Journal of Supercomputing manuscript No.
(will be inserted by the editor)

Factorized Solution of Generalized Stable Sylvester Equations Using Many-Core GPU Accelerators

Peter Benner · Ernesto Dufrechou · Pablo Ezzatti · Rodrigo Gallardo · Enrique S. Quintana-Ortí

the date of receipt and acceptance should be inserted later

Abstract We investigate the factorized solution of generalized stable Sylvester equations such as those arising in model reduction, image restoration, and observer design. Our algorithms, based on the matrix sign function, take advantage of the current trend to integrate high performance graphics accelerators (also known as GPUs) in computer systems. As a result, our realisations provide a valuable tool to solve large-scale problems on a variety of platforms.

Keywords Sylvester equation, Matrix sign function, Newton iteration, GPUs

1 Introduction

Consider the (continuous-time) *generalized Sylvester equation in factored form*

$$AXD + EXB + FG \;=\; 0, \tag{1}$$

where $A, E \in \mathbb{R}^{n\times n}$, $B, D \in \mathbb{R}^{m\times m}$, $F \in \mathbb{R}^{n\times p}$, $G \in \mathbb{R}^{p\times m}$, and $X \in \mathbb{R}^{n\times m}$ is the sought-after solution. Equation (1) has a unique solution if and only if $\alpha + \beta \neq 0$ for all $\alpha \in \Lambda(A, E)$ and $\beta \in \Lambda(B, D)$, where $\Lambda(U, V)$ denotes the generalized eigenspectrum of the matrix pencil $U - \lambda V$. In particular, this property holds for generalized *stable* Sylvester equations, where both $\Lambda(A, E)$ and $\Lambda(B, D)$ lie in the open left half plane. Sylvester equations have numerous applications in control theory, signal processing, filtering, model reduction,

Peter Benner
E-mail: benner@mpi-magdeburg.mpg.de
Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany.

Ernesto Dufrechou, Pablo Ezzatti and Rodrigo Gallardo
E-mail: {edufrechou,pezzatti,rgallardo}@fing.edu.uy
Facultad de Ingeniería, Universidad de la República, Uruguay.

Enrique S. Quintana-Ortí
E-mail: quintana@disca.upv.es
DISCA, Universitat Politècnica de València, Spain.

image restoration, etc., see, e.g., [1] and the references therein. In particular, in model reduction using cross-Gramians [2,3,4,5], image restoration [6], and observer design [7], $p \ll n, m$ and the solution X often exhibits a low (numerical) rank [8]. In such cases, it is beneficial to compute the factorized solution of the equation, from the perspectives of numerical accuracy as well as computational cost. Thus, we aim at computing a pair of matrices Y and Z^T, both with a small number of columns, such that $X = YZ$. Algorithms for the standard case ($E = I_n$, $D = I_m$) were first suggested in [9,1].

The factorized solution of Eq. (1) involves a considerable computational effort (in terms of arithmetic operations), asking for the application of high performance techniques when large problems are to be solved. To address this problem, following the general adoption of graphics processing units (GPUs) as powerful and ubiquitous parallel co-processors, in this work we propose efficient realizations based on the matrix sign function that are specifically designed to exploit this type of accelerators. Our experimental result confirm the efficacy of the sign function-based solvers and report a considerable performance advantage for this type of GPU-based solver.

Several previous works on parallel algorithms for solving linear matrix equations proposed implementations of an algorithm based on the (Hessenberg-)Schur decomposition of the coefficient matrices. As this is usually done using (Sca)LAPACK routines, the work in [10,11,12] concentrates on the solution of (quasi-)triangular Sylvester equations. The work in [12] focuses on task-parallelism, but not on GPU accelerators, and does not consider the generalized variant of the Sylvester equation targeted here; furthermore, [10,11] study the symmetric case of Lyapunov equations. The solution process in all three papers is not related to the iterative low-rank solver approach proposed in our paper, and the parallel performance of these solvers is limited by that of the QZ algorithm for the initial stage of the solution procedure. In comparison, we avoid the QZ algorithm completely and adhere to a matrix multiplication rich method that leverages the low-rank structure of the right-hand side for memory and computational savings. The paper [13] discusses an iterative scheme that treats the standard case of (1) without a factorized right-hand side, but from the timings and accuracy results listed in that paper, is not competitive with our approach. Our previous work on parallel and GPU-accelerated Lyapunov and Sylvester solvers summarized in [14] and further developed in [15] did not consider the generalized and factorized Sylvester case (1). As an additional contribution, we improve our GPU-enabled routine of the factorized solver and include two more variants, a hybrid CPU-GPU version and a dual-GPU version.

The rest of the paper is structured as follows. In Section 2, we briefly review the classical sign function solver for the generalized Sylvester equation, discuss the factored iteration, and propose an initial transformation of the equation that considerably reduces the cost per iteration. Then, in Section 3, we provide some details on how the Sylvester equation solvers are parallelized using many-core strategies. Numerical experiments reporting the accuracy and the high performance of the new methods on a hardware platform equipped with GPUs

are presented in Section 4. The final section summarizes the findings in this paper and gives some concluding remarks.

2 Iterative Schemes for Generalized Stable Sylvester Equations

Traditional solvers for generalized Sylvester equations consist of generalizations of the Bartels-Stewart (BS) method [16] and the Hessenberg-Schur method [17,18]. A different approach is to rely on iterative schemes for the computation of the matrix sign function. We adapt the basic Newton iteration used in this context [19] to solve the generalized formulations shown in Eq. (1) and obtain the solution in factored form. Similar algorithms have been proposed for the standard Sylvester equation in [1] and for the Lyapunov equation in [20,21]. We also propose an initial transformation of the equation that further reduces the cost of both the classical and the factored iterations.

2.1 Theoretical background

Consider a matrix $M \in \mathbb{R}^{l\times l}$ with no eigenvalues on the imaginary axis, and let $M = S \begin{bmatrix} J^- & 0 \\ 0 & J^+ \end{bmatrix} S^{-1}$ be its Jordan decomposition. The Jordan blocks in $J^- \in \mathbb{R}^{t\times t}$ and $J^+ \in \mathbb{R}^{(l-t)\times(l-t)}$ contain, respectively, the stable and unstable parts of $\Lambda(M, I_l)$. (Here, I_l denotes the square identity matrix of order l.) The *matrix sign function* of M is defined as $\mathrm{sign}(M) := S \begin{bmatrix} -I_t & 0 \\ 0 & I_{l-t} \end{bmatrix} S^{-1}$. By applying Newton's root-finding iteration to $M^2 = I_l$ in order to compute $\mathrm{sign}(M)$, with the starting point chosen as M, we obtain the Newton iteration for the matrix sign function:

$$M_0 := M, \quad M_{k+1} := \frac{1}{2}(M_k + M_k^{-1}), \quad k = 0, 1, 2, \ldots. \tag{2}$$

Under the given assumptions, the sequence $\{M_k\}_{k=0}^{\infty}$ converges to $\mathrm{sign}(M) = \lim_{k\to\infty} M_k$ [19], with an ultimately quadratic convergence rate. As the initial convergence may be slow, the use of acceleration techniques such as those in [22,23] is recommended. If X is a solution of Eq. (1), the similarity transformation defined by $\begin{bmatrix} I_n & X \\ 0 & I_m \end{bmatrix}$ can be used to block-diagonalize the block upper triangular matrix in Eq. (3) as shown in Eq. (4).

$$\tilde{H} = \begin{bmatrix} \tilde{A} & \tilde{C} \\ 0 & -\tilde{B} \end{bmatrix} = \begin{bmatrix} E^{-1}A & E^{-1}CD^{-1} \\ 0 & -BD^{-1} \end{bmatrix} \tag{3}$$

$$\begin{aligned} &\begin{bmatrix} I_n & X \\ 0 & I_m \end{bmatrix}^{-1} \begin{bmatrix} E^{-1}A & E^{-1}CD^{-1} \\ 0 & -BD^{-1} \end{bmatrix} \begin{bmatrix} I_n & X \\ 0 & I_m \end{bmatrix} = \\ &\begin{bmatrix} I_n & -X \\ 0 & I_m \end{bmatrix} \begin{bmatrix} E^{-1}A & E^{-1}CD^{-1} \\ 0 & -BD^{-1} \end{bmatrix} \begin{bmatrix} I_n & X \\ 0 & I_m \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & -B \end{bmatrix}. \end{aligned} \tag{4}$$

Using $\text{sign}\left(\tilde{H}\right)$, the relation given in (4), and the property of the sign function $\text{sign}\left(T^{-1}\tilde{H}T\right) = T^{-1}\text{sign}\left(\tilde{H}\right)T$, we derive Eq. (5) for the solution of Eq. (1).

$$\text{sign}\left(\tilde{H}\right) \;=\; \begin{bmatrix} -I_n & 2X \\ 0 & I_m \end{bmatrix}. \tag{5}$$

This relation forms the basis of the numerical algorithm derived next since it states that we can solve (1) by computing the matrix sign function of $\tilde{H}$ in (3).

$$H - \lambda K \;:=\; \begin{bmatrix} A & FG \\ 0 & -B \end{bmatrix} - \lambda \begin{bmatrix} E & 0 \\ 0 & D \end{bmatrix} = L(\tilde{H} - \lambda I_{n+m})M. \tag{6}$$

From Eq. (5), and using the equivalence of Eq. (6), where $L = \begin{bmatrix} E & 0 \\ 0 & I_m \end{bmatrix}$ and $M = \begin{bmatrix} I_n & 0 \\ 0 & D \end{bmatrix}$, we know that we can compute the solution of the generalized Sylvester equation by applying (2) to $\tilde{H}$. In doing so, we obtain in the first step

$$\begin{aligned} \tilde{H}_1 &= \tfrac{1}{2}(\tilde{H} + \tilde{H}^{-1}) = \tfrac{1}{2}(L^{-1}HM^{-1} + MH^{-1}L) \\ &= L^{-1}\left(\tfrac{1}{2}(H + LMH^{-1}LM)\right)M^{-1} = L^{-1}\left(\tfrac{1}{2}(H + KH^{-1}K)\right)M^{-1}. \end{aligned}$$

Repeating this calculation and denoting $H_0 := H = L\tilde{H}M$, we arrive at

$$H_{k+1} := \frac{1}{2}(H_k + LMH_k^{-1}LM) = \frac{1}{2}(H_k + KH_k^{-1}K), \quad k = 1, 2, \ldots, \tag{7}$$

so that $H_k = L\tilde{H}_k M$. Finally, taking limits on both sides, yields

$$H_\infty := \lim_{k\to\infty} H_k = L\text{sign}\left(\tilde{H}\right)M = \begin{bmatrix} -E & 2EXD \\ 0 & D \end{bmatrix}, \tag{8}$$

and $X = \frac{1}{2}E^{-1}H_{12}D^{-1}$, H_{12} denotes the upper right $n \times m$-block of H_∞.

2.2 Solution of the generalized Sylvester equation

In [1] it is observed that exploiting the block-triangular structure of the matrix pencil $H - \lambda K$, we obtain the following *classical generalized Newton iteration* for the solution of the generalized Sylvester equation (1):

$$\begin{aligned} A_0 &:= A, & A_{k+1} &:= \tfrac{1}{2}\left(A_k + EA_k^{-1}E\right), & & \\ B_0 &:= B, & B_{k+1} &:= \tfrac{1}{2}\left(B_k + DB_k^{-1}D\right), & & k = 0, 1, 2, \ldots. \\ C_0 &:= FG, & C_{k+1} &:= \tfrac{1}{2}\left(C_k + EA_k^{-1}C_kB_k^{-1}D\right), & & \end{aligned} \tag{9}$$

At convergence, the solution of (1) is computed by solving the linear equation

$$EXD = \frac{1}{2}\lim_{k\to\infty} C_k.$$

Also, from (8) we have $\lim_{k\to\infty} A_k = -E$ and $\lim_{k\to\infty} B_k = -D$, which suggests the stopping criterion

$$\max\left\{\frac{\|A_k + E\|_1}{\|E\|_1}, \frac{\|B_k + D\|_1}{\|D\|_1}\right\} \le \tau, \tag{10}$$

where τ is a tolerance threshold. One might choose $\tau = \gamma\varepsilon$ for the machine precision ε and, for instance, $\gamma = n$ or $\gamma = 10\sqrt{n}$. However, as the terminal accuracy sometimes cannot be reached, in order to avoid stagnation it is better to choose $\tau = \sqrt{\varepsilon}$ and to perform 1 to 3 additional iterations once this criterion is satisfied. Due to quadratic convergence of the Newton iteration (2), this is usually sufficient to reach the attainable accuracy, as already suggested and explained in the context of sign function based matrix equation solvers in [20].

2.3 Factored solution of the generalized Sylvester equation

In order to obtain a factorized solution of (1), we rewrite the iteration for C_k as two separate iterations:

$$\begin{aligned} F_0 &:= F, \quad F_{k+1} := \tfrac{1}{\sqrt{2}} \left[F_k,\ EA_k^{-1}F_k \right], \\ G_0 &:= G, \quad G_{k+1} := \tfrac{1}{\sqrt{2}} \begin{bmatrix} G_k \\ G_k B_k^{-1} D \end{bmatrix}, \end{aligned} \qquad k = 0, 1, 2, \ldots,$$

so that $C_{k+1} = F_{k+1}G_{k+1}$. Although this iteration is cheaper during the initial steps if $p \ll n, m$, this advantage is lost as the iteration advances since the number of columns in F_{k+1} and the number of rows in G_{k+1} is doubled in each iteration step. This can be avoided by applying a similar technique as that employed in [20] for the factorized solution of generalized Lyapunov equations. Let $F_k \in \mathbb{R}^{n\times p_k}$ and $G_k \in \mathbb{R}^{p_k\times m}$. We first compute a rank-revealing QR (RRQR) factorization [24] of G_{k+1} as defined above; that is, we calculate

$$\frac{1}{\sqrt{2}} \begin{bmatrix} G_k \\ G_k B_k^{-1} D \end{bmatrix} = U R \Pi_G, \quad R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix},$$

where U is orthogonal, Π_G is a permutation matrix, and R is upper triangular with $R_1 \in \mathbb{R}^{r\times m}$ of full row-rank. Then, we compute a RRQR factorization of $F_{k+1}U$:

$$\frac{1}{\sqrt{2}} \left[F_k,\ EA_k^{-1}F_k \right] U = V T \Pi_F, \quad T = \begin{bmatrix} T_1 \\ 0 \end{bmatrix},$$

where V is orthogonal, Π_F is a permutation matrix, and T is upper triangular with $T_1 \in \mathbb{R}^{t\times 2p_k}$ of full row-rank. Partitioning $V = [\, V_1,\ V_2 \,]$, with $V_1 \in \mathbb{R}^{n\times t}$, and computing

$$[\, T_{11},\ T_{12} \,] := T_1 \Pi_F, \qquad T_{11} \in \mathbb{R}^{t\times r},$$

we then get as the new iterates

$$F_{k+1} := V_1 T_{11}, \qquad G_{k+1} := R_1 \Pi_G,$$

which satisfy $C_{k+1} = F_{k+1}G_{k+1}$. Setting

$$Y := \frac{1}{\sqrt{2}} E^{-1} \lim_{k\to\infty} F_k, \qquad Z := \frac{1}{\sqrt{2}} \lim_{k\to\infty} G_k D^{-1},$$

we obtain the solution (1) in factored form $X = YZ$. If X has low numerical rank, the factors Y and Z will have a low number of columns and rows, respectively, and the storage space and computation time needed for the factored iteration will be lower than that of the classical iteration. In such case, $r, t \ll m, n$, and the cost of the current iteration for the factorized solution is $\frac{14}{3}(n^3 + m^3) + \mathcal{O}(2(n+m)^2)$ flops, where the cubic part comes from solving the linear systems and computing the matrix products $EA_k^{-1}E$ and $DB_k^{-1}D$; see [1, Section 4] for details of the complexity analysis .

2.4 Numerical performance

We next analyze the accuracy of the new Sylvester solvers borrowing examples from [25, 26]. We use IEEE double-precision floating-point arithmetic with machine precision $\varepsilon \approx 2.2204 \times 10^{-16}$. For the numerical evaluation, we implemented two MATLAB functions:

- `ggcsne`: The classical generalized Newton iteration for the generalized Sylvester equation in factored form as given in (9).
- `ggcsnc`: The factored variant of the generalized Newton iteration for the generalized Sylvester equation in factored form.

We compare these functions with the BS method as implemented in function `lyap` from the MATLAB Control Toolbox. As the BS solver in MATLAB cannot deal with the generalized Sylvester equation, we apply it to the transformed standard Sylvester equation $(E^{-1}A)X + X(BD^{-1}) + E^{-1}FGD^{-1} = 0$.

Example 1. A basic test case aimed to compute the *cross-Gramian* matrix W_{co} of a generalized linear time-invariant system of the form

$$M\dot{x}(t) = -Kx(t) + Bu(t), \quad y(t) = Cx(t). \tag{11}$$

This matrix is given by the solution of the generalized Sylvester equation

$$K\hat{W}_{co}M + M\hat{W}_{co}K + BC = 0, \tag{12}$$

and $W_{co} = \hat{W}_{co}M$. The cross-Gramian contains information of certain properties of the linear system [4] and can also be used for model reduction [2]. We employ the solvers to compute $\hat{W}_{co}$ for a system described in [25, Example 4.2] which comes from a model for heat control in a thin rod. The physical process is modeled by a linear-quadratic optimal control problem for the instationary 1D heat equation. Semi-discretization in space using finite elements leads to a system of the form (11), where M and K are the mass matrix and stiffness matrix, respectively, of the finite element approximation. Mesh refinement leads to systems of different orders n. The other parameters in this example are set to $a = 0.01$, $b = 2$, $c = 1$, $\beta_1 = 0$, $\beta_2 = 0.1$, $\gamma_1 = 0.9$, $\gamma_2 = 1$.

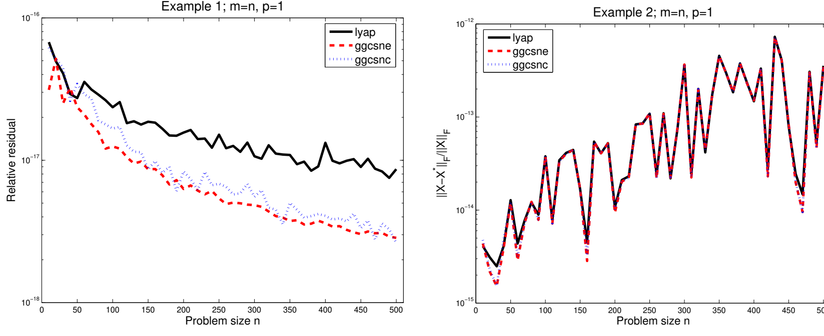

Fig. 1 Numerical performance of the generalized Sylvester equation solvers applied to Examples 1 (left) and 2 (right).

The left-hand side plot in Fig. 1 shows the results for various problem dimensions. As a measure of the quality of the solutions, we report the relative residuals

$$\frac{\|K\hat{W}_{co}^{*}M + M\hat{W}_{co}^{*}K + BC\|_F}{2\|K\|_F\|\hat{W}_{co}^{*}\|_F\|M\|_F + \|B\|_F\|C\|_F},$$

where $\hat{W}_{co}^{*}$ denotes the computed solution. For this example, the two factored solvers outperform the BS method by a margin that grows with the problem dimension n.

Example 2 [26, Example 13]. In this case we choose for real $a, b, d, e > 1$

$$\begin{aligned}
&\hat{A} = \mathrm{diag}\left(1, a, a^2, \ldots, a^{n-1}\right), && \hat{B} = \mathrm{diag}\left(1, b^{-1}, b^{-2}, \ldots, b^{-(n-1)}\right)\\
&\hat{D} = \mathrm{diag}\left(-1, -d^{-1}, -d^{-2}, \ldots, -d^{-(n-1)}\right), && \hat{E} = \mathrm{diag}\left(-1, -e, -e^2, \ldots, -e^{n-1}\right),\\
&\hat{H} = vv^T, \quad v = [1, 2, \ldots, n]^T && \hat{C} = -\hat{H}\hat{D} - \hat{H}\hat{B},
\end{aligned}$$

where the parameters a, b, d, and e regulate the eigenvalue distribution of the corresponding matrices. We then employ an equivalence transformation defined as $T = H_2 S H_1$, where

$$\begin{aligned}
H_1 &= I_n - \tfrac{2}{n} h_1 h_1^T, & h_1 &= [1, 1, \ldots, 1]^T,\\
H_2 &= I_n - \tfrac{2}{n} h_2 h_2^T, & h_2 &= \left[1, -1, \ldots, (-1)^{n-1}\right]^T,\\
S &= \mathrm{diag}\left(1, s, \ldots, s^{n-1}\right), \; s > 1,
\end{aligned}$$

to transform the matrices of the equation into

$$A = T^{-T}\hat{A}T^T,\;\; B = T\hat{B}T^{-1},\;\; D = T\hat{D}T^{-1},\;\; E = T^{-T}\hat{E}T^T,\;\; C = T^{-T}\hat{C}T^{-1}.$$

The factorized right-hand side matrix is then given by $F = -T^{-T}v$ and $G = v^T(D+B)T^{-1}$. In this example we set the parameters as $a = 1.001$, $b = 1.004$, $d = 1.002$, $e = 1.003$, and $s = 1.01$.

The right-hand side plot in Fig. 1 compares the relative errors in the computed solution X^*, $\frac{\|X - X^*\|_F}{\|X\|_F}$, for the different methods. The errors for all

three algorithms are remarkably similar and as small as could be expected from numerically backward stable methods.

3 Many-Core Versions

In this section we describe our GPU-accelerated realisations of the factored solver. Our routines off-load the most expensive computational stages of the method to the accelerator, leaving less-demanding or hardly parallelizable operations to the CPU.

Single-GPU variant, ggcsnc$_{gpu}$. As we stated previously, the method proposed for the generalized stable Sylvester equation is based on two simultaneous matrix iterations. From the perspective of computational cost, these iterations involve two major operations: the matrix inversion (computed as a matrix factorization and the solution of two triangular linear systems) and a matrix update, which can be performed via a matrix multiplication (GEMM). Our experiments show that these two operations (which are performed on both iteration matrices) represent approximately 85% of the total cost of the method. Therefore, an important acceleration can be expected from off-loading these operations to the GPU. In our implementation, basic linear algebra kernels such as matrix products, transpositions and norms, are performed using the *cuBLAS* GPU-accelerated library, while more complex operations, such as LU factorizations and triangular system solves rely on the *cuSolver* library. It should be noted that both operations occur in every iteration of the procedure. Regarding the data transfers, the equation matrices A, B, D and E are sent to the device only once, prior to the beginning of the procedure. Contrarily, the factors of the solution matrix are retrieved back to the CPU at each iteration, since the compression stage that uses the RRQR factorization to reduce the number of columns/rows of the factors is performed in the CPU. Without compression, the sign function iteration duplicates the number of columns/rows of the left/right factors of the solution at each iteration. The compression procedure leverages the low-rank property of the factors to keep the size of the factors bounded.

Hybrid variant, ggcsnc$_{hyb}$. As our method for the Sylvester equation combines operations performed in the CPU with others performed in the GPU, it is interesting to analyse how the computation on both devices can be overlapped to maximize the utilization of the hardware. To allow this, it is necessary to re-define the computation workflow. Concretely, we overlap the compression stage (performed in the CPU) with the matrix factorizations and triangular solves (computed in the GPU) corresponding to the next iteration, a strategy commonly referred as look-ahead [27]. This is convenient because the result of the linear systems in a given iteration are used to update the iteration matrices, which are then compressed. This variant can achieve important accelerations when the compression runtime is comparable to the cost of the

Operation	Kernel	Device
$\chi = EA_k^{-1}$	GETRF+GETRS	GPU1
$\xi = B_k^{-1}D$	GETRF+GETRS	GPU2
do		
$A_{k+1} = \frac{1}{2}(A_k + \chi E)$ $F_{k+1} = [F_k, \chi F_k]$	GEMM MEMCPY	GPU1
$B_{k+1} = \frac{1}{2}(B_k + D\xi)$ $G_{k+1} = \frac{1}{2}\begin{bmatrix} G_k \\ G_k\xi \end{bmatrix}$	GEMM MEMCPY	GPU2
start transfer of F_{k+1} and G_{k+1} to the CPU		
$\chi = EA_{k+1}^{-1}$	GETRF+GETRS	GPU1
$\xi = B_{k+1}^{-1}D$	GETRF+GETRS	GPU2
$G_{k+1} = U \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \Pi_G$ $F_{k+1}U = V \begin{bmatrix} T_1 \\ 0 \end{bmatrix} \Pi_F$ $[T_{11},\ T_{12}] := T_1 \Pi_F$ $F_{k+1} := V_1 T_{11}$ $G_{k+1} := R_1 \Pi_G$	GEQP3+GESVD ORGQR+GEQP3+GESVD LAPMT GEMMM LAPMT	CPU
until convergence		

Fig. 2 Algorithmic formulation of the $\texttt{ggcsnc}_{2gpus}$ variant. The factorized Newton iteration for the sign function has been re-organized so that the two sequences that compose the method can be isolated and executed in different devices.

matrix factorizations and linear system solution.

Dual GPU variant, $\texttt{ggcsnc}_{2gpus}$ The Newton iteration involves two independent recurrences (for $\{A\}_k$ and $\{B\}_k$) with a third one (for $\{C\}_k$) that depends on the other two. However, in the factored variant of the method, the $\{C\}_k$ recurrence is replaced by two independent ones (for $\{F\}_k$ and $\{G\}_k$). In turn, $\{F\}_k$ depends on $\{A\}_k$, while $\{G\}_k$ depends on B_k. This allows to completely separate the ($\{A\}_k$,$\{F\}_k$)-iteration from the ($\{B\}_k$,$\{G\}_k$)-one, handling each in a separate device. This approach offers two distinct benefits. On the one hand, duplicating the computational power to solve the main stages of the algorithm can strongly reduce the required runtime (up to 2× in the ideal case). On the other hand, the duplication of the memory allows addressing problems of larger scale. Although the two recurrences are independent, the compression of the factors requires a synchronization. Specifically, the synchronization occurs before the RRQR factorization of $F_{k+1}U$, given that U is the orthogonal matrix resulting from the RRQR factorization of G_{k+1}. The communication of data between the CPU memory and the devices also occurs at this point, where $F_{k+1} = [F_k, EA^{-1}F_k]$ and $G_{k+1} = [G_k; G_kB^{-1}D]$ are transferred to the CPU memory to be compressed there. It is important to note that, if the compression procedure keeps the number of columns of F_k and rows of G_k small, the cost of these transfers and the compression itself will be small compared with that of the operations that involve the square matrices A_k and B_k (of dimension n and m respectively); see the outline of the $\texttt{ggcsnc}_{2gpus}$ variant in Fig. 2.

Problem	Dim.	ggcsnc	$\texttt{ggcsne}_{gpu}$	$\texttt{ggcsnc}_{gpu}$	$\texttt{ggcsnc}_{hyb}$	$\texttt{ggcsnc}_{2gpus}$
Example 1	256	191	105	100	91	73
	512	460	446	340	322	219
	1024	2108	2454	1625	1582	901
	2048	11826	18156	10030	9928	5887
	4096	138610	141689	73067	72852	42339
Example 2	256	79	35	30	30	20
	512	287	147	106	106	73
	1024	1343	952	608	605	386
	2048	9959	9644	5291	5238	3211
Example 3	256	129	68	63	56	40
	512	364	295	207	204	135
	1024	1760	1581	987	982	574
	2048	10200	11317	6087	6092	3599
	4096	79347	87657	44056	44029	25830

Table 1 Execution time (in milliseconds) for the baseline and proposed variants, on the three example problems and different problem sizes.

4 Experimental Evaluation

In this section we analyse the performance of the generalized Sylvester equation solvers using two platforms for the experiments. The executions involving GPUs were performed on a server equipped with an Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz, 64GB of RAM, and two GeForce GTX 980 Ti GPUs with 6GB of GDDR5 memory. The multicore CPU experiments were performed on a server with 2×20-core Intel Xeon Gold 6138 CPUs @ 2.00GHz and 128GB of RAM. IEEE double-precision floating-point arithmetic was used for all the executions, and the multicore runs were performed with MATLAB R2018b on top of a multi-threaded instance of Intel MKL. The GPU executions were performed using CUDA 10.2.

Example 3. We construct matrices $\hat{A} \in \mathbb{R}^{n\times n}$, $\hat{B} \in \mathbb{R}^{m\times m}$, $F \in \mathbb{R}^{n\times p}$ and $G \in \mathbb{R}^{p\times m}$ with random entries uniformly distributed in U[-1,1]. Matrix $\hat{A}$ is then "stabilized" as $\hat{A} := \hat{A} - \|\hat{A}\|_F I_n$. Finally, we compute the QR factorization $\hat{A} = QR$ and set $A := R$ and $E := Q^T$. Matrices B and D are analogously obtained from a QR factorization of $\hat{B} - \|\hat{B}\|_F I_n$.

For the experimental evaluation we include two extra versions as baseline variants. First, a CPU-based variant (ggcsnc) of the iterative solver to compute the factorized solution of the Generalized Sylvester equation implemented in MATLAB[1]. This is the only variant that we execute in the Intel Xeon Gold processor, enabling MATLAB to use 20 CPU threads (the configuration that offers the best performance). Second, a GPU version of the Generalized Sylvester solver that works with the full (non-factored) solution ($\texttt{ggcsne}_{gpu}$). Table 1 summarizes the runtime required by the baseline variants as well as the three GPU versions proposed in this work: $\texttt{ggcsnc}_{gpu}$, $\texttt{ggcsnc}_{hyb}$

[1] The most expensive operation of this method are matrix operations, which MATLAB off-loads to external libraries (such as BLAS or LAPACK). MATLAB adds a small amount of overhead which makes this baseline only slightly unfair from the runtime perspective.

Problem	Inv. & Upd.	%	Comp.	%	$\texttt{ggcsnc}_{hyb}$ Theo.	$\texttt{ggcsnc}_{hyb}$ Achi.	$\texttt{ggcsnc}_{2gpus}$ Theo.	$\texttt{ggcsnc}_{2gpus}$ Achi.
Example 1	9639.5	96.1	115.0	1.2	115.0	102.0	4818.2	4143.0
Example 2	5012.7	94.7	78.9	1.5	78.9	53.0	2506.4	2080.0
Example 3	5998.8	96.9	8.3	0.2	8.3	-4.5	2999.4	2488.2

Table 2 Runtime and percentage of the total runtime of the main stages of the solver (left), and comparison between the theoretical and achieved runtime difference with respect to variant $\texttt{ggcsnc}_{gpu}$ (right) for the test cases of dimension 2048. Runtime values are in milliseconds.

and $\texttt{ggcsnc}_{2gpus}$. These implementations are employed to solve three scalable test cases. In our evaluation, we select the cases corresponding to the following matrix dimensions: $n = m = 256, 512, 1024, 2048$ and 4096.

The results show that the advantage of $\texttt{ggcsne}_{gpu}$ baseline regarding the ggcsnc variant decreases with the problem size. This mostly indicates that the factored solution offers significant benefits for large instances of the evaluated examples. This is confirmed when comparing the $\texttt{ggcsnc}_{gpu}$ and $\texttt{ggcsne}_{gpu}$ variants. For the smallest test cases, with $n = m = 256$, the acceleration obtained by $\texttt{ggcsnc}_{gpu}$ is mild, and grows consistently with the size of the problem in all three examples. According to the results, the $\texttt{ggcsnc}_{hyb}$ variant does not show any significant performance difference with $\texttt{ggcsnc}_{gpu}$ for Examples 2 and 3, while it only presents a modest advantage in Example 1. The attainable acceleration by this variant is constrained by the time taken by the RRQR compression algorithm, which is analysed later on.

The experiments also reveal that the $\texttt{ggcsnc}_{2gpus}$ variant clearly outperforms the remaining alternatives in all the tested instances. The acceleration achieved with respect to the ggcsnc and $\texttt{ggcsnc}_{gpu}$ variants differs according to the test case, but is close to $3\times$ and $1.7\times$, respectively, for the largest instance of each example.

At this point, it is important to remark that the GPU platform has rather poor double precision performance, and hence the experimental setup favours the CPU. Although higher accelerations are expected for high performance-oriented GPUs, the results show that significant performance gains are obtained with less expensive multi-GPU platforms.

In order to complete the analysis, we compare the results of Table 1 with those that would be obtained if a perfect overlapping was achieved in each case. In this line, Table 2 (left) offers the runtime, and percentage of total runtime, required by the most important operations, for the test cases with matrices dimension 2048. Specifically, we include two main stages: the inversion and update of the iteration matrices; and the compression of the solution matrix. As stated previously, the matrix inversion and update are performed on two matrices independently, and represent the most computationally-demanding parts. The $\texttt{ggcsnc}_{2gpus}$ variant leverages task-parallelism to off-load each inversion-update sequence to a different GPU, which makes the major achievable runtime reduction equivalent to half the total runtime of these stages. In comparison, in the $\texttt{ggcsnc}_{hyb}$ version the previous stages are concurrently computed

with the matrix compression, which implies that the best improvement for this case is equal to the runtime taken by the compression (as this operation typically requires significantly less runtime than the other overlapped operations). In Table 2 (right) we summarize the theoretical and actual runtime reductions. For the $\texttt{ggcsnc}_{hyb}$ solver, the runtime reductions are strongly correlated with the theoretical values as the achieved reductions are mostly equal to the theoretical. It should be remarked that the computational cost of the compression stage depends of the number of rows and columns of the matrices involved. In the three examples of Table 2, these numbers are 36, 53 and 7 respectively, which means that this variant only offers benefits when n and m are large. Regarding the $\texttt{ggcsnc}_{2gpus}$ variant, the results show that the observed runtime reductions are close to the theoretical values. Specifically, these reductions exceed 80% of the peak for the examples of Table 2, and the benefits increase with the runtime.

5 Conclusions

We have discussed a matrix sign function-based scheme to directly obtain the factorized solution of the generalized stable Sylvester equation. The factored iteration (in conjunction with rank-revealing QR) allows significant savings in computation time and memory requirements in case the solution has low numerical rank. The novel algorithm can be efficiently parallelized. In this work, we have designed and evaluated implementations that efficiently leverage both data- and task-parallelism on platforms equipped with multicore processors and one or two GPUs. The experimental results confirm the efficacy of the sign function-based solvers and report a considerable advantage that can be realised on massively parallel hardware.

In future work, we intend to generalize our approach to harness distributed platforms equipped with several GPUs, in order to handle large-scale problems.

Acknowledgments. We acknowledge support of the ANII – MPG Independent Research Group: "Efficient Hetergenous Computing" at UdelaR, a partner group of the Max Planck Institute in Magdeburg.

References

1. P. Benner, E. S. Quintana-Ortí, and G. Quintana-Ortí, "Solving stable Sylvester equations via rational iterative schemes," *J. Sci. Comp.*, vol. 28, no. 1, pp. 51–83, 2005.
2. R. Aldhaheri, "Model order reduction via real schur-form decomposition," *Internat. J. Control*, vol. 53, no. 3, pp. 709–716, 1991.
3. P. Benner and C. Himpe, "Cross-Gramian-based dominant subspaces," *Adv. Comput. Math.*, vol. 45, no. 5, pp. 2533–2553, 2019.
4. K. Fernando and H. Nicholson, "On a fundamental property of the cross-Gramian matrix," *IEEE Trans. Circuits and Systems*, vol. CAS-31, no. 5, pp. 504–505, 1984.
5. C. Himpe and M. Ohlberger, "Cross-Gramian based combined state and parameter reduction for large-scale control systems," *Mathematical Problems in Engineering*, vol. 2014, p. 843869, 2014.

6. D. Calvetti and L. Reichel, "Application of ADI iterative methods to the restoration of noisy images," *SIAM J. Matrix Anal. Appl.*, vol. 17, pp. 165–186, 1996.
7. B. Datta, *Numerical Methods for Linear Control Systems Design and Analysis*. Elsevier Press, 2003.
8. L. Grasedyck, "Existence of a low rank or H-matrix approximant to the solution of a Sylvester equation," *Numer. Lin. Alg. Appl.*, vol. 11, pp. 371–389, 2004.
9. P. Benner, "Factorized solution of Sylvester equations with applications in control," in *Proc. Intl. Symp. Math. Theory Networks and Syst. MTNS 2004*, 2004.
10. M. Köhler and J. Saak, "On GPU acceleration of common solvers for (quasi-) triangular generalized Lyapunov equations," *Par. Comp.*, vol. 57, pp. 212 – 221, 2016.
11. M. Köhler and J. Saak, "On BLAS level-3 implementations of common solvers for (quasi-) triangular generalized Lyapunov equations," *ACM Trans. Math. Softw.*, vol. 43, no. 1, art. no. 3, 2016.
12. A. Schwarz and C. Mikkelsen, "Robust task-parallel solution of the triangular Sylvester equation," in *International Conference on Parallel Processing and Applied Mathematics*. Springer, Cham, 2019.
13. M. Xiao, Q. Lv, Z. Xing, and Y. Zhang, "A parallel two-stage iteration method for solving continuous Sylvester equations," *Algorithms*, vol. 10, no. 3, art. no. 95, 2017.
14. P. Benner, P. Ezzatti, H. Mena, E. S. Quintana-Ortí, and A. Remón, "Solving matrix equations on multi-core and many-core architectures," *Algorithms*, vol. 6, no. 4, pp. 857–870, 2013. [Online]. Available: https://www.mdpi.com/1999-4893/6/4/857
15. E. Dufrechu, P. Ezzatti, E. S. Quintana-Ortí, and A. Remón, "Accelerating the Lyapack library using GPUs," *J. Supercomput.*, vol. 65, no. 3, p. 1114–1124, Sep. 2013. [Online]. Available: https://doi.org/10.1007/s11227-013-0889-8
16. R. Bartels and G. Stewart, "Solution of the matrix equation $AX + XB = C$: Algorithm 432," *Comm. ACM*, vol. 15, pp. 820–826, 1972.
17. W. Enright, "Improving the efficiency of matrix operations in the numerical solution of stiff ordinary differential equations," *ACM Trans. Math. Softw.*, vol. 4, pp. 127–136, 1978.
18. G. H. Golub, S. Nash, and C. F. Van Loan, "A Hessenberg–Schur method for the problem $AX + XB = C$," *IEEE Trans. Aut. Control*, vol. AC-24, pp. 909–913, 1979.
19. J. Roberts, "Linear model reduction and solution of the algebraic Riccati equation by use of the sign function," *Internat. J. Control*, vol. 32, pp. 677–687, 1980, (Reprint Tech. Report No. TR-13, CUED/B-Control, Cambridge Univ., Engineering Dept., 1971).
20. P. Benner and E. S. Quintana-Ortí, "Solving stable generalized Lyapunov equations with the matrix sign function," *Numer. Algorithms*, vol. 20, no. 1, pp. 75–100, 1999.
21. P. Benner, J. Claver, and E. Quintana-Ortí, "Parallel distributed solvers for large stable generalized Lyapunov equations," *Parallel Proc. Letters*, vol. 9, no. 1, pp. 147–158, 1999.
22. R. Byers, "Solving the algebraic Riccati equation with the matrix sign function," *Linear Algebra Appl.*, vol. 85, pp. 267–279, 1987.
23. N. Higham, "Computing the polar decomposition—with applications," *SIAM J. Sci. Statist. Comput.*, vol. 7, pp. 1160–1174, 1986.
24. T. Chan, "Rank revealing QR factorizations," *Linear Algebra Appl.*, vol. 88/89, pp. 67–82, 1987.
25. J. Abels and P. Benner, "CAREX – a collection of benchmark examples for continuous-time algebraic Riccati equations (version 2.0)," SLICOT Working Note 1999-14, Nov. 1999, available from `http://www.slicot.org`.
26. M. Slowik, P. Benner, and V. Sima, "Evaluation of the linear matrix equation solvers in SLICOT," *J. Numer. Anal. Ind. Appl. Math.*, vol. 2, no. 1–2, pp. 11–34, 2007.
27. J. I. Aliaga, R. M. Badia, M. Barreda, M. Bollhöfer, E. Dufrechou, P. Ezzatti, and E. S. Quintana-Ortí, "Exploiting task and data parallelism in ILUPACK's preconditioned CG solver on NUMA architectures and many-core accelerators," *Parallel Comp.*, vol. 54, pp. 97–107, 2016.