A GPU aware mixed precision solver for low rank algebraic Riccati equations

Peter Benner[1,3], Ernesto Dufrechou[2], Pablo Ezzatti[2], Alfredo Remón[1], Jens Saak[1]

[1] *Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany.*
{benner,remon,saak}@mpi-magdeburg.mpg.de
[2] *Instituto de Computación, Universidad de la República, 11300–Montevideo, Uruguay.*
{edufrechou,pezzatti}@fing.edu.uy
[3] *Institut für Analysis und Numerik, Fakultät für Mathematik, Otto-von-Guericke Universität Magdeburg, Germany.*
peter.benner@ovgu.de

SUMMARY

We investigate different alternatives for the solution of algebraic Riccati equations on hybrid hardware platforms (i.e., CPUs+GPUs). We evaluate a mixed precision approach which uses single precision arithmetic to obtain an approximation to the solution and later improves it to the desired precision applying some steps of an economic iterative refinement. This method exploits the higher performance of the hardware to accelerate the solver when single precision arithmetic is employed and simultaneously obtains a high accuracy solution with the iterative refinement. We extend this approach to exploit the low rank property of the equation, when possible, to further improve its efficiency. The experimental evaluation shows that the mixed precision approach reports time and energy savings and also provides similar or even more accurate solutions than well-known methods like the sign function iteration or the structure-preserving doubling algorithm.

Received ...

KEY WORDS: Algebraic Riccati Equations; Low Rank; Matrix Equations; Mixed precision; Graphics Processing Units

1. INTRODUCTION

The solution of continuous time algebraic Riccati equations (AREs) is required in several scientific and engineering applications, e.g., in linear quadratic optimal control (LQOC) and model order reduction problems. It is a computationally intensive operation that in general involves $\mathcal{O}(n^3)$ floating-point operations (flops) and therefore, the use of high performance computing techniques and hardware is necessary whenever n takes moderate to large values ($n > 1\,000$) and further structure like sparsity in the coefficients cannot be exploited. Two of the most widespread methods to tackle this sort of equations are the sign function iteration and the structure-preserving doubling algorithm. Software packages such as MESS [1], PLiC [2], or the MATLAB Control System Toolbox™, which is partly based on the SLICOT library [3], provide support for the solution of AREs.

In the last decade, the use of hybrid hardware platforms, i.e. machines that include multicore processors combined with hardware accelerators (e.g., Graphics Processing Units, GPUs), has been growing within the scientific computing field in general, and in the high performance computing

*Correspondence to: Jens Saak, Max Planck Institute for Dynamics of Complex Technical Systems, 39106–Magdeburg, Germany. E-mail: saak@mpi-magdeburg.mpg.de.

(HPC) community in particular. GPUs were originally developed to perform the graphics processing in computers, avoiding the use of the CPU, thus, allowing the CPU to concentrate on the remaining computations. However, GPUs have been progressively employed as a powerful intrinsically parallel hardware architecture to efficiently implement applications involving, e.g. vector operations. This is true especially since NVIDIA released CUDA [4, 5] in 2007, presenting a framework for general purpose computing that enables the use of parallel processing cores in NVIDIA GPUs to solve a wide variety of computational problems more efficiently than using a CPU only. Different studies have demonstrated the benefits of using GPUs to accelerate the computation of matrix equations [6, 7, 8] and matrix Riccati equations in particular [9, 10].

Additionally, energy consumption has become one of the major restrictions for the design of future supercomputers because of the economic costs of electricity, the negative effect of heat on the reliability of hardware components, and the negative environmental impact. While the advances in the performance of the hardware platforms in the Top500 list [11] show that an Exascale system may be available in the next quinquennium [12, 13, 14], a system of that capacity built over current technology would dissipate ridiculously large amounts of energy [15]. This has turned the decrease of the energy consumed by widely used algorithms into a critical line of work in the HPC community [14].

Considering the previously described situation, in [16] we studied preliminary the use of mixed precision methods to solve AREs for full rank problems. More in detail, we discussed a two stage method, where the first step is based on a low precision SDA method, while the second stage refines the approximate solution following a Newton procedure.

In this paper, we extend and enhance our previous developments including a low rank variant of the previously studied two stage–mixed precision solver. Specifically, the principal contributions of the present effort are:

- Evaluating in depth the full rank mixed precision algorithm to solve AREs.
- Extending the two stage method to solve AREs in order to leverage the low rank property of the solution of several problems.
- Experimentally studying the novel low rank mixed precision AREs solver, from performance and energy consumption perspectives.

The rest of the paper is structured as follows. In Section 2, we revisit the principal strategies to solve AREs, in particular we study the sign function method, the structure-preserving doubling algorithm and the Newton iteration as refinement procedure. Later, in Section 3, we detail the mixed precision approaches, studying and developing the low rank variant of each stage. This is followed by the experimental analysis carried out to empirically evaluate the proposed mixed precision methods in Section 4. Finally, Section 5 summarizes the main concluding remarks of this effort and delineates future research directions.

2. SOLUTION OF ARES

In this article we consider the solution of continuous time algebraic Riccati equations (AREs) of the following form:

$$0 = \mathcal{R}_c(X) := Q + A^T X + XA - XGX, \tag{1}$$

where A, Q and $G \in \mathbb{R}^{n\times n}$ are given, and $X \in \mathbb{R}^{n\times n}$ is the sought-after solution. Under certain conditions [17], the ARE (1) has a unique c-stabilizing solution X_c, which is symmetric positive semidefinite. (Here, X_c c-stabilizing means that $A_c := A - GX_c$ is c-stable; i.e., it has all its eigenvalues in the open left half plane.)

A number of methods have been proposed for the solution of AREs (e.g., see [18]). In this section we briefly review two of the most popular: the sign function and the structure-preserving doubling algorithm (SDA) methods. Additionally, we review an iterative refinement scheme that is able to improve the precision of an acceptable initial solution by means of a Newton iteration.

2.1. The Sign Function method

Algorithm 1: `GECRSG`: Sign function to solve algebraic Riccati equations.

Input: Matrices A, G, Q from (1)
Output: Approximation to the stabilizing solution X_c

1 $Y_0 := H = \begin{bmatrix} A & G \\ Q & -A^T \end{bmatrix}$
2 **for** $k = 0, 1, 2, \ldots$ *until convergence* **do**
3 $\quad Y_{k+1} := \frac{1}{2}\left(Y_k + Y_k^{-1}\right)$ $\quad (16n^3\texttt{flops})$
4 Solve $\begin{bmatrix} Y_{01} \\ Y_{11} + I_n \end{bmatrix} X = -\begin{bmatrix} I_n + Y_{00} \\ Y_{10} \end{bmatrix}$ $\quad (13n^3\texttt{flops})$

The solution of an ARE (1) can be defined by the invariant subspaces of the Hamiltonian matrix H defined as

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

Additionally, it can be shown that from a basis of the H-invariant subspace corresponding to the n eigenvalues in the open left half of the complex plane, the c-stabilizing solution of the associated ARE [19] can be obtained. This solution can be computed by calculating the Sign Function of H,

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

and then resolving X from the deflating subspace property

$$[\operatorname{sign}(H) + I_{2n}] \begin{bmatrix} I_n \\ X \end{bmatrix} = 0$$

by solving an overdetermined linear system (e.g., via the least squares method). The procedure is summarized in Algorithm 1.

Note that the dimension of H doubles that of A and hence, a high performance matrix inversion kernel is mandatory to enable the solution of large problems. However, since the loop in `GECRSG` implements a Newton procedure, it exhibits a remarkable convergence rate that makes it very appealing, provided H has no eigenvalues on or very close to the imaginary axis. As the convergence criterion we use the one proposed in [6]. It is especially attractive, since it is computed concurrently with the update of the matrix for the next iteration.

2.2. The Structure-Preserving Doubling Algorithm

In the last years, the SDA has received considerable attention as an ARE solver because of its simplicity, efficiency, and convergence properties [20]. Note that the algorithm originally works on the discrete-time ARE, such that we need to apply a Cayley transformation first, to turn the continuous-time ARE (1) into its discrete-time counterpart. Here we will review only the practical aspects of its implementation, referring the reader interested in the theory behind this method to the above reference.

Algorithm 2 (`GESDA`) reflects a basic implementation of the SDA for the solution of an ARE. The major operations (from the computational point of view) are annotated to their right with the processing cost of a basic implementation. Let us consider only the iterative loop:

- The cost of the algorithm is $(2/3 + 16)n^3$ flops per iteration. Its high cost can be partially compensated by the parallel efficiency of the operations involved in the routine, namely, matrix-matrix products and linear system solves.

- A practical convergence criterion is to check during the iteration for

$$\frac{\|Y_k\|_F}{\|Y_{k+1}\|_F} < \tau_S, \tag{2}$$

with $\tau_S = \sqrt{\varepsilon} \cdot n$, and perform then 2 additional steps. The convergence of the iteration is asymptotically quadratic, which ensures the maximum attainable accuracy.

Algorithm 2: `GESDA`: SDA method for the solution of algebraic Riccati equations.

Input: Matrices A, G, Q from (1)
Output: stabilizing solution $X_c = X_{k+1}$

`/* Transformation */`
$\gamma := \max(1, 2\|A\|_F)$
$A_\gamma := A - \gamma I_n$
$\hat{Q} := QA_\gamma^{-1}$ $((\frac{2}{3}+2)n^3\texttt{flops})$
$\hat{W} := (A_\gamma^T + \hat{Q}G)^{-1}$ $(4n^3\texttt{flops})$
`/* Initialization */`
$A_0 := I_n + 2\gamma\hat{W}^T$
$G_0 := 2\gamma(A_\gamma^{-1}G)\hat{W}$ $((\frac{2}{3}+4)n^3\texttt{flops})$
$X_0 := 2\gamma\hat{W}\hat{Q}$ $(2n^3\texttt{flops})$
`/* Main loop */`
for $k = 0, 1, 2, \ldots$ *until convergence* **do**
  $\hat{W} := G_k X_k$ $(2n^3\texttt{flops})$
  $\hat{A} := (I_n + \hat{W})^{-1} A_k$ $((\frac{2}{3}+2)n^3\texttt{flops})$
  $Y_k := \hat{A} X_k A_k$ $(4n^3\texttt{flops})$
  $X_{k+1} := X_k + Y_k$
  if *not converged* **then**
    $G_{k+1} := G_k + A_k G_k (I_n + \hat{W}^T)^{-1} A_k^T$ $(6n^3\texttt{flops})$
    $A_{k+1} := A_k \hat{A}$ $(2n^3\texttt{flops})$

It should be highlighted that this method is rich in BLAS-3 operations, and these kinds of operations are most appropriate for modern hardware platforms.

2.3. An iterative refinement for AREs solutions

In Benner et al. [21], the authors describe an iterative method for the solution of an ARE. Specifically, given an approximation to the solution of the ARE, X_0, the procedure in Algorithm 3 (`GEIR`) performs an iterative refinement that successively approximates the solution X until the desired precision is reached. At every step, the `GEIR` method solves a Lyapunov equation.

Algorithm 3: `GEIR`**:** Newton method for the iterative refinement.

Input: Matrices A, G, Q from (1) and initial guess X_0 for the solution
Output: improved solution X_k

for $k = 0, 1, 2, \ldots$ *until convergence* **do**
  $P_k := Q + A^T X_k + X_k A - X_k G X_k$
  `Solve` $(A - GX_k)^T N_k + N_k (A - GX_k) = P_k$
  $X_{k+1} := X_k + N_k$

In practice, provided a relatively accurate X_0, a few steps of algorithm `GEIR` are enough to get the desired solution as this procedure is a variant of Newton's method for AREs, indicating quadratic

convergence. The suitability of GEIR requires a cheap method to compute the initial guess X_0 and an efficient Lyapunov solver.

3. MIXED PRECISION ARE SOLVERS

In this section we propose mixed precision approaches for solving AREs. We start by revisiting the full rank mixed precision method and later describe the details of the design and implementation of our new low rank version.

3.1. Full rank mixed precision AREs solver

Our first approach to define a mixed precision solver for AREs is a direct extension of the iterative refinement described in Section 2.3.

Specifically, the initial approximation X_0 can be efficiently obtained executing some steps of the GESDA method, which can even be performed using single precision (SP) arithmetic. This way, the solver benefits from the better performance that the hardware offers in SP arithmetic computations (Intel CPUs are $2\times$ faster and this factor is larger for NVIDIA GPUs). Our implementation of this method leverages the GPU to compute highly parallel BLAS-3 or LAPACK operations, employing the CPU to address the operations with a lower parallelism degree.

For the second stage, i.e. the iterative refinement, we implement a Newton procedure using double precision (DP) arithmetic. More in detail, we exploit the efficient Lyapunov equation solver presented in [22]. The solver implements the sign function iteration (Algorithm 1) and relies on a tuned CPU-GPU matrix inversion kernel.

3.2. Low rank mixed precision solver

It is frequent in optimal control problems that one can only influence the system under investigation by very few control inputs and take only a small number of measurement outputs compared to the total number of degrees of freedom. Then, the matrices G and Q in (1) have low rank, since they are given as $G = BB^T$ and $Q = C^TC$, where $B \in \mathbb{R}^{n\times m}$ and $C \in \mathbb{R}^{p\times n}$, with p and m much smaller than n. In these situations, a fast decay of the singular values of the stabilizing solution X_c can be expected [23], which motivates the approximation of X_c by a low rank factorization. Note that we assume both B and C to be of full column and row rank, respectively, such that e.g. the Cholesky decomposition in (6) is well defined.

The previous mixed precision solver is unable to leverage the low rank properties of this kind of problems, i.e. it always works with the full dense $n \times n$ matrices. In the following sections we modify the existing algorithms such that they do work with the much smaller B and C factors, without forming G and Q explicitly. Consequently, we aim to obtain a factorized low rank approximation of the stabilizing solution X_c.

3.2.1. Low rank SDA solver

Our low rank variant of the SDA procedure is based on Algorithm 2. As the previous method, it begins by applying the Cayley transform [24] to the input data, so that the main iteration computes the solution to the discrete-time ARE instead of solving the continuous-time one.

We modified this procedure to take advantage of the low rank structure of the equation. The expressions for the matrices $\hat{W}$ and A_0 of Algorithm 2 can be derived from the previous expressions in a straight-forward way. However, in order to leverage the low rank properties of G and Q, and to modify the iteration so that it produces a factored low rank approximation to the stabilizing solution, the expressions for G_0 and X_0 have to be replaced by their respective low rank factors B_0 and C_0.

To obtain a symmetric factorization of the matrix

$$G_0 = 2\gamma(A_\gamma^{-1}BB^T)\hat{W} = B_0B_0^T, \tag{3}$$

Algorithm 4: `LRSDA`: Low rank variant of the SDA

Input: Matrices A, B, C forming (1) with $G = BB^T$ and $Q = C^TC$
Output: Low rank factored solution $X_c = C_{k+1}^T C_{k+1}$

`/* Apply Cayley transform to obtain the DARE from the CARE */`

$\gamma = \max(1, 2\|A\|_F)$
$A_\gamma = A - \gamma * I_n$
$W = (A_\gamma + BB^T A_\gamma^{-T} C^T C)^{-1}$
$A_0 = 2\gamma W + I_n$
$\hat{W} = CWB$
$Z_G Z_G^T \leftarrow I_m - (CA_\gamma^{-1}B)^T \hat{W}$ (*Cholesky factorization)*
$B_0 = \sqrt{2\gamma} A_\gamma^{-1} B Z_G^T$
$m = \texttt{num\_columns}(B_0)$
$Z_X Z_X^T \leftarrow I_p - \hat{W}(CA_\gamma^{-1}B)^T$ (*Cholesky factorization)*
$C_0 = \sqrt{2\gamma} Z_X C A_\gamma^{-1}$
$p = \texttt{num\_rows}(C_0)$
for $k = 0, 1, 2, \ldots$ *until convergence* **do**
  $K_k K_k^T \leftarrow I_m + B_k^T C_k^T C_k B_k$ (*Cholesky factorization)*
  $L_k L_k^T \leftarrow I_p + C_k B_k B k^T C_k^T$ (*Cholesky factorization)*
  $A_{k+1} = A_k^2 - A_k K_k^{-1} B_k B_k^T K_k^{-T} C_k^T C_k A_k$
  $B_{k+1} = [B_k, A_k B_k K_k^{-1}]$
  $C_{k+1} = [C_k^T, (L_k^{-1} C_k A_k)^T]^T$
  apply column compression to B_{k+1} and C_{k+1}
  $m = \texttt{num\_columns}(B_{k+1})$
  $p = \texttt{num\_rows}(C_{k+1})$

without forming it explicitly, since that matrix would be of size $n \times n$, we first applied the Sherman-Morrison-Woodbury formula (SMWF) to $\hat{W}$ which yields

$$\hat{W} = A_\gamma^{-T} - A_\gamma^{-T} C^T C (A_\gamma + BB^T A_\gamma^{-T} C^T C)^{-1} BB^T A_\gamma^{-T}. \tag{4}$$

Expanding $\hat{W}$ in (3) and rearranging adequately, we obtain

$$\begin{aligned} G_0 &= 2\gamma A_\gamma^{-1} BB^T (A_\gamma^{-T} - A_\gamma^{-T} C^T C \hat{W} BB^T A_\gamma^{-T}) \\ &= 2\gamma A_\gamma^{-1} B (B^T A_\gamma^{-T} - B^T A_\gamma^{-T} C^T C \hat{W} BB^T A_\gamma^{-T}) \\ &= 2\gamma A_\gamma^{-1} B (I_m - (CA_\gamma^{-1}B)^T C \hat{W} B) B^T A_\gamma^{-T}. \end{aligned} \tag{5}$$

Then, since G_0 should be symmetric and positive semi-definite (with a definite central factor by our assumptions on B and C) we perform a Cholesky factorization of the central factor (of size $m \times m$) to obtain

$$Z_G Z_G^T = I_m - (CA_\gamma^{-1}B)^T C \hat{W} B. \tag{6}$$

We can now rewrite $G_0 = B_0 B_0^T$, where

$$B_0 = \sqrt{2\gamma} A_\gamma^{-1} B Z_G. \tag{7}$$

A similar procedure is followed to form the matrix C_0. This time we obtain a symmetric factorization of

$$X_0 = 2\gamma \hat{W} C^T C A_\gamma. \tag{8}$$

Algorithm 5: Column compression method for an LDL^T factored matrix.

Input: Matrices L, D and compression tolerance τ
Output: Compressed matrices $\hat{L}, \hat{D}$

$Q, R \leftarrow$ qr_fact(L)
$V \Lambda V^T \leftarrow$ eigendecomposition of RDR^T with eigenvalues sorted by decaying magnitude
Set r to the number of eigenvalues with magnitude greater than τ
$\hat{L} = QV_{1:r}$
$\hat{D} = \text{diag}(\text{diag}(\Lambda)_{1:r})$

By replacing $\hat{W}$ with the SMWF expansion and rearranging the expression we obtain

$$\begin{aligned} X_0 &= 2\gamma(A_\gamma^{-T} - A_\gamma^{-T}C^T C\hat{W}^T BB^T A_\gamma^{-T})C^T CA_\gamma^{-1}) \\ &= 2\gamma(A_\gamma^{-T}C^T CA_\gamma^{-1} - A_\gamma^{-T}C^T C\hat{W}^T BB^T A_\gamma^{-T}C^T CA_\gamma^{-1}) \\ &= 2\gamma A_\gamma^{-T}C^T(CA_\gamma^{-1} - C\hat{W}^T BB^T A_\gamma^{-T}C^T CA_\gamma^{-1}) \\ &= 2\gamma(CA_\gamma^{-1})^T(I_p - C\hat{W}^T B(CA_\gamma^{-1}B)^T)CA_\gamma^{-1}. \end{aligned} \tag{9}$$

Then, as before, we perform the Cholesky factorization (of size $p \times p$)

$$Z_X Z_X^T = I_p - C\hat{W}^T B(CA_\gamma^{-1}B)^T. \tag{10}$$

We now have $X_0 = C_0^T C_0$, where

$$C_0 = \sqrt{2\gamma} Z_X^T CA_\gamma^{-1}. \tag{11}$$

After the initial matrices A_0, B_0 and C_0 have been computed, the recurrences for the matrices A, B and C are given by

$$\begin{aligned} A_{k+1} &= A_k^2 - A_k K_k^{-1} B_k B_k^T K_k^{-T} C_k^T C_k A_k, \\ B_{k+1} &= [B_k, A_k B_k K_k^{-1}], \\ C_{k+1} &= [C_k^T, (L_k^{-1} C_k A_k)^T]^T, \end{aligned} \tag{12}$$

where K_k and L_k are the Cholesky factors of $I_m + B_k^T C_k^T C_k B_k$ and $I_p + C_k B_k Bk^T C_k^T$ respectively. Note that these recurrences are equivalent to the ones for G and X in Algorithm 2 if we write $X_k = C_k^T C_k$ and $G_k = B_k B_k^T$. The resulting procedure is outlined in Algorithm 4.

As in the full rank version, we use single precision arithmetic to compute this stage. Additionally, the GPU is employed to perform the most computationally demanding operations, i.e. the Cholesky factorizations and BLAS-3 operations.

3.2.2. Low rank variant of the refinement

Our low rank variant of the Newton method is based on a loop with two main stages (Algorithm 3). The first stage takes the current approximation to the stabilizing solution $X_k = L_k D_k L_k$ as input and returns a low rank approximation to the residual $\mathcal{R}_c(X_k)$ of the ARE, such that $\mathcal{R}_c(X_k) \approx \hat{W}_k \hat{S}_k \hat{W}_k^T$ with $\hat{S}_k$ diagonal. To achieve this, we assemble block matrices W_k and S_k so that $\mathcal{R}_c(X) = W_k S_k W_k^T$. Here S_k is not diagonal, but we immediately compress these matrices using the LDL^T `compression` method described in Algorithm 5, which retrieves a $\hat{W}_k$ matrix with fewer columns than W_k and a diagonal $\hat{S}_k$ matrix. Obviously, the quality of the approximations can be controlled by the compression tolerance τ. As providing an efficient massively parallel implementation of the compression technique is itself a demanding endevour, in this version we execute the compression step entirely in the CPU, setting the optimization of this stage as future work.

Algorithm 6: Low rank variant of the iterative refinement.

Input: Matrices A, B, C forming (1) with $G = BB^T$ and $Q = C^TC$, p the number of rows in C, and a factored solution approximation L_0, D_0

Output: Improved factored solution L_{k+1}, D_{k+1}

for $k = 0, 1, 2, \ldots$ **do**

$\tilde{D} = D_k L_k^T B B^T L_k^T D_k$

$W_k = \begin{bmatrix} L_k & A^T L_k & C^T \end{bmatrix}$

$S_k = \begin{bmatrix} -\tilde{D} & D_k & 0 \\ D_k & 0 & 0 \\ 0 & 0 & I_p \end{bmatrix}$

Compress W_k and S_k into $\hat{W}_k$ and $\hat{S}_k$ using Algorithm 5

$\hat{A} = A - BB^T L_k D_k L_k^T$

Solve Lyapunov equation $\hat{A}\hat{L}_k\hat{D}_k\hat{L}_k^T + \hat{L}_k\hat{D}_k\hat{L}_k^T\hat{A}^T = \hat{W}_k\hat{S}_k\hat{W}_k^T$ for $\hat{L}_k$ and $\hat{D}_k$

$L_{k+1} = [L_k \;\; \hat{L}_k]$

$D_{k+1} = \begin{bmatrix} D_k & 0 \\ 0 & \hat{D}_k \end{bmatrix}$

Compress L_{k+1} and D_{k+1} using Algorithm 5

The second stage consists of solving the same Lyaunov equation as in Algorithm 3, only that this time G_k, X_k and P_k are in factorized form. We utilize a variant of the sign function Lyapunov solver, that also delivers a factorized aproximation $N_k = \hat{L}_k\hat{D}_k\hat{L}_k^T$ to the solution. This solver presents a straightforward adaption of the factored method proposed in [25] to the LDL^T structure. Consequently, the update of the solution X_{k+1} is replaced by appending adequately the $\hat{L}_k$ and $\hat{D}_k$ factors of the Lyapunov solution to the previous L_k and D_k matrices. The resulting process is summarized in Algorithm 6. From an implementation perspective, we extended the GPU hybrid LL^T factored Lyapunov solver so that it can manipulate the new LDL^T matrices and preserve their structure.

4. EXPERIMENTAL EVALUATION

The following paragraphs summarize the experimental evaluation performed for the novel mixed precision (MP) ARE solvers presented in this work. This evaluation focuses not only on the runtime required to solve the Riccati equations but also on the energy consumption implied by the different methods.

In order to provide a baseline to analyze the performance of the new methods, we also run experiments for two other GPU-based ARE solvers that use double precision (DP) arithmetic for their computations. The main details of the four methods are:

Sign function based solver This solver is built with the `GECRSG` procedure. Additionally, the variant employed presents a highly optimized CPU-GPU matrix inversion kernel, see [22] for details.

SDA-based solver The implementation evaluated in this work executes the most time consuming operations on the GPU, i.e. the $O(n^3)$ operations, while operations that exhibit a fine-grain parallelism are performed on the CPU. Whenever it is possible, both processors concurrently perform their tasks, resulting in significant time savings. Finally, the computation of inverses is replaced by the use of the LU factorization of the related matrix.

Full rank mixed precision This is the full rank variant of the mixed precision method described in Section 3. The implementation offloads the most time consuming operations to the GPU, in

GPU	Processor	NVIDIA K40 “Kepler” GK110B
	# Cores	2,880
	Memory	12 GB GDDR5
CPU	Processor	i7-4770
	# Cores	4
	Frequency	3.40 GHz
	Main memory	16 GB DDR3
SW	Compiler	`icc` 14.0.0
	CUDA Version	6.5

Table I. Platform employed in the experimental evaluation.

other words the $O(n^3)$ operations in the single precision SDA and the double precision matrix inversions in the Newton procedure.

Low rank mixed precision Low rank variant of the method described in Section 3. The implementation also uses the GPU to accelerate some of the most computationally demanding kernels similar to the full counterpart.

The rest of this section includes the description of the test cases employed, the main aspects of the hardware platform used for the experiments and, finally, the experimental results themselves and their analysis.

4.1. Test cases

Three test-cases of dimension $n = 1\,357$, $5\,177$, and $9\,669$ are employed to evaluate the routines. The test-cases evaluated were extracted from the Oberwolfach[†] benchmark collection. In particular, two instances of the STEEL PROFILE (with $n = 1\,357$ and $5\,177$) and another from the FLOW METER problem ($n = 9\,669$).

It should be noted that in all three cases, $G = BB^T$ and $Q = C^TC$, where $B \in \mathbb{R}^{n\times m}$ and $C \in \mathbb{R}^{p\times n}$ and $m, p \ll n$. For both instances of the STEEL PROFILE problem we have $m = 7$, $p = 6$, while for the FLOW METER problem $m = 1$ and $p = 5$.

4.2. Evaluation platform

The hardware platform used to perfom the experiments is based on an NVIDIA K40 GPU. This sort of graphics cards offers a theoretical performance in double precision arithmetic much higher than average consumer cards.

In a previous effort [16], we compared the performance of some of the solvers on two different GPU-based platforms, an NVIDIA K40 and an NVIDIA TitanX from the Maxwell generation. While the former is a HPC GPU, with improved double precision performance, the latter is a powerful consumer GPU with a remarkable performance in single precision, but $32\times$ slower when working in double. The conclusion extracted from this effort is that our mixed precision method (only the-full rank variant in this work) takes the most advantage of consumer GPUs, since our method leverages the single precision (SP) performance in the first stage. Taking this behaviour into account, this time we only include the results extracted in the more restrictive hardware for the mixed precision paradigm.

Table I details the hardware and software employed in our tests.

Power/energy was measured via RAPL to gauge the consumption from the servers package and DRAM, and the NVML library to obtain the energy dissipation from the GPU.

[†]Available at `http://cise.ufl.edu/research/sparse/matrices/Oberwolfach/index.html`, see also `https://portal.uni-freiburg.de/imteksimulation/downloads/benchmark`

4.3. Performance evaluation

We first evaluate the computational performance of the sign function and the SDA fixing the number of iterations of each solver so that they reach comparable accuracy results. The residual error is computed as

$$\text{RRes} = \|\mathcal{R}_c(X^*)\|_F / (\|Q\|_F + 2\|A\|_F \|X^*\|_F + \|G\|_F \|A\|_F^2). \tag{13}$$

The results summarized in Tables II and III show that the two solvers behave similarly for all test cases. Specifically, the SDA solver slightly outperforms the sign function solver, being 10% faster for the medium size instance and 20% faster for the larger one. This behaviour is explained by noting that the formulation of the SDA, strongly based on matrix products, is in general better suited to exploit the GPU than the sign function counterpart.

PROBLEM	# STEPS	SIGN FUNC.	SOLVER	TOTAL	REL. RES.
RAIL_1357	10	3.99	0.33	4.36	1.51E-17
RAIL_5177	11	70.93	10.71	82.17	4.96E-17
FLOW_9669	13	410.46	68.64	481.04	2.12E-10

Table II. Runtimes (in sec.) and relative residuals of the Sign Function solver.

PROBLEM	# STEPS	TIME	REL. RES.
RAIL_1357	24	2.17	4.96E-16
RAIL_5177	27	86.69	4.61E-16
FLOW_9669	24	419.60	7.99E-12

Table III. Runtimes (in sec.) and relative residuals of the SDA solver.

To evaluate our mixed precision schemes, we modified the number of SDA and iterative refinement steps. We fixed the parameters so that a comparable accuracy with the traditional DP methods is reached. The differing iteration numbers for the corresponding loops give an impression how they influence the total execution time. The optimization of these values for minimum execution time, while at the same time guaranteeing maximum accuracy, is a topic for another work, though.

The results for the full rank version are summarized in Table IV, where the column *Lyap* shows the number of iterations of the sign function method performed to solve the corresponding Lyapunov equation at each step of the GEIR algorithm. The configuations displayed in the table are those that reach an accurancy level similar to the one reached by traditional double precision variants of the Sign Function and SDA solvers. Considering the runtimes summarized in that table, it is clear that the mixed precision solver offers significant runtime reductions (for the larger test case almost 80%). In the medium size case, the differences are less drastic ranging between 20% and 30%. Finally, the full rank mixed precision variant does not offer any benefits for the smallest case. These results show that this variant offers scalability in the problem dimension and is better than the traditional approaches (sign function and SDA) when the dimension of the addressed problems is larger than a certain threshold, i.e. when the use of mixed precision can compensate its overhead.

Given that during the SDA, the sign function Lyapunov solver, and the Newton iteration of the low rank variant, the number of columns/rows of the factor of the respective solutions grows with each iteration, we consider a column/row compression technique to reduce the size of the factors. This compression can imply a considerable amount of runtime, since it involves the computation of the eigenvalues of the modified square center matrix, and can affect the accuracy of the result since typically some information is lost during the compression. However, for the compression tolerance used in these experiments (relative values of 10^{-7} and 10^{-16} for single and double precision procedures, respectively), we did not observe any significant impact on the accuracy when compressing the factors in all steps.

	#STEPS			RUNTIMES			
PROBLEM	GESDA	GEIR	Lyap.	GESDA	GEIR	TOTAL	Rel. res.
RAIL_1357	10	1	10	-	-	-	1.75E-14
	10	2	8	-	-	-	1.62E-14
	15	1	9	-	-	-	9.10E-15
	15	2	8	-	-	-	1.78E-15
	20	1	9	1.16	1.09	2.25	6.92E-16
	20	2	8	1.15	1.87	3.02	3.70E-16
RAIL_5177	10	1	10	-	-	-	6.53E-15
	10	2	9	-	-	-	4.99E-15
	15	1	10	-	-	-	1.44E-15
	15	2	9	-	-	-	1.00E-15
	20	1	10	33.03	17.24	50.27	7.42E-16
	20	2	9	33.45	29.89	63.34	1.31E-16
FLOW_9669	10	1	7	-	-	-	2.44E-09
	10	2	7	101.56	114.87	216.43	3.94E-13
	15	1	7	-	-	-	2.15E-09
	15	2	7	150.68	115.09	265.77	3.73E-13
	20	1	10	-	-	-	5.39E-10
	20	2	8	187.44	128.94	316.38	7.21E-15

Table IV. Runtimes (in sec.) and relative residuals reported by the mixed precision full rank solver.

Regarding the runtime dedicated to the compression, we noticed that compressing the solution every time that the factors are expanded turns out to be beneficial. The reason is that although skipping the compression step saves time in the current iteration, the compression step will be more costly in the next one, as this cost scales rapidly with the size of the factors. Considering that the accuracy reached by the different methods is not significantly affected, we utilize this strategy for all the experiments.

Table V summarizes the achieved accuracy of the low rank solver for different configurations of single precision SDA iterations, refinement steps, and sign function iterations inside the Lyapunov solver. The data shows that the effect of the number of steps performed by the SP solver on the accuracy reached diminishes as the dimension of the problem grows. As a consequence, the refinement steps have a strong impact on the final accuracy. This is specially relevant in the larger instance. Regarding the comparison between the accuracy obtained by the full and low rank solvers with the same number of iterations, it can be noted that the full rank solver obtains slightly better results, specially in the smaller cases. The execution times of the low rank solver are presented in the same table. The results show that the MP (i.e. mixed precision) low rank solver is able to significantly improve the execution times of the full rank counterpart. The analysis of the results reveals that the performance associated with this method has a similar behavior but improves the values reached by the full rank counterpart. Specifically, the resolution of the smallest case with the low rank variant implies more runtime than when the full rank version is employed. In the case of medium size, the differences in the runtime are limited but with values near to a 50% improvement when the low rank MP solver is compared with the sign function or SDA methods. Finally, for the largest case, the novel method outperfoms the traditional methods with acceleration values in the order of $3\times$. These results are aligned with the theory, since the benefits of the low rank variant become more important when the difference between the dimension n of the problem and the ranks m and p of the factors B and C is large enough to hide the overhead implied by the reshaping and compression of intermediate matrices.

PROBLEM	#STEPS			RUNTIMES			
	GESDA	GEIR	Lyap.	GESDA	GEIR	TOTAL	Rel. res.
RAIL_1357	10	1	10	-	-	-	1.74E-08
	10	2	8	-	-	-	5.98E-12
	15	1	9	-	-	-	8.52E-09
	15	2	8	-	-	-	7.98E-12
	20	1	9	-	-	-	3.99E-10
	20	2	8	-	-	-	2.19E-14
	5	3	6	0.46	4.05	4.51	2.94E-16
RAIL_5177	10	1	10	-	-	-	4.47E-09
	10	2	9	-	-	-	5.57E-13
	15	1	10	-	-	-	9.86E-10
	15	2	9	-	-	-	7.18E-13
	20	1	10	-	-	-	6.43E-10
	20	2	9	-	-	-	4.52E-13
	5	3	6	9.28	42.26	51.54	1.37E-16
FLOW_9669	10	1	7	-	-	-	1.87E-09
	10	2	7	-	-	-	9.92E-11
	15	1	7	-	-	-	1.06E-09
	15	2	7	-	-	-	3.26E-11
	20	1	10	152.43	79.72	232.15	3.77E-12
	20	2	8	147.06	129.16	276.22	1.45E-16
	8	3	4	59.08	92.93	152.01	5.02E-12

Table V. Runtimes (in sec.) and relative residuals reported by the low rank mixed precision solver.

4.4. Energy evaluation

In a second experiment, we measure the energy consumption related with the best configuration of each method in our experimental hardware platform. Tables VI, VII and VIII show the energy consumption of the four solvers.

SOLVER	PROBLEM	# STEPS	ENERGY (J)	RUNTIME (S)
SIGN FUNC.	RAIL_1357	10	668.36	4.36
	RAIL_5177	11	14,855.00	82.17
	FLOW_9669	13	93,936.00	481.04
SDA	RAIL_1357	24	457.43	2.17
	RAIL_5177	27	18,568.00	86.69
	FLOW_9669	24	96,166.00	419.60

Table VI. Energy of the double precision solvers.

PROBLEM	#STEPS			ENERGY (J)			RUNTIME (S)
	GESDA	GEIR	Lyap.	GESDA	GEIR	TOTAL	Total
RAIL_1357	20	1	9	202.94	148.93	351.87	2.25
RAIL_5177	20	1	10	6,044.80	3,278.40	9,323.20	50.27
FLOW_9669	10	2	7	16,545.00	24,737.00	41,282.00	216.43

Table VII. Energy of the mixed precision full rank solver.

Considering the traditional methods, i.e. sign function and SDA, the results suggest that the sign function method is more efficient than the SDA from the energy consumption perspective. Note that

PROBLEM	#STEPS			ENERGY (J)			RUNTIME (S)
	GESDA	GEIR	Lyap.	GESDA	GEIR	TOTAL	TOTAL
RAIL_1357	5	3	6	58.64	546.29	604.93	4.51
RAIL_5177	5	3	6	1,216.80	5,935.80	7,152.60	51.54
FLOW_9669	8	3	4	8,098.90	14,281.00	22,379.90	152.01

Table VIII. Energy of the mixed precision low rank solver.

the differences between the runtimes of both methods are larger than the differences in their energy consumption.

Regarding the mixed precision methods, the data extracted from the experiments shows that this strategy signifies important energy savings compared to the traditional methods evaluated. The difference between the energy consumption of the mixed precision and the double precision methods is larger than the difference in runtime, which indicates that the mixed precision strategy demands less power. Additionally, the improvement seems to increase with the dimension of the problem. It should be highlighted that in the largest case, FLOW_9669, the differences in energy consumption between the low rank MP version and the DP solvers are more than fourfold, while the runtime difference is only threefold.

Comparing the two versions of the mixed precision solver, it can be noticed that the low rank variant consumed less power than the full rank counterpart to reach similar levels of accuracy. This becomes more evident for the larger case, where the low rank variant strongly outperforms the full rank method in runtime ($1.4\times$ in FLOW_9669), making the differences in energy consumption even more important ($1.8\times$ in the same test case).

5. CONCLUDING REMARKS AND FUTURE WORK

In this work, we have addressed the use of mixed precision techniques for the numerical solution of algebraic Riccati equations. Specifically, we have extended our preliminary study, in which we presented a full rank mixed precision method to solve AREs, and we have developed and implemented a novel mixed precision solver for AREs able to leverage the low rank characteristics of a large number of problems. Additionally, both approaches exploit the computational power offered by the NVIDIA GPUs offloading the most demanding computation stages to this coprocessor.

The experimental results show that the mixed precision strategy manages to outperform well known methods to solve this kind of equations, like the SDA and the Sign Function methods, both in terms of runtime and energy consumption. Moreover, the low rank variant of our method is clearly superior to the original full rank version for problems that present this characteristic. The gains become larger with larger problems, showing the suitability of this strategy for large problems.

As part of future work we intend to perform a more detailed study in order to develop an automatic mechanism to select the optimal compression configuration. We are also interested in exploring other kinds of hardware platforms, as well as more levels of arithmetic precision.

REFERENCES

1. Benner P, Köhler M, Saak J. Matrix Equation Sparse Solver (MESS) library. URL `http://www.mpi-magdeburg.mpg.de/projects/mess/`.
2. Benner P, Quintana-Ortí ES, Quintana-Ortí G. PLiC library. URL `http://www3.uji.es/~quintana/plic/plic/`.
3. *SLICOT*. `http://www.slicot.org`.
4. Kirk D, Hwu W. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2010.
5. Farber R. *CUDA Application Design and Development*. Morgan Kaufmann, 2011.
6. Benner P, Ezzatti P, Kressner D, Quintana-Ortí ES, Remón A. A mixed-precision algorithm for the solution of Lyapunov equations on hybrid CPU-GPU platforms. *Parallel Computing* 2011; **37**(8):439–450, doi:10.1016/j.parco.2010.12.002.

7. Raczyński D, Stanisławski W. Controllability and observability Gramians parallel computation using GPU. *Journal of Theoretical and Applied Computer Science* 2012; **6**(1):47–66.
8. Dufrechu E, Ezzatti P, Quintana-Ortí ES, Remón A. Accelerating the Lyapack library using GPUs. *The Journal of Supercomputing* 2013; **65**(3):1114–1124, doi:10.1007/s11227-013-0889-8.
9. Peinado J, Ibañez JJ, Arias E, Hernández V. Speeding up solving of differential matrix Riccati equations using GPGPU computing and MATLAB. *Concurr. Comput. : Pract. Exper.* Aug 2012; **24**(12):1334–1348, doi:10.1002/cpe.1835.
10. Benner P, Ezzatti P, Mena H, Quintana-Ortí E, Remón A. Solving matrix equations on multi-core and many-core architectures. *Algorithms* 2013; **6**(4):857–870, doi:10.3390/a6040857.
11. The top500 list 2017. Available at `http://www.top500.org`.
12. Bergman K, Borkar S, Campbell D, Carlson W, Dally W, Denneau M, Franzon P, Harrod W, Hill K, Hiller J, *et al.*. Exascale computing study: Technology challenges in achieving exascale systems. DARPA IPTO ExaScale Computing Study 2008.
13. Dongarra J, Beckman P, Moore T, Aerts P, Aloisio G, Andre JC, Barkai D, Berthou JY, Boku T, Braunschweig B, *et al.*. The international ExaScale software project roadmap. *Int. J. of High Performance Computing & Applications* 2011; **25**(1):3–60, doi:10.1177/1094342010391989.
14. Duranton M, Black-Schaffer D, De Bosschere K, Maebe J. The HiPEAC vision for advanced computing in Horizon 2020 2013. URL `http://www.hipeac.net/v13`.
15. The Green500 list 2017. Available at `http://www.green500.org`.
16. Benner P, Dufrechou E, Ezzatti P, Remón A. Studying mixed precision techniques for the solution of algebraic Riccati equations. *2nd Workshop on Power-Aware Computing 2017 (PACO2017), Ringberg Castle, Germany, 5-8 July 2017*, 2017, doi:10.5281/zenodo.815496.
17. Lancaster P, Rodman L. *Algebraic Riccati equations*. Oxford Science Publications, The Clarendon Press, Oxford University Press: New York, 1995.
18. Bini DA, Iannazzo B, Meini B. *Numerical solution of algebraic Riccati equations*, *Fundamentals of Algorithms*, vol. 9. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2012, doi:10.1137/1.9781611972092.
19. Benner P, Byers R, Quintana-Ortí E, Quintana-Ortí G. Solving algebraic Riccati equations on parallel computers using Newton's method with exact line search. *Parallel Computing* 2000; **26**(10):1345–1368, doi:10.1016/S0167-8191(00)00012-0.
20. Chu EKw, Fan HY, Lin WW. A structure-preserving doubling algorithm for continuous-time algebraic Riccati equations. *Linear Algebra and its Applications* 2005; **396**:55–80, doi:10.1016/j.laa.2004.10.010.
21. Benner P, Byers R. An exact line search method for solving generalized continuous-time algebraic Riccati equations. *IEEE Trans. Autom. Control.* 1998; **43**(1):101–107, doi:10.1109/9.654908.
22. Benner P, Ezzatti P, Quintana-Ortí E, Remón A. Matrix inversion on CPU-GPU platforms with applications in control theory. *Concurrency and Computat.: Pract. Exper.* 2013; **25**(8):1170–1182, doi:10.1002/cpe.2933.
23. Benner P, Bujanović Z. On the solution of large-scale algebraic Riccati equations by using low-dimensional invariant subspaces. *Linear Algebra and its Applications* 2016; **488**:430–459, doi:10.1016/j.laa.2015.09.027.
24. Li T, Chu EKw, Lin WW, Weng PCY. Solving large-scale continuous-time algebraic Riccati equations by doubling. *Journal of Computational and Applied Mathematics* 2013; **237**(1):373 – 383, doi:10.1016/j.cam.2012.06.006.
25. Benner P, Quintana-Ortí ES, Quintana-Ortí G. Balanced truncation model reduction of large-scale dense systems on parallel computers. *Math. Comput. Model. Dyn. Syst.* 2000; **6**(4):383–405, doi:10.1076/mcmd.6.4.383.3658.