Solving Algebraic Riccati Equations on Hybrid CPU-GPU Platforms

Pablo Ezzatti[1], Enrique S. Quintana-Ortí[2], and Alfredo Remón[2]

[1] Centro de Cálculo–Instituto de Computación, Universidad de la República, 11.300–Montevideo, Uruguay, pezzatti@fing.edu.uy
[2] Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain, {quintana,remon}@icc.uji.es

Abstract. The solution of Algebraic Riccati Equations is required in many linear optimal and robust control methods such as LQR, LQG, Kalman filter, and in model order reduction techniques like the balanced stochastic truncation method. Numerically reliable algorithms for these applications rely on the sign function method, and require $\mathcal{O}(8n^3)$ floating-point arithmetic operations, with n in the range of $10^3 - 10^5$ for many practical applications. In this paper we investigate the use of graphics processors (GPUs) to accelerate the solution of Algebraic Riccati Equations by off-loading the computationally intensive kernels to this device. Experiments on a hybrid platform compose by state-of-the-art general-purpose multi-core processors and a GPU illustrate the potential of this approach.

1 Introduction

In this paper we target the solution of large-scale Algebraic Riccati Equations (AREs) of the form

$$F^T X + XF - XGX + Q \;=\; 0$$

where F, X, G and $Q \in \mathbb{R}^{n \times n}$. We consider a system to be large-scale if $n \sim \mathcal{O}(1,000) - \mathcal{O}(100,000)$.

This kind of equations appears in many linear optimal and robust control methods like the infinite horizon time-invariant Linear-Quadratic Regulator problem (LQR), the infinite horizon time-invariant Linear-Quadratic-Gaussian control problem (LQG), the Kalman filter, and in model order reduction techniques as the Balanced Stochastic Truncation (BST) method [1].

The numerical method for the solution of AREs considered in this paper is based on the sign function method computed via the Newton iteration method. This technique requires the computation of dense linear algebra operations. Although there exist several other approaches to solve AREs, in general those require a larger computational cost.

Recent work on the implementation of BLAS kernels and the major factorization routines for the solution of linear systems [8, 5, 2, 6] has demonstrated the potential of graphics processors (GPUs) to yield high performance on the computation of dense linear algebra operations which can be cast in terms of

matrix-matrix products. In [4] we studied the solution of standard Lyapunov equations on GPUs using high performance computing techniques. Here, we further extend this work optimizing the computation of the different stages in the sign function method for the solution of AREs, and other minor computations required (build matrix and solve the overdetermined system).

The target architecture is a hybrid platform consisting of a general-purpose multi-core processor and a GPU. We exploit these two resources by designing a hybrid numerical algorithm to solve AREs that performs fine-grained computations on the CPU while off-loading the most computationally intensive operations to the GPU.

The rest of the paper is structured as follows. In Section 2 we briefly review the sign function method for AREs and describe an efficient approach to compute it on a hybrid CPU-GPU platform. In Section 3 we present some experimental results that illustrate the efficiency attained by the numerical algorithms on a platform consisting of two Intel QuadCore processors connected to a NVIDIA Tesla C2050 GPU via a PCI-e bus. Finally, in Section 4 we provide a few concluding remarks and future work.

2 The Riccati sign function method

The stabilizing solution of an ARE (i.e. one that $F + GX$ is symmetric and stable matrix) of the form

$$F^T X + XF - XGX + Q \;=\; 0 \tag{1}$$

can be defined by the invariant subspaces of the pencil $H - \lambda I_{2n}$, where n is the dimension of matrix H, I_{2n} is the identity matrix of dimension $2n$ and H is the Hamiltonian matrix defined as

$$H = \begin{bmatrix} F & G \\ -Q & -F^T \end{bmatrix}. \tag{2}$$

Additionally, it can be shown that the matrix generated by a base of the H invariant spaces is the solution of the associated ARE in Eq. (1) (see [3]). This solution can be obtained computing the sign function (sign) of H,

$$\mathrm{sign}(H) = Y = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}, \tag{3}$$

and then, solving the overdetermined system (e.g., applying the least squares method),

$$\begin{bmatrix} Y_{11} \\ Y_{12} + I_n \end{bmatrix} X = \begin{bmatrix} I_n - Y_{10} \\ -Y_{00} \end{bmatrix}. \tag{4}$$

Algorithm `GECRSG` summarizes the steps to be computed to solve an ARE with the described method.

Algorithm `GECRSG`**:**

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

`for` $k = 0, 1, 2, \ldots$ *until convergence*

$$H_{k+1} \leftarrow \tfrac{1}{2c_k}\left(H_k + c_k^2 H_k^{-1}\right)$$

$$\text{solve} \begin{bmatrix} Y_{11} \\ Y_{12} + I_n \end{bmatrix} X = \begin{bmatrix} I_n - Y_{10} \\ -Y_{00} \end{bmatrix}$$

The Newton iteration for the sign function usually presents a fast convergence rate, which is ultimately quadratic. Additionally, initial convergence can be accelerated using several techniques. In our approach, we employ a scaling defined by the parameter

$$c_k = \sqrt{\|H_k\|_{\text{fro}} / \|H_k^{-1}\|_{\text{fro}}}.$$

2.1 Hybrid implementation of the Riccati solver

In this section we describe an efficient implementation for the solution of AREs. This implementation is specially designed for hybrid computational platforms composed by multi-core CPU connected to a GPU.

There are two kinds of computational units on the target platform, and the objective of the hybrid implementation proposed is to reduce the computational time of the AREs solver executing each operation on the most convenient architecture. Particulary, the largest operations are executed on the GPU while the fine-grain operations are executed on the CPU.

The procedure to solve AREs can be divided into three steps:

- First, is necessary to build matrix H performing some matrix-matrix multiplications. Due to the relatively small dimensions of the matrices involved, these operations require a moderate cost. For this reason, and with the aim to reduce data transfers overheads, those operations are performed on the CPU.
- Second, the sign function for the extended matrix (Eq. 2) is computed. The proposal is based on an efficient hybrid matrix inversion kernel.
  The hybrid algorithm proceeds as follows. At the beginning of each iteration, the CPU transfers matrix H_k to the GPU. Then, the CPU and the GPU cooperate in the inversion of matrix H_k. Finally, the inverse matrix is transferred to the CPU and the iteration finishes. The implementation includes processing by blocks, hybrid and concurrent computing (CPU + GPU), look-ahead techniques [7], and padding. The rest of the operations are performed on the CPU since they require a minor computational effort and can be efficiently executed using multi-thread level parallelism, i.e., employing multi-thread implementations of BLAS and LAPACK to compute linear algebra operations and using OpenMP in other cases.

– Finally, the overdetermined system is solved. To do so, a multi-thread version of routine `DGEQP3`, included in LAPACK, is employed. Other minor operations are also executed on the CPU and parallelized using OpenMP.

3 Numerical Experiments

In this section we evaluate the parallel performance of the AREs solver presented in the previous section. The target platform consists of two INTEL Xeon Quad-Core E5410 processors at 2.33GHz, connected to an NVIDIA Tesla C2050 via a PCI-e bus (see Table 1 for more details). We employed a multi-threaded implementation of BLAS in MKL (version 10.2) for the general-purpose processor and NVIDIA CUBLAS (version 3.2) for the GPU. We set `OMP_NUM_THREADS=8` so that one thread is employed per core in the parallel execution of the MKL routines in the Intel Xeon QuadCore processors, and `OMP_NUM_THREADS=1` on the evaluation of the sequential version.

Processors	#cores	Freq. (GHz)	L2 (MB)	Memory (GB)
INTEL Xeon	8	2.3	12	8
NVIDIA Fermi	448	1.3	–	3

Table 1. Hardware employed in the experiments.

We compare three different implementations: a sequential implementation (ARES_SCPU) that is executed on a single CPU core (used as the reference implementation), a parallel multi-thread implementation (ARES_MTCPU) that exploits all the cores on the CPU, and a hybrid CPU-GPU implementation (ARES_HYB) that executes operations concurrently on the GPU and on the CPU cores.

We employ double precision arithmetic on the solution of two instances from the `STEEL_I` model reduction problem, extracted from the Oberwolfach benchmark collection (University of Freiburg)[3].

The `STEEL_I` model arises in a manufacturing process for steel profiles. The goal in this problem 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 in the example in this benchmark. The problem dimensions depends of the discretization mesh, the two versions employed in this work are $\texttt{STEEL\_I}_{1357}$ with $n = 1,357$ and $\texttt{STEEL\_I}_{5177}$ with $n = 5,177$.

Table 2 summarizes the results obtained with all the implementations evaluated. The execution time dedicated to build matrix H is shown in column 2;

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

Implementation	H init.	sign(H)	System solver	Total time(s)
STEEL_I$_{1357}$				
ARES_SCPU	0.10	118.15	3.23	121.48
ARES_MTCPU	0.05	22.34	0.57	22.92
ARES_HYB	0.05	10.93	0.57	11.55
STEEL_I$_{5177}$				
ARES_SCPU	1.52	6404.65	325.34	6730.61
ARES_MTCPU	0.86	1127.87	25.05	1153.78
ARES_HYB	0.78	292.93	24.92	318.63

Table 2. Execution time (in seconds) of the AREs solvers for the STEEL_I benchmark.

column 3 shows the time required compute sign(H); column 4 displays the time spent on the solution of the overdeterminated system; and column 5 shows the accumulated time. All the times given in Table 2 include the costs to perform all the necessary CPU-GPU data transfers.

A careful study of the sign function procedure, demonstrates that the computational effort is focused on the computation of matrix inverses. This operation is accelerated at the ARE_MTCPU implementation using multi-thread codes. The ARE_HYB variant goes a step forward on the optimization of the matrix inverse procedure using the Gauss-Jordan elimination method, which is more suitable for its execution on parallel architectures, and off-loading part of the computations to the GPU.

Times reported for the STEEL_I$_{1357}$ instance show a great benefit from the usage of the multi-core (ARE_MTCPU) and the hybrid (ARE_HYB) implementation, which are 5 and 10 times faster than the sequential implementation respectively. But from the results obtained for STEEL_I$_{5177}$ we can conclude that these differences are increased for larger problems. In this case, ARE_MTCPU is nearly 6 times faster than ARE_CPU, while ARE_HYB is more than 21 times faster. This is due to the fact that larger problems presents a larger inherent parallelism, and therefore, they are more suitable for the massively parallel architecture present at the GPU.

4 Concluding Remarks

We have presented two high performance parallel implementations for the solution of AREs. They differ on the target platform, variant ARE_MTCPU is designed for its execution on a multi-core CPU, while ARE_HYB is optimized for its execution on a hybrid platform composed by a CPU and a GPU. ARE_HYB demonstrated to be a high performance AREs solver which exploits the capa-

bilities of both architectures, the multi-core CPU and the many-core GPU. Two levels of parallelism are employed at this implementation: at the inner level, multi-thread computational kernels included at the BLAS library (MKL and CUBLAS) are employed to compute the most time-consuming linear algebra operations; at the outer level, operations proceed concurrently in both architectures, overlapping computations on the CPU and on the GPU.

Some experimental results show that large-scale AREs can be tackle with this kind of platforms in a reasonable computational time.

The promising results obtained encourage us to continue improving the developed implementations employing new high performance computing techniques and architectures. The future and on-going work includes:

- Exploit the use of multiple GPUs to further reduce the computational time and increase the dimension of the affordable problems.
- Evaluate the use of mixed precision techniques that allow to perform most of the computations in single precision arithmetic (note that the throughput on both architectures is larger for single precision arithmetic).

Acknowledgements

The authors would like to thank Jimena Ferreira for his valuable suggestions that improved this work.

References

1. A.C. Antoulas. *Approximation of Large-Scale Dynamical Systems*. SIAM Publications, Philadelphia, PA, 2005.
2. S. Barrachina, M. Castillo, F. D. Igual, R. Mayo, E. S. Quintana-Ortí, and G. Quintana-Ortí. Exploiting the capabilities of modern GPUs for dense matrix computations. *Concurrency and Computation: Practice and Experience*, 21:2457–2477, 2009.
3. 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 Comput.*, 26(10):1345–1368, 2000.
4. 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, *Euro-Par 2009, Parallel Processing - Workshops*, number 6043 in Lecture Notes in Computer Science, pages 132–139. Springer-Verlag, 2009.
5. P. Bientinesi, F. D. Igual, D. Kressner, and E. S. Quintana-Ortí. Reduction to condensed forms for symmetric eigenvalue problems on multi-core architectures. In *Proceedings of the 8th International Conference on Parallel Processing and Applied Mathematics – PPAM'09*, Lecture Notes in Computer Science. Springer. To appear.
6. H. Ltaif, S. Tomov, R. Nath, P. Du, and J. Dongarra. A scalable high performance cholesky factorization for multicore with GPU accelerators. Lapack working note 223, University of Tennessee, 2009.

7. A. Strazdins. A comparison of lookahead and algorithmic blocking techniques for parallel matrix factorization. TR-CS-98-07 07, The Australian National University, 1998.
8. Vasily Volkov and James Demmel. LU, QR and Cholesky factorizations using vector capabilities of GPUs. Technical Report UCB/EECS-2008-49, EECS Department, University of California, Berkeley, May 2008.