MAX-PLANCK-GESELLSCHAFT

Max Planck Institute Magdeburg
Preprints

Peter Benner Ernesto Dufrechou Pablo Ezzatti
Enrique S. Quintana-Ortí Alfredo Remón

Extending Lyapack for the Solution of Band Lyapunov Equations on Hybrid CPU-GPU Platforms

MAX–PLANCK–INSTITUT
FÜR DYNAMIK KOMPLEXER
TECHNISCHER SYSTEME
MAGDEBURG

MPIMD/14-13 July 24, 2014

Abstract

The solution of large-scale Lyapunov equations is an important tool for the solution of several engineering problems arising in optimal control and model order reduction. In this work we investigate the case when the coefficient matrix of the equations presents a band structure. Exploiting the structure of this matrix we can achive relevant reductions in the memory requirements and the number of floating-point operations. Additionally, the new solver efficiently leverages the parallelism of CPU-GPU platforms. Furthermore, it is integrated in the LYAPACK library to facilitate its use. The new codes are evaluated on the solution of several benchmarks, exposing significant runtime reductions with respect to the original CPU version in LYAPACK.

Contents

Peter Benner, Alfredo Remón
Computational Methods in Systems and Control Theory, Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg Germany,
{benner,remon}@mpi-magdeburg.mpg.de
Ernesto Dufrechou, Pablo Ezzatti
Instituto de Computación, Univ. de la República, Montevideo, Uruguay,
{edufrechou,pezzatti}@fing.edu.uy
Enrique S. Quintana-Ortí
Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaime I, Castellón, Spain
quintana@icc.uji.es

Imprint:

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

http://www.mpi-magdeburg.mpg.de/preprints/

1 Introduction

Lyapunov (matrix) equations appear in a number of engineering and scientific problems as, for example, in model order reduction and many areas of control theory; see [1, 8, 14]. The solution of Lyapunov equations has been widely studied and several efficient and numerically reliable methods can be found in the literature. In many applications, the Lyapunov equations are of large dimension, and, in consequence, their solution involves a large amount of data and a vast computational cost. In response, the use of High Performance Computing (HPC) techniques and architectures is mandatory. To illustrate this, the solver provided by LYAPACK makes an intensive use of tuned HPC kernels in BLAS and LAPACK.

In a set of applications, the coefficient matrix of in the Lyapunov equation exhibits an specific structure: symmetry, banded, etc. In such cases, the Lyapunov solver can be specialized to exploit the benefits of the related structure to reduce either, the memory requirements, the computational cost, or both.

In recent years, Graphics Processing Units (GPUs) have shown remarkable performance in the computation of large-scale matrix operations, and particularly, in the solution of matrix equations; see [4, 5] among others. In addition to its performance, GPUs present other interesting properties, such as a low Watt-per-floating-point arithmetic operation ratio and an afforable price.

In this paper we address the solution of large Lyapunov equations with a banded coefficient matrix. Our approach exploits the structure of the matrix and the ample hardware parallelism of the GPUs to obtain a Lyapunov solver that can be applied to the solution of large-scale problems. Additionally, we have integrated the proposal in the LYAPACK library [13].

The rest of the paper is organized as follows. In Section 2 we introduce the Lyapunov equations and LYAPACK toolbox. The proposed solver is presented in Section 3 and evaluated in Section 4. Finally some concluding remarks are outlined in Section 5.

2 Solution of Lyapunov equations

Consider the Lyapunov matrix equation with factored right hand side:

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

where $A \in \mathbb{R}^{n\times n}$ is the coefficient matrix, $B \in \mathbb{R}^{n\times m}$, and $X \in \mathbb{R}^{n\times n}$ is the sought-after solution. Such matrix equations play a key role in a few control theory applications. For example, the solution of one or more Lyapunov equations are required by SVD-based methods for balanced truncation model reduction [1, 6], Newton's methods for the algebraic Riccati equation associated with linear-quadratic optimal control problems [2], stabilization methods and stability tests for linear dynamical systems as well as the computation of the H_2-norm of stable linear control systems [8, 10]. Complex control systems or discretizations of partial differential equations easily lead to matrices of order $n = 10^4$ to 10^5 or even larger.

In this work we focus in the solution of (1) when A is a band matrix, meaning that all the nonzero entries of A reside in a narrow band, including the main diagonal and a set of consecutive super- and sub-diagonals.

We investigate the solution of Lyapunov equations via the LR-ADI (low-rank Alternating Direction Implicit iteration) method [12], which takes advantage of the frequently encountered low-rank property of B [1] to compute a low-rank approximation to a full-rank factor of X. This approach reduces notoriously the memory requirements and the number of arithmetic operations needed, facilitating the solution of large-scale problems.

Specifically, given an "l–cyclic" set of complex shifts $\{p_1, p_2, \ldots\}$, with $p_k = p_{k+l}$, $p_k = \alpha_k + \beta_k \jmath$, and $\jmath = \sqrt{-1}$, the cyclic LR-ADI for the Lyapunov equation can be formulated as follows:

$$\begin{array}{rcl} V_0 & := & (A + p_1 I_n)^{-1} B, \\ V_{k+1} & := & V_k - \delta_k (A + p_{k+1} I_n)^{-1} V_k, \end{array} \qquad \begin{array}{rcl} S_0 & := & \sqrt{-2\ \alpha_1}\ V_0, \\ S_{k+1} & := & [S_k\ ,\ \gamma_k V_{k+1}], \end{array} \tag{2}$$

where $\gamma_k := \sqrt{\alpha_{k+1}/\alpha_k}$, $\delta_k := p_{k+1} + \overline{p_k}$, $\overline{p_k}$ stands for the conjugate of p_k, and I_n denotes the identity matrix of order n. Upon convergence, after $\bar{k}$ iterations, a low-rank matrix $S_{\bar{k}} \in \mathbb{R}^{n \times \bar{k}m}$ is obtained such that $X \approx S_{\bar{k}} S_{\bar{k}}^T$.

The convergence of the LR-ADI method can be accelerated by a careful selection of the shifts $\{p_1, p_2, \ldots\}$. In practice, the computation of these parameters involves an Arnoldi iteration with the coefficient matrix A and its inverse. For further details on the convergence of the LR-ADI iteration and the properties of heuristic vs. optimal procedures to select the shift parameters, see [12, 15].

From the computational point of view, the cost of the algorithm lies in two basic linear algebra operations: the matrix-matrix products to compute the shifts, and the solution of the linear systems in Eq. (2). Tuned kernels that exploit the band structure of A report relevant savings in both operations, and hence, in the Lyapunov solver.

2.1 The LYAPACK TOOLBOX

LYAPACK [13] is a MATLAB toolbox which offers a variety of numerical routines for the solution of Lyapunov and Riccati equations, model order reduction, and optimal control. The toolbox is appropriate for large and sparse problems and/or structured dynamical systems. Two main properties of LYAPACK are its user-friendly interface and a flexible modular architecture which facilitates its extension and customization.

The Lyapunov solver in LYAPACK is the cornerstone for most of the problems addressed by the toolbox. The solver is an efficient implementation of the LR-ADI method. The principal routines in LYAPACK perform the most expensive computations in terms of three types of basic matrix operations (BMOs): $Z := AY$, $Z := A^{-1}Y$, and $Z := (A + pI_n)^{-1}Y$. In all cases, one of the inputs of the BMO is the coefficient matrix of the Lyapunov equation, A, while Y is a matrix with a small number of columns and p is a scalar. LYAPACK's strategy to accommodate coefficient matrices with different nonzero patterns (e.g., sparse, tridiagonal, band,...) is a *reverse communication interface* also adopted, e.g., by ARPACK and PETSc. Thus, it is the user's responsibility to supply a tuned kernel to compute each BMO (hereafter, referred to as *user-supplied functions* or USFs) that efficiently leverages the nonzero pattern of the problem on a specific target architecture.

In summary, the use of USFs yields a “soft” object-oriented polymorphism. Data from the matrix A is stored into hidden global variables, and the purpose of the USFs is to manipulate these data structures (e.g., create and release the global data) and compute the corresponding BMOs. In consequence, the user has the opportunity to adapt the solver to the particular features of a given problem and the hardware architecture, providing tuned USFs, while the numerical methods and properties in LYAPACK are preserved.

2.2 CPU-GPU version of LYAPACK

In [9] we provided a general framework to use CUDA kernels from LYAPACK with minimal changes to the library. Precisely, the design philosophy underlying this toolbox gives us the opportunity to leverage the acceleration of a GPU without modifying the contents of the library, by implementing the appropriate CUDA-enabled USFs. Concretely, this solution leverages the MATLAB Mex API that provides interoperability between MATLAB and other programming languages such as C, Fortran or, in our case, CUDA. This approach also permits to invoke hybrid multi-threaded and CUDA codes that simultaneously execute tasks on both CPU and GPU, and attaining and efficient use of all the available resources in the hardware.

3 High performance band Lyapunov solver

As previously stated, the main computational kernels during the computation of the LR-ADI method for the solution of band Lyapunov matrix equations are the band matrix-matrix product and the solution of band linear systems. We present an implementation of the LR-ADI method based on tuned implementations of both operations for a CPU-GPU architecture. These kernels are integrated in the LYAPACK library via USFs. Specifically, the library plugs the USFs into the two main BMOs in the solver, namely the product of matrices and the linear systems solver. Thus, we obtain a specific Lyapunov solver for the band case. We emphasize that the performance of this solver is dictated by the performance of the USFs provided.

Additionally, our implementations aim at reducing the number and volume of data transfers between the CPU and the GPU memory address spaces, by re-utilizing data that are already in the GPU memory. For example, given that the shifted systems to be solved for V_{k+1} during the LR-ADI iteration in (2) share the same coefficient matrix, except for the shift parameter, we transfer the matrix A to the GPU memory only once, and construct/solve the different systems there.

3.1 Band linear system solver

The solution of a linear system, $AX = B$, can be tackled in the following steps:

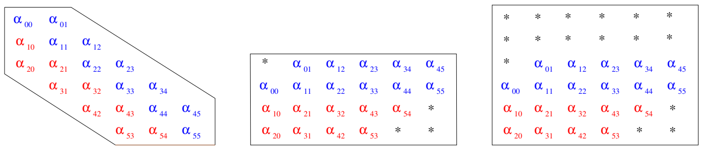
α_{00} α_{01}
α_{10} α_{11} α_{12}
α_{20} α_{21} α_{22} α_{23}
α_{31} α_{32} α_{33} α_{34}
α_{42} α_{43} α_{44} α_{45}
α_{53} α_{54} α_{55}
* α_{01} α_{12} α_{23} α_{34} α_{45}
α_{00} α_{11} α_{22} α_{33} α_{44} α_{55}
α_{10} α_{21} α_{32} α_{43} α_{54} *
α_{20} α_{31} α_{42} α_{53} * *
* * * * * *
* * * * * *
* α_{01} α_{12} α_{23} α_{34} α_{45}
α_{00} α_{11} α_{22} α_{33} α_{44} α_{55}
α_{10} α_{21} α_{32} α_{43} α_{54} *
α_{20} α_{31} α_{42} α_{53} * *

Figure 1: 6×6 band matrix with upper and lower bandwidths $k_l = 2$ and $k_u = 1$, respectively (left); packed storage scheme used in LAPACK (center); and modified storage scheme with $nb = 2$ (right).

1. Compute the factorization of A ($LU = A$).
2. Solve the lower triangular system ($LY = B$).
3. Solve the upper triangular system ($UX = Y$).

If A is a band matrix, then L and U are triangular band matrices. In particular, let k_l and k_u denote, respectively, the lower and upper bandwidths of A; then L is a lower triangular band matrix with bandwidth k_l, and U is an upper triangular band matrix with bandwidth $k_l + k_u$. Therefore, exploiting the structures of A, L and U, yields important reductions in the computational cost of the three steps. The library LAPACK provides a specific routine, GBSV, for the solution of band linear systems that exploit the structure of A in all the computations[1]. The LAPACK solver relies on the packed storage and the specific kernels for band matrices in BLAS. The use of a multi-threaded implementation of BLAS provides a parallel efficient solver for current CPUs.

The storage format for band matrices defined in BLAS permits a regular access pattern and, at the same time, allows important memory savings (see Figure 1 for an example). Nevertheless, it also constrains the performance of GBSV (see [3]).

In a previous work [3] we presented a GPU-based solver for band linear systems that mitigates the limitations encountered in the LAPACK routines. In particular, we introduce minor changes to the matrix storage scheme that, at the cost of a moderate increase in the memory requirements (see Figure 1 - right), allows to overcome the limitations encountered in routine GBSV and reports relevant gains in terms of performance and execution time. The solver is based on two hybrid CPU-GPU kernels that compute the LU factorization of a band matrix and solve a band triangular system. The implementation includes several HPC techniques, reaches a remarkable performance, and outperforms a LAPACK-like routine based on tuned CUBLAS kernels. However it still presents some drawbacks:

- Since it is a three-stage method, two synchronization points are mandatory.
- The matrix A is fully accessed twice, first during the computation of the LU factorization and later for the solution of the two band triangular systems (the

[1] For numerical reasons, pivoting is required in the factorization of A. Although, for simplicity, it is not included in our discussion, the LAPACK implementations and all the implementations presented include pivoting.

lower triangular system in stage 2, and the upper triangular part in the second solver).

We next present a solver with a single synchronization point and a reduced number of memory accesses. The central idea is a reorganization of the procedure, merging two stages: the factorization of A and the solution of the triangular system with L. This new approach presents the following advantages:

- Data locality: we anticipate the solution of the linear system involving L, as this operation is performed when the procedure is still manipulating these blocks.
- Concurrent execution: we can overlap two operations on two different processors, and therefore we can hide the computational cost of the least time-consuming one.
- Less memory accesses are required, since the lower triangular part of A (L) is accessed only once.

3.2 Band matrix-matrix multiplication

The band matrix-matrix product of the form

$$C = \beta C + \alpha AB, \tag{3}$$

is an operation with a large intrinsic parallelism. It is a level-3 BLAS operation and hence, high performance can be expected from it. The BLAS specification includes a routine to compute the band matrix-vector product (GBMV), but none for the computation of a band matrix-matrix product. Consequently, libraries that implement the BLAS specification provide efficient implementations of the band matrix-vector product but not for the related matrix-matrix product. This is the case for INTEL MKL or NVIDIA CUBLAS.

The effortless approximation to compute the related matrix-matrix product is to use the routine GBMV iteratively (one for each column of CB). However this approach leads to unnecessary memory accesses and, therefore, to an inefficient use of the memory hierarchy in the architecture. The drawbacks from this naive implementation can be overcomed using a blocked algorithm. Next, we present such a blocked algorithm, GBMM$_{BLK}$, and two implementations that off-load the most expensive computations to the GPU.

3.2.1 Algorithm GBMM$_{BLK}$

The algorithm is divided into two loops, see Figures 3 and 4. For simplicity, we assume momentarily that $\alpha = \beta = 1$ in (3). The outer loop (Figure 3) partitions the matrices B and C into blocks of c columns; at each iteration of the loop, the elements in the active column-block of C are computed. The inner loop progresses along the columns of C, from left to right, computing its elements. Figure 4 shows the operations performed in the inner loop. The matrices B and C are partitioned row-wise, while A

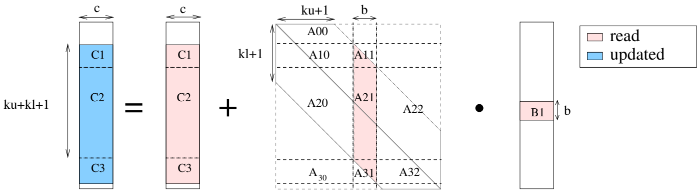

Figure 2: Elements read and updated during an iteration of the inner loop of the $\text{GBMM}_{BLK}^{inner}$ algorithm.

```
Algorithm: $[C] := \text{GBMM}_{BLK_{outer}}(C, A, B, k_u, k_l)$
Partition $C \rightarrow \left( C_L \,\|\, C_R \right)$, $B \rightarrow \left( B_L \,\|\, B_R \right)$
    where $C_L$, $B_L$ have 0 columns

while $n(C_L) < n(C)$ do
    Determine block size $c$
    Repartition
        $\left( C_L \,\|\, C_R \right) \rightarrow \left( C_0 \,\|\, C_1 \,|\, C_2 \right)$, $\left( B_L \,\|\, B_R \right) \rightarrow \left( B_0 \,\|\, B_1 \,|\, B_2 \right)$
          where $C_1$, $B_1$ have $c$ columns
    ========================================================
    $C_1 \quad := \quad \text{GBMM}_{BLK_{inner}}(C_1, A, B_1, k_u, k_l)$
    ========================================================
    Continue with
        $\left( C_L \,\|\, C_R \right) \leftarrow \left( C_0 \,|\, C_1 \,\|\, C_2 \right)$, $\left( B_L \,\|\, B_R \right) \leftarrow \left( B_0 \,|\, B_1 \,\|\, B_2 \right)$
endwhile
```

Figure 3: Outer loop of the algorithm that computes $C := A \cdot B + C$, with A general band with upper and lower bandwidth k_u and k_l, respectively.

requires a $3 \times 2 \rightarrow 5 \times 3$ re-partition. At each iteration, blocks A_{11}, A_{21} and A_{31} are accessed. One relevant property of this partitioning is that A_{11} and A_{31} are lower and upper triangular, respectively. Figure 2 details the blocks accessed and updated at a given iteration of the inner loop.

GBMM_{BLK} employs two blocksizes, one per loop: c defines the number of columns of C computed in a given iteration of the outer loop; and b is the number of columns of A that are accessed at each iteration of the inner loop. These two parameters need to be tuned in order to optimize the execution of the algorithm for a particular target architecture.

The main advantage of algorithm GBMM_{BLK} is that it can be implemented using BLAS-3 kernels and hence, a high performance can be expected from it.

3.2.2 Implementation GBMM

The GBMM routine is a straight-forward implementation of algorithm GBMM_{BLK}. In an initial stage, all the matrices are initially copied to the device. Then, βC is computed using the SCAL routine in CUBLAS. This is not a BLAS-3 operation, but presents a relatively small computational cost and can be efficiently computed on the device due

```
Algorithm: $[E] := \textsc{gbmm}_{BLK_{inner}}(E, A, D, k_u, k_l)$
Partition $E \rightarrow \left(\begin{array}{c} E_T \\ \hline E_M \\ \hline E_B \end{array}\right)$, $A \rightarrow \left(\begin{array}{c|c} A_{TL} & \\ \hline A_{ML} & A_{MR} \\ \hline & A_{BR} \end{array}\right)$, $D \rightarrow \left(\begin{array}{c} D_T \\ \hline D_B \end{array}\right)$
    where $E_T$, $D_T$ have 0 elements; $A_{TL}$ is $0 \times 0$ and $E_M$, $A_{ML}$ have $k_l$ rows
while $m(E_T) < m(E)$ do
    Determine block size $b$
    Repartition
        $\left(\begin{array}{c} E_T \\ \hline E_M \\ \hline E_B \end{array}\right) \rightarrow \left(\begin{array}{c} E_0 \\ \hline E_1 \\ E_2 \\ \hline E_3 \\ E_4 \end{array}\right)$, $\left(\begin{array}{c|c} A_{TL} & \\ \hline A_{ML} & A_{MR} \\ \hline & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & & \\ \hline A_{10} & A_{11} & \\ A_{20} & A_{21} & A_{22} \\ \hline & A_{31} & A_{32} \\ & & A_{42} \end{array}\right)$, $\left(\begin{array}{c} D_T \\ \hline D_B \end{array}\right) \rightarrow \left(\begin{array}{c} D_0 \\ \hline D_1 \\ \hline D_2 \end{array}\right)$
        where $D_1$ has $b$ rows;
              $E_1$ has 0 rows if $m(D_0) < (k_u + 1)$ and has $b$ rows otherwise;
              $E_3$ has 0 rows if $m(D_0) > (n(A) - k_l - 1)$ and has $b$ rows otherwise;
              $A_{11}$ is empty if $m(D_0) < (k_u + 1)$ and is $b \times b$ otherwise;
              $A_{31}$ is empty if $m(D_0) > (n(A) - k_l - 1)$ and is $b \times b$ otherwise
    ____________________________________________
        $E_1 := E_1 + A_{11} \cdot D_1$
        $E_2 := E_2 + A_{21} \cdot D_1$
        $E_3 := E_3 + A_{31} \cdot D_1$
    ____________________________________________
    Continue with
        $\left(\begin{array}{c} E_T \\ \hline E_M \\ \hline E_B \end{array}\right) \leftarrow \left(\begin{array}{c} E_0 \\ E_1 \\ \hline E_2 \\ \hline E_3 \\ E_4 \end{array}\right)$, $\left(\begin{array}{c|c} A_{TL} & \\ \hline A_{ML} & A_{MR} \\ \hline & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & & \\ A_{10} & A_{11} & \\ \hline A_{20} & A_{21} & A_{22} \\ & A_{31} & A_{32} \\ \hline & & A_{42} \end{array}\right)$, $\left(\begin{array}{c} D_T \\ \hline D_B \end{array}\right) \leftarrow \left(\begin{array}{c} D_0 \\ D_1 \\ \hline D_2 \end{array}\right)$
endwhile
```

Figure 4: Inner loop of the algorithm that computes $C := A \cdot B + C$, with A general band with upper and lower bandwidth k_u and k_l, respectively.

to its inherent parallelism. Next, the product of matrices is computed, and finally C is transferred back to the CPU. The update of E_2 (see Figure 4) requires a general matrix-matrix product (GEMM), an operation supported by CUBLAS. The updates of the blocks E_1 and E_3 require two matrix products involving triangular matrices, A_{11} and A_{31}, respectively. Routine TRMM of CUBLAS implements this operation in the form

$$C = \alpha AB, \tag{4}$$

but unfortunately differs from the functionality required by GBMM$_{BLK}$, since (4) overwrites C with the result. To overcome this problem, routine GBMM performs two operations in order to update E_1:

$$\begin{aligned} (\textsc{trmm}) \quad W &:= AD_1, & (5) \\ (\textsc{geam}) \quad E_1 &:= E_1 + W. & (6) \end{aligned}$$

Problem	n	$k_u = k_l$	# nonzeros	m
RAIL$_S$	5,177	139	35,185	7
RAIL$_M$	20,209	276	139,233	7
RAIL$_L$	79,841	550	553,921	7

Table 1: Instances of the RAIL example from the Oberwolfach model reduction collection.

Next to each operation in (5)-(6) is the CUBLAS routine that supports it. The update of E_3 is analogous. This procedure requires a reduced auxiliary storage $W \in \mathbb{R}^{b \times c}$.

High performance should be expected from this implementation since it fully employs tuned CUBLAS routines.

3.2.3 Implementation GBMM$_{ms}$

This implementation is based on the modified storage scheme (see Figure 1 - right) and presents some benefits over GBMM. In the modified storage scheme, the strictly lower triangular part of A_{31} is conveniently placed in the added rows, as is the strictly upper part of A_{11}. In consequence, there is no need to operate with them as triangular matrices and the updates of all the blocks in E can be performed in the same manner, specially via matrix-matrix products (routine GEMM from CUBLAS). In fact, this allows to go a step further, and the updates of $[{E_1}^T, {E_2}^T, {E_3}^T]^T$ can be merged and performed by means of a single matrix-matrix product. This approach presents two main advantages: it performs a reduced number of invocations to CUBLAS kernels, and it avoids operations involving small triangular matrices.

On the other hand, there are also two drawbacks related to this implementation. First of all, the memory requirements are enlarged. In addition, the number of arithmetic operations is also increased, as it operates with the null elements that lie in the strictly upper and lower triangles of A_{11} and A_{31}.

4 Experimental Evaluation

We evaluate the performance of the new codes for the solution of model order reduction problems extracted from the Oberwolfach benchmark collection [11]. In particular, from three instances of the same problem that differ in the dimension of the coefficient matrix; see Table 1. As the coefficient matrix in the RAIL problem is a sparse but not band matrix, we employ the Reverse Cuthill-McKee method [7] to transform this data into a band matrix. The execution times of the proposed solver are compared with those obtained with the LYAPACK CPU-based solver.

The performance evaluation was carried out using two different hardware platforms: An INTEL i7-2600 processor at 3.3 GHz with 8 GB of RAM connected to a NVIDIA S2070 (Platform I), and an INTEL i3-3220 processor at 3.4 GHz with 16 GB of RAM connected to a NVIDIA K20 (Platform II); In both platforms we use the CentOS Rel.

Platform	Problem	Version	USFs time (s)	Total time (s)	speedup
Platform I	RAIL$_S$	CPU	2.59	2.66	–
		CPU-GPU	1.79	1.88	1.4
	RAIL$_M$	CPU	19.61	19.72	–
		CPU-GPU	9.40	9.50	2.1
	RAIL$_L$	CPU	232.47	232.73	–
		CPU-GPU	54.04	54.31	4.3
Platform II	RAIL$_S$	CPU	3.57	3.65	–
		CPU-GPU	2.55	2.64	1.4
	RAIL$_M$	CPU	30.74	30.87	–
		CPU-GPU	12.75	12.87	2.4
	RAIL$_L$	CPU	408.54	408.93	–
		CPU-GPU	74.04	74.65	5.4

Table 2: Experimental comparison of the CPU and hybrid CPU-GPU lyapack solvers in Platform I and Platform II.

6.4 O.S. and the `gcc` v4.4.7 compiler. The CPU solver makes an intensive use of kernels intel mkl 11.1, while the hybrid CPU-GPU solver is built upon intel mkl 11.1 and nvidia cuda/cublas 5.0 kernels.

Table 2 shows the results obtained by the hybrid CPU-GPU and the CPU-based lyapack solvers. The first three columns show the platform, the instance of the RAIL problem, and the solver employed in that order. The execution time required by the USFs and the whole solver are reported in the next columns, while the last column shows the speed-up of the CPU-GPU implementation with respect to the CPU one.

These results exhibit the relevance of the USFs in the entire process, since an average they require more than 97% of the total time. Thus, the optimization efforts must focus on the USFs. The new solver exhibits a higher scalability, yielding better speed-ups for larger problems. In particular, the small problem presents a 1.4× speedup, while the largest problem evaluated reports a speedup up to 5.4×. The best time of both solvers is obtained in Platform I. This is explained by the superiority of the CPU in Platform I. The CPU solver in lyapack is more than 75% faster in the platform equipped with the intel i7-2600 processor. Also the hybrid CPU-GPU solver benefits from the more powerful CPU in Platform I. Although most of the computations in the hybrid solver are off-loaded to the GPU, it is still an hybrid code with synchronization points between the CPU and the GPU processors. Nevertheless, in the hybrid approach the relevance of the CPU processor is mitigated and, hence, while the CPU solver is nearly 2× faster in Platform I, the hybrid CPU-GPU solver is only 40% faster. Note also, that the GPU in Platform II is more powerful than that of Platform I.

5 Concluding Remarks

We have presented new hybrid CPU-GPU routines that accelerate the solution of band Lyapunov equations by off-loading the computationally expensive operations to the GPU. We extended the LYAPACK toolbox with a CPU-GPU solver that includes an efficient data manipulation to minimize transferences between host and device memories, a tuned band matrix-matrix multiplication reformulated to leverage BLAS-3 operations, and a novel band linear system solver based on a tailored storage scheme, look-ahead techniques and overlapped CPU-GPU computations.

The experimental results obtained using three test cases (with dimensions that vary from 5,177 to 79,841, and bandwidths from 131 to 550), extracted from the Oberwolfach benchmark collection, on two platforms equipped with an NVIDIA 2070 and an NVIDIA K20 GPU respectively, reveal speed-ups up to 5.4× when compared with the corresponding solver based on the INTEL MKL library. They also show the relevance of the USFs in the solver, as they require more than 97% of the total time. In consequence the LYAPACK design permits to effectively adapt the solver to the specific features of a given problem.

As part of future work, we plan to enhance the performance of Lyapunov solvers for other structured matrices, e.g., symmetric band matrices, and integrate these efforts into M.E.S.S. library (the successor of LYAPACK). Furthermore, we plan to study the impact of the new GPU-accelerated algorithms on energy consumption.

Acknowledgements

Ernesto Dufrechou and Pablo Ezzatti acknowledge the support from *Programa de Desarrollo de las Ciencias Básicas*, and *Agencia Nacional de Investigación e Innovación* of Uruguay. Enrique S. Quintana-Ortí was supported by project TIN2011-23283 of the Ministry of Science and Competitiveness (MINECO) and EU FEDER, and project P1-1B2013-20 of the *Fundació Caixa Castelló-Bancaixa* and UJI.

References

[1] Antoulas, A.: Approximation of Large-Scale Dynamical Systems. Philadelphia, PA (2005)

[2] Benner, P.: Contributions to the numerical solution of algebraic Riccati equations and related eigenvalue problems. Logos-Verlag (1997)

[3] Benner, P., Dufrechou, E., Ezzatti, P., Igounet, P., Quintana-Ortí, E.S., Remón, A.: Accelerating band linear algebra operations on GPUs with application in model reduction. Lecture Notes in Computer Science, Vol. 8584, pp. 386–400. Springer-Verlag (2014)

[4] Benner, P., Ezzatti, P., Kressner, D., Quintana-Ortí, E.S., Remón, A.: Accelerating model reduction of large linear systems with graphics processors. Lecture Notes in Computer Science, Vol. 7134, pp. 88–97. Springer (2010)

[5] Benner, P., Ezzatti, P., Kressner, D., Quintana-Ortí, E.S., Remón, A.: A mixed-precision algorithm for the solution of Lyapunov equations on hybrid CPU-GPU platforms. Parallel Computing **37**(8), 439–450 (2011)

[6] Benner, P., Mehrmann, V., Sorensen, D. (eds.): Dimension Reduction of Large-Scale Systems, *Lecture Notes in Computational Science and Engineering*, vol. 45. Springer-Verlag, Berlin/Heidelberg, Germany (2005)

[7] Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the 1969 24th National Conference, ACM '69, pp. 157–172. ACM, New York, NY, USA (1969)

[8] Datta, B.: Numerical Methods for Linear Control Systems. Elsevier Science (2004)

[9] Dufrechou, E., Ezzatti, P., Quintana-Ortí, E.S., Remón, A.: Accelerating the Lyapack library using GPUs. The Journal of Supercomputing **65**, 1114–1124 (2013)

[10] Green, M., Limebeer, D.: Linear robust control. Prentice-Hall information and system sciences series. Prentice Hall (1995)

[11] IMTEK: Oberwolfach model reduction benchmark collection. `http://www.imtek.de/simulation/benchmark/`

[12] Penzl, T.: A cyclic low-rank smith method for large sparse Lyapunov equations. SIAM J. Sci. Comput. **21**(4), 1401–1418 (1999)

[13] Penzl, T.: LYAPACK: A Matlab toolbox for large Lyapunov and Riccati equations, model reduction problems, and linear-quadratic optimal control problems. users guide (version 1.0) (2000)

[14] Petkov, P., Christov, N., Konstantinov, M.: Computational Methods for Linear Control Systems. Hertfordshire, UK (1991)

[15] Wachspress, E.: The ADI Model Problem. Springer New York (2013)

Max Planck Institute Magdeburg Preprints