P. Benner P. Ezzatti H. Mena
E. S. Quintana-Ortí A. Remón

Solving Matrix Equations on Multi-core and Many-core Architectures

Max Planck Institute Magdeburg Preprints

MPIMD/11-07 **September 27, 2011**

Impressum:

Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg

Publisher:
Max Planck Institute for Dynamics of Complex Technical Systems

Address:
Max Planck Institute for Dynamics of Complex Technical Systems
Sandtorstr. 1
39106 Magdeburg

www.mpi-magdeburg.mpg.de/preprints

Solving Matrix Equations on Multi-core and Many-core Architectures

P. Benner* P. Ezzatti † H. Mena‡
E. S. Quintana-Ortí A. Remón§

Abstract

We address the numerical solution of Lyapunov, algebraic and differential Riccati equations, via the matrix sign function, on platforms equipped with general-purpose multi-core processors and, optionally, one or more graphics processing units (GPUs). In the paper, we review the solvers for these equations as well as the underlying methods, emphasizing their concurrency and scalability, and providing a few details on their parallel implementation. Our experimental results show that this class of hardware provides sufficient computational power to tackle large-scale problems, which only a few years ago would have required a cluster of computers.

Keywords: Control theory , Lyapunov and Riccati equations , high performance.

*Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany benner@mpi-magdeburg.mpg.de.

†Centro de Cálculo-Instituto de Computación, Univ. de la República, Montevideo, Uruguay, pezzatti@fing.edu.uy

‡Departamento de Matemática, Escuela Politécnica Nacional, Quito, Ecuador, hermann.mena@epn.edu.ec

§Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaime I, Castellón, Spain, {quintana,remon}@icc.uji.es

1 Introduction

Matrix equations are frequently encountered in control theory applications like, e.g., model reduction or linear-quadratic optimal control problems, involving dynamical linear systems modeling a variety of physical phenomena or chemical processes [1]. In general, reliable (i.e., numerically stable) methods for the solution of these equations require a number of floating-point operations (flops) that is cubic in the number of states of the dynamical linear systems, say n. Therefore, the solution of matrix equations with n of $O(1,000)$ and larger asks for the use of high performance architectures, often parallel computers, as well as highly concurrent numerical methods.

In this paper we address the solution of three types of control theory problems, specifically Lyapunov equations and algebraic/differential Riccati equations (AREs/DREs). PLiC (see, e.g., [9, 8]) can be viewed as an extension of some of the methods in SLICOT* for message-passing platforms (e.g., clusters of computers and other kinds of distributed-memory systems) that employs ScaLAPACK [10] for computation and BLACS [11] for communication. Thus, the PLiCMR package offers a tool for the numerical solution of very-large scale problems, provided a proportional amount of hardware resources are employed.

In recent years, we have witnessed a rapid evolution in the number and computational power of processing units (cores) featured by general-purpose CPUs, and an increasing adoption of GPUs as hardware accelerators in scientific computing. A number of works have demonstrated the remarkable speed-up these systems can provide for the solution of dense and sparse linear algebra problems. A few works have presently targeted the numerical solution of matrix equations, which basically can be decomposed into primitive linear algebra problems, using this class of hardware [3, 5, 6, 7]. In this paper we address the rapid solution of Lyapunov equations, AREs and DREs, on multi-core processors as well as GPUs, with the following contributions:

- We show that the matrix sign function provides a crucial building block for the efficient parallel solution of these three types of matrix equations on multi-core CPUs and hybrid CPU-GPU platforms.

- While most previous work focuses on performance, experimentally evaluating matrix equations solvers using single-precision arithmetic [3, 5,

*`http://www.slicot.de/`.

6], double-precision is the convention for linear algebra problems. In this paper we update these data by reporting double-precision results on the latest available version of GPU architecture from NVIDIA: the Fermi processor, which doubles the number of processing cores with respect to the previous generation, and which was mostly employed in those works as well as in [4].

- We follow the design in [5] for DREs, implementing and evaluating algorithms that employ multiple GPUs for the solution of Lyapunov equations and AREs.

- Overall, we provide a clear demonstration that commodity hardware, available in most current desktop systems, offers sufficient computational power to solve large-scale matrix equations, with $n \approx 5,000 - 10,000$.

The rest of the paper is structured as follows. In Section 2 we revisit the matrix sign function, which is the highly parallel building block underlying our Lyapunov equation and ARE solvers described in the first two subsections there. The third subsection then addresses the solution of the DRE using the matrix-sign function-based Lyapunov solver just introduced. Several implementation details for the different solvers/equations are provided next, in Section 3. The main contribution of this paper, namely the experimental evaluation of double-precision implementations of these solvers, using an state-of-the-art CPU-GPU platform, follows in Section 4 and a few concluding remarks close the paper in Section 5.

2 Matrix Sign Function-Based Solvers

The sign function method [17] is an efficient tool to solve Lyapunov, Sylvester and Riccati equations on parallel message-passing computers [9, 8] as well as on hardware accelerators [4]. The convenience of the sign function method is based on two properties: First, it is composed of well-known basic linear algebra operations that exhibit a high degree of concurrency. Moreover, high performance implementations for parallel architectures of these operations are included in linear algebra libraries like BLAS and LAPACK, and their extensions for GPUs (CUBLAS, from NVIDIA) and message-passing platforms (e.g., ScaLAPACK). Second, it is an iterative algorithm which presents a fast convergence rate, asymptotically quadratic.

Several schemes have been proposed in the literature to compute the sign function. Among them, the Newton iteration illustrated in Algorithm `GESINE` below exhibits a remarkable simplicity and efficiency.

Algorithm `GESINE`**:**
$A_0 \leftarrow A$
$k \leftarrow 0$
`repeat`
  $A_{k+1} \leftarrow \left(A_k + A_k^{-1}\right)/2$
  $k \leftarrow k + 1$
`until` $\sqrt{\|A_k - A_{k-1}\|_1} < \tau_s \|A_k\|_1$

The most time consuming operation in Algorithm `GESINE` is the inversion of A_k. Given a square matrix A of order n, this operation renders the cost of the whole algorithm as $\mathcal{O}(n^3)$ flops. The cubic computational cost in the matrix dimension is inherited by all solvers described next.

To avoid stagnation in the iteration, we set $\tau_s = n \cdot \sqrt{\varepsilon}$, where ε stands for the machine precision, and perform one additional iteration step after the stopping criterion is satisfied. Due to the asymptotic quadratic convergence of the Newton iteration, this is usually enough to reach the attainable accuracy.

2.1 Solution of Lyapunov equations

Algorithm `GECLNC`, presented below, illustrates a variant of the sign function method for the solution of a Lyapunov equation of the form

$$AX + XA^T = -BB^T, \tag{1}$$

where $A \in \mathbf{R}_{n\times n}$ is c-stable (i.e., all its eigenvalues have negative real part) and $B \in \mathbf{R}_{n\times m}$ are the coefficient matrices, and $X \in \mathbf{R}_{n\times n}$ is the desired solution.

Algorithm `GECLNC`**:**

$A_0 \leftarrow A$, $\tilde{S}_0 \leftarrow B^T$

$k \leftarrow 0$

`repeat`

Compute the rank-revealing QR (RRQR) decomposition

$$\frac{1}{\sqrt{2c_k}} \left[\tilde{S}_k, \;\; c_k \tilde{S}_k A_k^{-T} \right] = Q_s \begin{bmatrix} U_s \\ 0 \end{bmatrix} \Pi_s$$

$\tilde{S}_{k+1} \leftarrow U_s \Pi_s$

$A_{k+1} \leftarrow \frac{1}{\sqrt{2}} \left(A_k/c_k + c_k A_k^{-1} \right)$

$k \leftarrow k + 1$

`until` $\sqrt{\|A_k - I\|_1} < \tau_l$

On convergence, after $\tilde{k}$ iterations, $\tilde{S} = \frac{1}{\sqrt{2}} \tilde{S}_{\tilde{k}}$, is a full (row-)rank approximation of S, so that $X = S^T S \approx \tilde{S}^T \tilde{S}$.

In practice, the scaling factor c_k is used to accelerate the convergence rate of the algorithm, (i.e., reduce the number of required iterations). In our case, we set

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

We choose $\tau_l = \tau_s$ and perform an extra step after the convergence criterion is satisfied. Note that the number of columns of $\tilde{S}_k$ is doubled at each iteration and, in consequence, the computational and storage costs associated with its update increase with each iteration. This growth can be controlled by computing a RRQR factorization, which introduces a relatively low overhead. This approach reports important gains when the number of iterations needed for convergence is large, or when the number of columns of B is large. The RRQR decomposition can be obtained by means of the traditional QR factorization with column pivoting [14] plus a reliable rank estimator. Note that Q_s is not accumulated as it is not needed in subsequent computations. This reduces the cost of computing the RRQR significantly.

2.2 Solution of algebraic Riccati equations

The sign function method can be also employed to compute the stabilizing solution of an algebraic Riccati equation (ARE) of the form

$$F^T X + XF - XGX + Q \;=\; 0, \tag{2}$$

where F, G, $Q \in \mathbf{R}_{n\times n}$, and the solution $X \in \mathbf{R}_{n\times n}$ satisfies that $F - GX$ is symmetric and c-stable.

The matrix sign function can be also employed to solve eq:ARE. In particular, the solution to the ARE can be obtained from the c-stable invariant subspace of the Hamiltonian matrix [2]:

$$H = \begin{bmatrix} F & G \\ -Q & -F^T \end{bmatrix}, \tag{3}$$

which can be extracted by first obtaining the matrix sign function of H,

$$\mathrm{sign}(H) = Y = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix}, \tag{4}$$

and, then, solving the over-determined system,

$$\begin{bmatrix} Y_{11} \\ Y_{12} + I_n \end{bmatrix} X = \begin{bmatrix} I_n - Y_{21} \\ -Y_{11} \end{bmatrix}. \tag{5}$$

Algorithm `GECRSG` summarizes the above steps to solve the ARE in eq:ARE with this method.

Algorithm `GECRSG`:

$H_0 \leftarrow \begin{bmatrix} F & G \\ -Q & -F^T \end{bmatrix}$

Apply `GESINE` to compute $Y = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21}Q & Y_{22} \end{bmatrix} \leftarrow \mathrm{sign}(H_0)$

Solve $\begin{bmatrix} Y_{11} \\ Y_{12} + I_n \end{bmatrix} X = \begin{bmatrix} I_n - Y_{21} \\ -Y_{11} \end{bmatrix}$ for X

2.3 Solution of differential Riccati equations

The matrix sign function can be also applied in combination with the Rosenbrock method for the solution of an autonomous symmetric differential Riccati equation (DRE) of the form

$$\begin{aligned} \dot{X}(t) &= Q(t) + X(t)A(t) + A(t)^T X(t) - X(t)S(t)X(t) \equiv F(t, X(t)), \\ X(t_0) &= X_0, \end{aligned} \tag{6}$$

where $t \in [t_0, t_f]$, $A(t) \in \mathbb{R}^{n\times n}$, $Q(t) = Q(t)^T \in \mathbb{R}^{n\times n}$, $S(t) = S(t)^T \in \mathbb{R}^{n\times n}$, and $X(t) \in \mathbb{R}^{m\times n}$. Here, we assume that the coefficient matrices are piecewise continuous locally bounded matrix-valued functions which ensures the existence and uniqueness of the solution to (6); see, e.g., [1, Thm. 4.1.6].

The application of the Rosenbrock method of order one to an autonomous symmetric DRE of the form (6) yields:

$$\tilde{A}_k^T X_{k+1} + X_{k+1}\tilde{A}_k = -Q - X_k S X_k - \frac{1}{h}X_k, \tag{7}$$

where $X_k \approx X(t_k)$ and $\tilde{A}_k = A - SX_k - \frac{1}{2h}I$; see [15, 16] for details. In addition we assume, $Q = C^T C$, $C \in \mathbb{R}^{p\times n}$, $S = BB^T$, $B \in \mathbb{R}^{n\times m}$, $X_k = Z_k Z_k^T$, $Z_k \in \mathbb{R}^{n\times z_k}$, 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 (7) results in

$$\tilde{A}_k^T X_{k+1} + X_{k+1}\tilde{A}_k = -N_k N_k^T, \tag{8}$$

where $\tilde{A}_k = A - B(Z_k(Z_k^T B))^T - \frac{1}{2h}I$. The procedure that is obtained from this elaboration is offered in Algorithm `ROS1`. Observing that $\operatorname{rank}(N_k) \leq p+m+z_k \ll n$, we can use the sign function method to solve (8), as illustrated in Algorithm `GECLNC` (see subsection 3.1).

Algorithm `ROS1`:

$t_0 \leftarrow a$

`for` k=0 to $\lceil \frac{b-a}{h} \rceil$
  $\tilde{A}_k = A - B(Z_k(Z_k^T B))^T - \frac{1}{2h}I$
  $N_k = [\, C^T \;\; Z_k(Z_k^T B) \;\; \sqrt{h^{-1}} Z_k \,]$
  Apply `GECLNC` to obtain Z_{k+1}, a low-rank approximation to
    the solution of $\tilde{A}_k^T X_{k+1} + X_{k+1}\tilde{A}_k = -N_k N_k^T$
  $t_{k+1} = t_k + h$
`end for`

Note that, since we obtain the low rank factor of the solution of (8), the cost associated to the updates of both $\tilde{A}_k$ and $\tilde{N}_k$ is drastically decreased, and thus, the most time consuming operation in Algorithm `ROS1` is the solution of (8), i.e., the execution of Algorithm `GECLNC`.

3 High Performance Implementations

In this section we briefly introduce several high performance implementations for the solution of the different matrix equations in study. All the developed routines are based on the sign function method. Three different implementations are proposed for each matrix equation: one based on the use of multi-threaded BLAS for general-purpose multi-core processors, a hybrid CPU-GPU one, and a hybrid CPU-multiGPU implementation.

The multi-core implementations basically employ high performance routines from BLAS and LAPACK to leverage the hardware parallelism of these architectures. In addition, OpenMP directives are utilized to accelerate the computation of other minor operations.

Hybrid CPU-GPU implementations exploit the massively parallel architecture of the GPU to reduce the time-to-response of large and highly parallel computations (e.g., large matrix-matrix products), while operations that feature a reduced computational cost, and/or that present a fine grain parallelism, are executed on the CPU. In general, our hybrid CPU-GPU implementations aim at reducing the overhead introduced by CPU-GPU data transfers. Thus, data are transferred only when this is amortized with a reduction of the global execution time. In practice, the most time consuming operation present in the solution of Lyapunov and Riccati equations via the sign function method is the matrix inversion. This operation is dramatically accelerated by off-loading most of the computations to the GPU [4]. To enhance the performance, the matrix inversion is computed via the Gauss-Jordan elimination method, which is more suitable for the GPU architecture than the traditional approach based on the LU factorization; see [13].

Hybrid CPU-GPU implementations offer remarkable performance, but the dimension of the problems that can be tackled with them is limited by the size of the GPU memory (typically, 3 Gbytes). Hybrid implementations that combine a CPU with multiple GPUs partially overcome this problem since, as the number of GPUs grows, the aggregated size of the memory is also increased. The use of several GPUs also increments the computational power of the platform, and thus, can potentially reduce the execution time. These implementations are based on a multi-GPU matrix inversion kernel, where the CPU and the GPUs present in the platform cooperate in the computation of the matrix inverse; see [12].

3.1 Lyapunov solvers

Three implementations of Algorithm `GECLNC` are proposed. In the multi-core routine, GECLNC_MC, all the computations are performed on the CPU. High performance linear algebra kernels from BLAS and LAPACK are employed to execute most of the computations (QR factorization, matrix inversion, matrix-matrix products and the computation of the scaling factor), while OpenMP directives are employed to parallelize other minor computations, like matrix addition, matrix scaling and the computation of the loop guard.

In routine GECLNC_GPU, each operation is performed on the most convenient device. The CPU computes the RRQR decomposition and the update of matrix $\tilde{S}_{k+1}$, while CPU and GPU cooperate in the computation of A_{k+1}. In particular, the GPU is employed to accelerate the computation of A_k^{-1}.

Finally, GECLNC_MGPU employs several GPUs to accelerate the computation of the matrix inverse. This implementation permits the solution of larger problems and potentially reduces their execution time.

3.2 Algebraic Riccati equations

Three implementations are presented for the solution of AREs as well: a multi-core, a hybrid CPU-GPU, and a multi-GPU variant. GECRSG_MC refers to the multi-core implementation that employs multi-threaded kernels from BLAS. In addition, OpenMP directives are used to parallelize the construction of matrix H, the matrix addition and scaling required for the update of H_{k+1}, and other minor computations. Multi-thread routines from BLAS and LAPACK are used to compute H_k^{-1} as well as to solve the overdetermined system in the last stage of the method.

During the development of the GECRSG_GPU variant, some experiments [7] demonstrated that, in practice, only the matrix inversion can be accelerated using the GPU as off-loading more computations to the GPU reported heavy data transfer overheads. The same inversion kernel employed in GECLNC_GPU can be leveraged here to compute H_k^{-1}. Note that GECRSG_GPU can only be used to solve problems where H fits in the GPU memory, which in general is much smaller than that of the main system, and that the dimension of H doubles that of A.

Finally, the multi-GPU implementation, GECRSG_MGPU, accelerates the matrix inversion kernel and partially alleviates the constraint of the mem-

ory dimension introduced by GECRSG_GPU.

3.3 Differential Riccati equations

As in the previous subsections, we have developed a CPU-based and two GPU-based implementations. The computational cost of Algorithm `ROS1` is clearly reduced if Z_{k+1} is a low-rank factor. In this case, the total cost of the algorithm is diminished to that required by the solver of the Lyapunov equations.

The multi-core implementation, ROS1_MC, employs BLAS and LAPACK kernels to update $\tilde{A}_k$ and $\tilde{N}_k$, and routine GECLNC_MC to obtain the solution of (8). Once more, OpenMP directives have been employed to parallelize some minor computations like matrix additions and norm computations.

The hybrid CPU-GPU implementation, ROS1_GPU, executes each operation on the most convenient device. In particular, the update of $\tilde{A}_k$ and $\tilde{N}_k$ require several matrix-matrix products, matrix additions and scalings that are computed on the CPU. Despite matrix-matrix products of large matrices are suitable for the GPU architecture, the dimensions of B and Z_k^T are usually too small to amortize the cost of data transfers. Routine GECLNC_GPU is employed to obtain the solution of the Lyapunov equation.

The last implementation, ROS1_MGPU, employs several GPUs to accelerate the solution of Lyapunov equations using the GECLNC_MGPU variant.

4 Experimental Results

We evaluate the performance of the implementations using two problems from the *Oberwolfach Model Reduction Benchmark Collection*[†]: STEEL (we employ two instances of this problem, STEEL_S and STEEL_L. For both cases, $m = 7$ and $p = 6$. The order of the system is n=1,357 for the STEEL_S instance and 5,177 for the STEEL_L instance) and FLOW_METER (the dimensions of this problem are $n = 9,669$, $m = 1$, $p = 5$.).

Experiments are performed on a computer equipped with two INTEL Xeon QuadCore E5440 processors at 2.83 GHz, with 16 GB of RAM, connected to an NVIDIA Tesla S2050 (consisting of four NVIDIA M2050 GPUs) via a PCI-e bus.

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

	GECLNC_MC	GECLNC_GPU	GECLNC_MGPU
STEEL$_S$	0.51	0.54	0.73
STEEL$_L$	24.05	7.52	6.79
FLOW_METER	239.35	61.70	40.53

Table 1: Execution time (in secs.) for the solution of Lyapunov equations.

A multi-thread version of the INTEL MKL library (version 11.0) provides the necessary LAPACK and BLAS kernels for the CPU, and NVIDIA CUBLAS (version 3.2) for the GPU computations. Experiments are performed in double precision arithmetic.

Table 1 shows the results obtained with the Lyapunov equation solvers using the three implementations. GECLNC_MC is the fastest option for the small problem, STEEL$_S$, while the use of the GPU reports important gains for the two largest problems. GECLNC_GPU is approximately 3.5× faster than the CPU variant for the STEEL$_L$ and the FLOW_METER benchmarks. The multi-GPU routine, GECLNC_MGPU, obtains the best results for the larger problems, being 50% faster than GECLNC_CPU for the largest evaluated problem.

Table 2 summarizes the results for the solution of algebraic Riccati equations. Note that the codes to build matrix H_0 (column 2) and solve the over-determined system (column 3) are similar for the three implementations evaluated. Columns 4–9 report the time dedicated to compute the sign function and the total execution time for each variant. GECRSG_GPU obtains the best result for the STEEL$_S$ problem, and clearly outperforms the multi-core routine. GECRSG_MGPU is the faster variant for the other two problems. Specifically, it is 8× and 1.5× faster for the STEEL$_L$ problem than the GECRSG_MGPU and the GECRSG_GPU variants, respectively. Results for the largest problem, FLOW_METER, illustrate the efficiency of the multi-GPU routine. Compared with the CPU variant, it reports an accelerating factor larger than 13×, while GECRSG_GPU cannot solve this problem due to the limited size of the memory.

Finally, Table 3 shows the execution time obtained for the solution of differential Riccati equations using the sign function-based Lyapunov solver. Two values are reported for each implementation, the time to compute the

			GECRSG_MC		GECRSG_GPU		GECRSG_MGPU	
	H_0	System	Fsign	Total	Fsign	Total	Fsign	Total
STEEL$_S$	0.10	0.59	28.60	29.29	12.07	12.76	15.48	16.17
STEEL$_L$	1.36	24.75	1,467.69	1,493.80	267.56	293.67	162.75	188.86
FLOW_METER	4.76	151.25	9,263.71	9,419.72			533.25	689.26

Table 2: Execution time (in secs.) for the solution of algebraic Riccati equations.

	ROS1_MC		ROS1_GPU		ROS1_MGPU	
	Fsign	Total	Fsign	Total	Fsign	Total
STEEL$_S$	5.80	5.97	5.86	6.02	8.15	8.31
STEEL$_L$	262.36	264.36	87.90	90.24	79.46	81.80
FLOW_METER	2,664.78	2,674.54	626.99	637.08	391.81	401.64

Table 3: Execution time (in secs.) for the solution of differential Riccati equations.

sign function (Fsign) and the total execution time. Most of the time, approximately a 97%, is dedicated to the sign function method (i.e., the Lyapunov solver). The ROS1_MC implementation attains the best execution time for the small problem, STEEL$_S$, while it is clearly outperformed for larger problems. The multi-GPU variant, ROS1_MGPU, is the best option for the solution of large problems. In comparison with the multi-core implementation, it is 3× and 6× faster for the STEEL$_L$ and the FLOW_METER problems, respectively. The single GPU variant also obtains remarkable performances in all the experiments, but ROS1_MGPU is 50% faster for the solution of the FLOW_METER benchmark.

5 Conclusions

We have addressed the solution of three types of matrix equations that arise in control theory applications: Lyapunov, algebraic Riccati and differential Riccati equations. Three implementations are presented for each solver: one that runs on a multi-core CPU; and two hybrid implementations, one for a system equipped with a multi-core CPU and a GPU, and one for a platform composed by a multi-core CPU connected to several GPUs. All the implementations heavily make use of high performance kernels from linear algebra libraries like MKL and CUBLAS, and OpenMP directives.

Numerical results employing three benchmarks, extracted from the *Oberwolfach Model Reduction Benchmark Collection*, show the efficiency attained by the proposed routines. The hybrid implementations, based on the use of a single GPU, report a remarkable performance, being more than 4× faster than their corresponding multi-core counterparts for large problems. However, their applicability is limited by the size of the GPU memory. This limitation is partially overcome in the multi-GPU implementations which, in addition, are 50% faster in the solution of large dimension problems for the three matrix equations evaluated.

Acknowledgements The researchers at Universidad Jaume I were supported by project CICYT TIN2008-06570-C04-01 and FEDER.

References

[1] H. Abou-Kandil, G. Freiling, V. Ionescu, and G. Jank. *Matrix Riccati equations in control and systems theory*. Basel, Switzerland, 2003.

[2] P. Benner, R. Byers, E. S. Quintana-Ortí, and G. Quintana-Ortí. Solving algebraic Riccati equations on parallel computers using Newton's method with exact line search. *Parallel Computing*, 26(10):1345–1368, 2000.

[3] P. Benner, P. Ezzatti, D. Kressner, E. S. Quintana-Ortí, and A. Remón. Accelerating model reduction of larger linear systems with graphics processors. In *PARA'10 Applied Computing: State-of-the-Art in Scientific Computing*, Lecture Notes in Computer Science. Springer-Verlag, 2011. Submitted.

[4] P. Benner, P. Ezzatti, Daniel 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*, 37:439–450, 2011.

[5] P. Benner, P. Ezzatti, H. Mena, E. S. Quintana-Ortí, and A. Remón. Solving differential Riccati equations on multi-GPU platforms. In *10th International Conference on Computational and Mathematical Methods in Science and Engineering – CMMSE 2011*, pages 178–188, 2011.

[6] P. Benner, P. Ezzatti, E. S. Quintana-Ortí, and A. Remón. Using hybrid CPU-GPU platforms to accelerate the computation of the matrix sign function. In H.X. Lin, M. Alexander, M. Forsell, A. Knüpfer, R. Prodan, L. Sousa, and A. Streit, editors, *7th Int. Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, Lecture Notes in Computer Science, Vol. 6043, pages 132–139. Springer-Verlag, 2009.

[7] P. Benner, P. Ezzatti, E. S. Quintana-Ortí, and A. Remón. Accelerating BST methods for model reduction with graphics processors. In *9th Int. Conference on Parallel Processing and Applied Mathematics*, 2011. Submitted.

[8] P. Benner, E. S. Quintana, and G. Quintana. Solving linear-quadratic optimal control problems on parallel computers. *Optimization Methods & Software*, 23(6):879–909, 2008.

[9] P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. State-space truncation methods for parallel model reduction of large-scale systems. 29:1701–1722, 2003.

[10] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, 1997.

[11] Jack J. Dongarra, Robert A. van de Geijn, and R. Clint Whaley. Two dimensional basic linear algebra communication subprograms. In *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, March 1993.

[12] P. Ezzatti, E. S. Quintana-Ortí, and A. Remón. High performance matrix inversion on a multi-core platform with several GPUs. In *19th Int. Euromicro Conf. on Parallel, Distributed and Network-Based Processing*, pages 87–93, 2011.

[13] P. Ezzatti, E. S. Quintana-Ortí, and A. Remón. Using graphics processors to accelerate the computation of the matrix inverse. *The Journal of Supercomputing*, 2011. To appear.

[14] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 2nd edition, 1989.

[15] H. Mena. *Numerical Methods for Large-Scale DIfferential Riccati Equations with Applications in Optimal of Partial Differential Equations*. PhD thesis, Escuela Politécnica Nacional, Quito, Ecuador, 2007.

[16] H. Mena and P. Benner. Numerical solution of large scale differential Riccati Equations arising in optimal control problems. Technical report, Max Planck Institute Magdeburg. In preparation.

[17] J.D. Roberts. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. 32:677–687, 1980. (Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department, 1971).