in H_∞ control of linear-time varying systems as well as in differential games, e.g., [1, 2]. Unfortunately, in most control problems fast and slow modes occur. Then, the DRE will be fairly stiff, so implicit methods have to be used for solving the DRE numerically. Matrix-valued algorithms based on generalizations of the Rosenbrock methods have been proved to yield accurate solutions for large-scale DREs arising in optimal control problems for parabolic partial differential equations [4, 6]. Using Rosenbrock methods for solving DREs requires the solution of one Lyapunov equation per iterative step. The Lyapunov equation is usually solved by exploiting the structure of the matrices (sparsity, symmetry, low rank), see, e.g., [3]. However, in some applications, a large interval of integration has to be considered and/or a thiner mesh is required to describe the solution accurately, which turns these methods un affordable due to their high computational cost.

We will focus on the method of order one, i.e, the so-called linearly implicit Euler method, on equidistant meshes. For practical purposes, the method should be used with adaptive time steps as suggested in [4]. Here we will focus on the parallel perfomance of the computation of one time step, which is independet of the grid chosen. Thus, we keep things as simple as possible for this purpose. The resulting Lyapunov equation is solved via the highly parallel sign function method, where the most time-consuming operation is the computation of a matrix inversion per step. We analyze two different approaches for the matrix inversion: the traditional method based on the LU factorization and the Gauss-Jordan elimination method.

We present several high-performance implementations of the Rosenbrock method for a hybrid platform composed of multi-core processors and several graphics processors. The use of high-performance kernels of several linear algebra libraries, and several optimization techniques, like padding or the concurrent computation in all the devices of the platform, report a remarkable performance in the developed implementations.

This paper is organized as follows. In Section 2, we briefly describe the application of the Rosenbrock method of order one to DREs and the sign function method for solving Lyapunov equations. The different implementations are described in Section 3. Then, in Section 4, numerical experiments showing the performance of the proposed methods are included. Finally, conclusions and future work are pointed out.

 ISBN: 978-84-614-6167-7

2 Numerical solution of DREs

We focus on the solution of DREs arising in optimal control for ordinary differential equations. Using the Rosenbrock method of order one for solving an autonomous symmetric DRE of the form (1) yields:

$$\tilde{A}_k^T X_{k+1} + X_{k+1}\tilde{A}_k = -Q - X_k S X_k - \frac{1}{h} X_k \tag{2}$$

where $X_k \approx X(t_k)$ and $\tilde{A}_k = A - SX_k - \frac{1}{2h}I$; see [6, 4] for details. In addition, we assume that

$$\begin{aligned} Q &= C^T C, & C &\in \mathbb{R}^{p\times n}, \\ S &= BB^T, & B &\in \mathbb{R}^{n\times m}, \\ X_k &= Z_k Z_k^T, & Z_k &\in \mathbb{R}^{n\times z_k}, \end{aligned}$$

with p, m, $z_k \ll n$. If we denote $N_k = [\,C^T,\ Z_k(Z_k^T B),\ \sqrt{h^{-1}}Z_k\,]$, then the Lyapunov equation in (2) results in

$$\tilde{A}_k^T X_{k+1} + X_{k+1}\tilde{A}_k = -N_k N_k^T, \tag{3}$$

where $\tilde{A}_k = A - B(Z_k(Z_k^T B))^T - \frac{1}{2h}I$. Observing that $\operatorname{rank}(N_k) \leq p + m + z_k \ll n$, we can efficiently solve (3) with the sign function method as described in the following subsection. This is stated in Algorithm 1

Algorithm 1: Rosenbrock method of order one for DREs

Data: $A \in \mathbb{R}^{n\times n}$, B, C, Z_0 satisfying (2), $t \in [a, b]$, and step size h.
Result: (Z_k, t_k) such that $X_k \approx Z_k Z_k^T$, $Z_k \in \mathbb{R}^{n\times z_i}$ with $z_i \ll n$.

```
1 begin
2     t_0 := a
3     for k := 0 to ⌈(b−a)/h⌉ do
4         N_k := [C^T, Z_k(Z_k^T B), √(h^{-1}) Z_k]
5         Compute Z_{k+1} such that the low rank factor product Z_{k+1} Z_{k+1}^T
          approximates the solution of Ã_k^T X_{k+1} + X_{k+1} Ã_k = −N_k N_k^T.
6         t_{k+1} := t_k + h.
7     end
8 end
```

2.1 The sign function method

The matrix sign function is an efficient tool to solve stable Lyapunov equations. There exist several iterative schemes for the computation of this matrix function. Among those, the Newton iteration described in Algorithm 2 is specially appealing for its simplicity, efficiency, parallel performance, and asymptotic quadratic convergence [7].

 ISBN: 978-84-614-6167-7

Algorithm 2: Matrix sign function for Lyapunov equations

Data: $A \in \mathbb{R}^{n \times n}$, $N \in \mathbb{R}^{j \times n}$.
Result: $\tilde{S}$ such that $\tilde{S}^T \tilde{S} \approx X$ and $A^T X + XA = NN^T$.

begin
  $A_0 := A$
  $\hat{S}_0 := N^T$
  $k := 0$
  repeat
    $A_{k+1} := \frac{1}{\sqrt{2}} \left(A_k / c_k + c_k A_k{}^{-1} \right)$
    Compute the rank-revealing QR (RRQR) decomposition
    $\frac{1}{\sqrt{2c_k}} \left[\tilde{S}_k, \; c_k \tilde{S}_k (A_k^{-1})^T \right] = Q_s \begin{bmatrix} U_s \\ 0 \end{bmatrix} \Pi_s$.
    $\tilde{S}_{k+1} := U_s \Pi_s$
    $k := k + 1$
  until $\sqrt{\frac{\|A_{k+1} + I\|_\infty}{n}} < \tau \|A_k\|_\infty$
end

Algorithm 2 roughly requires $2n^3$ floating-point arithmetic operations (flops) per iteration.

On convergence, $\tilde{S}$ is the factor of the approximated solution and satisfies $X \approx \tilde{S}^T \tilde{S}$. Convergence can be accelerated using several techniques [7]. In our approach, we employ a scaling defined by the parameter

$$c_k = \sqrt{\|A_k{}^{-1}\|_\infty / \|A_k\|_\infty}.$$

In the convergence test, τ is a tolerance threshold for the iteration that is usually set as a function of the problem dimension and the machine precision ϵ.

3 High-performance implementations

In this section we describe several high-performance codes for the solution of DREs. All the implementations use linear algebra libraries (e.g., MKL or CUBLAS) and employing single precision arithmetic. A double precision accurate solution can be obtained, at a low-computational cost, applying an iterative refinement technique like the one proposed in Benner et al. [8].

The solution of the Lyapunov equation is the most expensive part when solving DREs using the Rosenbrock method (see Algorithm 1). Particularly, most of the computational cost is due to the matrix inversion required at each iteration of the sign function method (Algorithm 2). Thus, we have optimized the matrix inversion process. The rest of operations are mainly matrix-matrix products of small matrices, which can be efficiently computed on the multi-core architecture invoking a multi-thread implementation of BLAS.

 ISBN: 978-84-614-6167-7

We present three different algorithms/implementations of the Rosenbrock method which, basically differ in the procedure to compute the matrix inverse. All the remaining steps in these algorithms/implementations are performed on the CPU and therefore, we do not go into detail.

3.1 Implementation on a multi-core CPU: `Ros(CPU)`

The traditional approach to compute the inverse of a matrix $A \in \mathbb{R}^{n \times n}$ is based on Gaussian elimination (i.e., the LU factorization), and consists of the following three steps:

1. Compute the LU factorization $PA = LU$, where $P \in \mathbb{R}^{n \times n}$ is a permutation matrix, and $L, U \in \mathbb{R}^{n \times n}$ are unit lower and upper triangular factors, respectively, see [9].
2. Invert the triangular factor $U \rightarrow U^{-1}$.
3. Solve the system $XL = U^{-1}$ using backward substitution for X.
4. Undo the permutations $A^{-1} := XP$.

LAPACK [10] is a high-performance linear algebra library, which provides efficient routines to compute the previous steps. In particular, routine `getrf` yields the LU factorization (with partial pivoting) of a nonsingular matrix (Step 1), while routine `getri` computes the inverse matrix of A using the LU factorization obtained by `getrf` (Steps 2–4). The computational cost of computing a matrix inverse following the previous four steps is $2n^3$ flops.

3.2 Implementation on a many-core GPU: `Ros(GPU)`

The traditional algorithm for matrix inversion (see Section 3.1) shows some limitations from the high-performance computing point of view. There is no possibility to compute concurrently several steps, so parallelism has to be extracted within each step. Also, step 2 and 3 are unbalanced, because they operate on triangular matrices. The Gauss-Jordan elimination algorithm (GJE) is a reordering of the computations performed by the Gaussian elimination procedure for matrix inversion. Thus, the arithmetic cost of matrix inversion using GJE is the same as the one based on the LU factorization. However, the GJE method is better suited for parallelization. We present an implementation of the GJE method on a GPU.

Algorithm 4 illustrates a blocked version of the GJE for matrix inversion. A description of the unblocked version (`GEINGJ`), called from inside the blocked one, can be found in [11]; for simplicity, the application of pivoting during the factorization is concealed; see [11]. The bulk of computations is cast in terms of matrix-matrix products; an operation which exhibits a high degree of concurrency. Therefore, GJE is a highly appealing method for matrix inversion on emerging architectures like GPUs, where many computational units are available and a highly tuned implementation of the matrix-matrix product is available (e.g., in the NVIDIA CUBLAS library).

 ISBN: 978-84-614-6167-7

Algorithm 3: Blocked Gauss-Jordan elimination algorithm for matrix inversion

Data: $A \in \mathbb{R}^{n \times n}$.
Result: $A := A^{-1}$.

1 **begin**
2 $t_0 := a$
3 **for** $k := 1$ *to* $\lceil \frac{n}{b} \rceil$ **do**
4 $A \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$ where $A_{00} \in \mathbb{R}^{(k-1)b \times (k-1)b}$, $A_{11} \in \mathbb{R}^{b \times b}$
5 $[A_{01},\ A_{11},\ A_{21}]^T := \texttt{GEINGJ}([A_{01},\ A_{11},\ A_{21}]^T)$
6 $A_{00} := A_{00} + A_{01}A_{10}$
7 $A_{20} := A_{20} + A_{21}A_{10}$
8 $A_{10} := A_{11}A_{10}$
9 $A_{02} := A_{02} + A_{01}A_{12}$
10 $A_{22} := A_{22} + A_{21}A_{12}$
11 $A_{12} := A_{11}A_{12}$
12 **end**
13 **end**

3.3 Implementation on multi-GPU: `Ros(MGPU)`

`Ros(MGPU)` is in essence an extension of the `Ros(GPU)` solver which targets platforms equipped with multiple GPUs. As mentioned before, the critical operation of the Rosenbrock method is the matrix inversion. In this variant we employ a highly tuned implementation for this operation that targets a platform with several GPUs connected to a single CPU. This matrix inversion routine includes several optimization techniques, in particular, the use of optimized CUBLAS kernels, padding to accelerate the GPU-memory access, an optimized task schedule that permits the concurrent computation in all the devices, a look-ahead approach to minimize the negative impact from the critical path, a cyclic distribution that maximizes load balance, and the use of two block-sizes that allows to adapt the routine execution to the particularities of the CPU and the GPU architectures simultaneously. See [12] for more details on the matrix inversion routine.

4 Numerical Results

In this section we evaluate the performance of the implementations introduced in Section 3 on a hybrid platform composed of a multi-core CPU and several many-core GPUs. Particularly, the experiments are performed on a computer with two INTEL Xeon QuadCore E5530 processors at 2.27GHz, connected to an NVIDIA Tesla C1060 (consisting of four NVIDIA Tesla S1060 GPUs) via a PCI-e bus (more details about the platform can be found in Table 4).

 ISBN: 978-84-614-6167-7

Algorithm 4: Blocked Gauss-Jordan elimination algorithm for matrix inversion

Require: $A \in \mathbb{R}^{n \times n}$

1: $t_0 := a$

2: **for** $k := 1$ to $\lceil \frac{n}{b} \rceil$ **do**

3: Partition $A \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$ where $A_{00} \in \mathbb{R}^{(k-1)b \times (k-1)b}$, $A_{11} \in \mathbb{R}^{b \times b}$

4: $[A_{01},\ A_{11},\ A_{21}]^T := \texttt{GEINGJ}([A_{01},\ A_{11},\ A_{21}]^T)$

5: $A_{00} := A_{00} + A_{01}A_{10}$

6: $A_{20} := A_{20} + A_{21}A_{10}$

7: $A_{10} := A_{11}A_{10}$

8: $A_{02} := A_{02} + A_{01}A_{12}$

9: $A_{22} := A_{22} + A_{21}A_{12}$

10: $A_{12} := A_{11}A_{12}$

11: **end for**

Processors	#proc.	#cores (per proc.)	Frequency (GHz)	L2 cache (MB)	Memory (GB)
INTEL QuadCore E5530	2	4	2.27	8	48
NVIDIA TESLA c1060	4	240	1.3	–	(4x4)16

Table 1: Hardware employed in the experiments.

A multi-thread version of the INTEL MKL library (version 10.2) provided the necessary LAPACK and BLAS kernels for the CPU, and NVIDIA CUBLAS (version 2.1) for the GPU computations.

4.1 Test examples

We evaluate the performance of the implementations using two problems from the *Oberwolfach Model Reduction Benchmark Collection*[1]: the semi-discretized heat transfer problem for the optimal cooling of steel profiles (STEEL), and the butterfly gyro problem (GYRO). In the following we briefly describe these two models.

4.1.1 STEEL

This model arises in a manufacturing method for steel profiles. The goal is to design a control that yields moderate temperature gradients when the rail is cooled down. The mathematical model corresponds to the boundary control for a 2-D heat equation. A finite element discretization, followed by adaptive refinement of the mesh, results

[1] `http://www.imtek.de/simulation/benchmark/`.

 ISBN: 978-84-614-6167-7

in several instances of this benchmark. We employed two instances of this problem, STEEL$_S$ and STEEL$_L$. For both DREs, $m = 7$ and $p = 6$. The order of the system (size of A) is n=5,177 for the STEEL$_S$ instance and 20,209 for the STEEL$_L$ instance.

4.1.2 GYRO

The Butterfly is a vibrating micro-mechanical gyro that has been proposed for inertial navigation applications. The model is a simplified version which includes the pure structural mechanics problem only. It is designed to test model reduction approaches. The dimension of the mechanical system described by a system of second-order differential equations is $n = 17,361$. Hence, for the numerical experiments we first need to transform the system into a first order one, the main dimension of the transformed system is thus $n = 34,722$.

4.2 Numerical Results

All the experiments were done using single-precision arithmetic, and all the reported execution times include the overhead introduced by data transfers between the CPU and the GPUs memory spaces.

We first study the STEEL case, because our three implementations can tackle this problem using the available hardware (the large dimension of the matrices involved in GYRO did not allow us to solve this problem using `Ros(GPU)`).

Tables 2 and 3 present the total time as well as the time spend in the sign function Lyapunov solver (F_{sign}) required to solve the DRE associated with both instances of STEEL on the $[0, 1]$ interval with a stepsize of $h = 0.1$.

`Ros(CPU)` Time		`Ros(GPU)` Time		Speed-up		`Ros(MGPU)` Time		Speed-up	
F_{sign}	Total	F_{sign}	Total	F_{sign}	Total	F_{sign}	Total	F_{sign}	Total
92.06	94.65	47.55	50.13	1.94	1.89	26.44	28.99	3.48	3.27

Table 2: Execution time (in seconds) and speed-up obtained for the STEEL$_S$ benchmark.

`Ros(CPU)` Time		`Ros(GPU)` Time		Speed-up		`Ros(MGPU)` Time		Speed-up	
F_{sign}	Total	F_{sign}	Total	F_{sign}	Total	F_{sign}	Total	F_{sign}	Total
8703.2	9061.6	3406.5	3712.7	2.56	2.44	1338.2	1688.5	6.50	5.37

Table 3: Execution time (in seconds) and speed-up obtained for the STEEL$_L$ benchmark.

The experimental results in the tables show that most of the time (approximately 97%) is dedicated to the computation of the sign function method. They also shown how this computation can be drastically accelerated using graphics processors. The

 ISBN: 978-84-614-6167-7

hybrid CPU-GPU implementation accelerates the computation time of F_{sign} between 1.94× and 2.56×. The use of the four GPUs available on the platform enhances this acceleration factor to 3.48× for the STEEL$_S$ problem and 6.5× for the STEEL$_L$ case.

In a second experiment we evaluate the performance of the CPU and the multi-GPU implementations (`Ros(CPU)` and `Ros(MGPU)`) using the GYRO example. Table 4 summarizes the results obtained for this benchmark. Once more, most of the time is spent in the computation of F_{sign} and important time reductions are obtained from the use of the four GPUs. In this problem routine `Ros(MGPU)` computes F_{sign} approximately 9× faster than the `Ros(CPU)` implementation.

`Ros(CPU)` Time		`Ros(MGPU)` Time		Speed-up	
F_{sign}	Total	F_{sign}	Total	F_{sign}	Total
44919.69	46136.36	4986.36	6141.64	9.01	7.51

Table 4: Execution time (in seconds) and speed-up obtained for the GYRO benchmark.

From the results we can conclude that the hybrid CPU-GPU implementation reports an important reduction of the computational time, but the amount of memory limits its application to small and medium problems. The multi-GPU permits to target larger problems while simultaneously providing a notorious performance, e.g., accelerating the execution of the GYRO problem 7.51×.

Finally, we remark that in the `Ros(CPU)` implementation based on the LAPACK routines for the matrix inversion, approximately the 97% from the execution time is dedicated to the computation of the sign function. Despite the effort to optimize this operation in the GPU-based implementations (`Ros(GPU)` and `Ros(MGPU)`), the sign function still concentrates a 80% of the total time for the fastest implementation and the largest problem studied in this work.

5 Conclusions and future work

The numerical results show the dramatic acceleration when using GPUs for solving DREs. The single GPU-based implementation reports a speed-up of 2.5× over the multi-core CPU implementation based on LAPACK. The multi-GPU implementation, executed on four GPUs, increases this ratio up to 7.5× Another remarkable advantage from the use of several GPUs is the increment of the aggregated memory. As each device includes its own memory, an increase in the number of devices reports an increment in the amount of available memory and, hence, also the dimension of the affordable problems. This is specially important in many optimal control problems, where the dimension of the related mathematical models is extremely large.

The use of four GPUs reduces the percentage of time dedicated to compute matrix inverses from 97% to 80%. The use of more GPUs will probably keep decreasing this percentage. In general, GPUs show an excellent relationship between cost and com-

 ISBN: 978-84-614-6167-7

puting power, achieving more FLOPS per dollar than the traditional high performance architectures in many application areas, e.g., in the execution of dense linear algebra operations.

The acceleration of other stages involved in Algorithm 1, the implementation of higher order methods and a stepsize control will be discussed in future works.

Acknowledgments

Enrique S. Quintana-Ortí and Alfredo Remón were supported by projects PROMETEO 2009/013 and CICYT TIN2008-06570-C04. This work was partially done while Hermann Mena was visiting the Universidad Jaime I with the support from the program "Pla de suport a la investigació 2009" from Universidad Jaime I, and continued while Pablo Ezzatti, Hermann Mena, and Alfredo Remón were visiting the Max Plank Institute (MPI) in Magdeburg. Pablo Ezzatti, Hermann Mena, and Alfredo Remón gratefully acknowledge support received from the MPI.

References

[1] H. Abou-Kandil, G. Freiling, V. Ionescu and G. Jank, *Matrix Riccati Equations in Control and Systems Theory*, Birkhäuser, Basel, Switzerland, 2003.

[2] A. Ichikawa and H. Katayama, *Remarks on the time-varying H_∞ Riccati equations*, Sys. Cont. Lett. 37(5):335-345, 1999.

[3] P. Benner, J-R. Li and T. Penzl, *Numerical Solution of Large Lyapunov Equations, Riccati Equations, and Linear-Quadratic Control Problems*, Numerical Linear Algebra with Applications, Vol. 15, No. 9, pp. 755-777, 2008.

[4] H. Mena, *Numerical Solution of Differential Riccati Equations Arising in Optimal Control Problems for Parabolic Partial Differential Equations*, PhD thesis, Escuela Politécnica Nacional, 2007.

[5] P. Benner and H. Mena, *Rosenbrock methods for solving differential Riccati equations*, Max Planck Institute Magdeburg, Preprints, 2011, in preparation.

[6] P. Benner and H. Mena, *Numerical solution of large scale differential Riccati Equations arising in optimal control problems*, Max Planck Institute Magdeburg, Preprints, 2011, in preparation.

[7] N.J. Higham, *Functions of Matrices: Theory and Computation*, SIAM, Philadelphia, USA, 2008.

[8] P. Benner, P. Ezzatti, D. Kressner, E. S. Quintana-Ortí and A. Remón, *A mixed-precision algorithm for the solution of Lyapunov equations on hybrid CPU-GPU platforms*, Parallel Computing, to appear.

 ISBN: 978-84-614-6167-7

[9] G. Golub and C. V. Loan, *Matrix Computations, 3rd Edition*, The Johns Hopkins University Press, Baltimore, 1996.

[10] E. Anderson, Z. Bai, J. Demmel, J. E. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. E. McKenney, S. Ostrouchov and D. Sorensen, *LAPACK Users' Guide*, SIAM 1992

[11] E.S. Quintana-Ortí, G. Quintana-Ortí, X. Sun and R.A. van de Geijn, *A note on parallel matrix inversion*, SIAM J. Sci. Comput., vol. 22, pp. 1762-1771, 2001.

[12] P. Ezzatti, E. S. Quintana-Ortí and A. Remón, *High Performance Matrix Inversion on a Multi-core Platform with Several GPUs*, Proceedings of the 19th Euromicro International Conference on Parallel, Distributed and Network based Processing, pp. 87-93, 2011.

 ISBN: 978-84-614-6167-7