Studying mixed precision techniques for the solution of algebraic Riccati equations

Peter Benner[1] Ernesto Dufrechou[2] Pablo Ezzatti[3] Alfredo Remón[4]

We evaluate different algorithms and the use of a mixed-precision approach for the solution of Algebraic Riccati Equations (AREs). The mixed-precision method obtains an approximation to the solution using single-precision arithmetic and then, this approximation is improved via a cheap iterative refinement. Some numerical results show that the mixed-precision solver reports time and energy savings and also provides similar or even more accurate solutions than well-known methods like the Sign Function or SDA on CPU-GPU platforms.

1 Introduction

We consider the solution of the algebraic Riccati equation (ARE)

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

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 [8], 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.)

The solution of AREs is required in some scientific and engineering applications, e.g., in linear quadratic optimal control (LQOC) and model order reduction problems. It is a computationally intensive operation that 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 ($n > 1,000$). Software packages as MESS[1], PLiC[2] or the MATLAB Control System Toolbox$^{\text{TM}}$ provide support for the solution of AREs.

2 Solution of AREs

A number of methods have been proposed for the solution of AREs (e.g., see [6]). In this section we briefly review two of the more popular, the Sign Function and the Structure-Preserving Doubling Algorithm (SDA) methods. Additionally, we review an iterative refinement scheme that mixes single precision (SP) and double precision (DP) arithmetic computations to get the maximum performance of the underlying hardware.

[1] Max Planck Institute for Dynamics of Complex Technical Systems, 30.106-Magdeburg, Germany, benner@mpi-magdeburg.mpg.de

[2] Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, edufrechou@fing.edu.uy

[3] Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, pezzatti@fing.edu.uy

[4] Max Planck Institute for Dynamics of Complex Technical Systems, 30.106-Magdeburg, Germany, remon@mpi-magdeburg.mpg.de

2.1 The Sign Function method

The solution of an ARE (1) can be defined by the invariant subspaces of the pencil $H - \lambda I_{2n}$, where H is the Hamiltonian matrix 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 [4] can be obtained. This solution can be computed by the Sign Function of H, $\text{sign}(H) = Y = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$, and then solving an overdetermined system (e.g., via the least squares method). The procedure is summarized in Algorithm GECRSG.

Algorithm GECRSG

```
$H_0 := \begin{bmatrix} A & G \\ -Q & -A^T \end{bmatrix}$
for $k = 0, 1, 2, \ldots$ *until convergence*
    $H_{k+1} := \frac{1}{2}\left(H_k + H_k^{-1}\right)$                    ($16n^3$ flops)
Solve $\begin{bmatrix} Y_{11} \\ Y_{12} + I_n \end{bmatrix} X = \begin{bmatrix} I_n - Y_{10} \\ -Y_{00} \end{bmatrix}$    ($13n^3$ flops)
```

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, GECRSG exhibits a remarkable convergence rate that makes it very appealing. The variant here evaluated employs a highly tuned CPU-GPU matrix inversion kernel, see [5] for details.

2.2 The Structure-Preserving Doubling Algorithm (SDA)

In the last years, the SDA has received considerable attention as an ARE solver because of its simplicity, efficiency, and convergence properties [7].

Algorithm 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 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

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

  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 GESDA
$\gamma := \max(1, 2\,\|A\|_F)$
$\hat{A} := A - \gamma I_n$
$\hat{Q} := Q\hat{A}^{-1}$                                  $((2/3 + 2)n^3$ flops)
$\hat{W} := (\hat{A}^T + \hat{Q}G)^{-1}$                    $(4n^3$ flops)
$A_0 := I_n + 2\gamma\hat{W}^T$
$G_0 := 2\gamma(\hat{A}^{-1}G)\hat{W}$                      $((2/3 + 4)n^3$ flops)
$X_0 := 2\gamma\hat{W}\hat{Q}$                              $(2n^3$ flops)
for $k = 0, 1, 2, \ldots$ until convergence
    $\hat{W} := G_k X_k$                                    $(2n^3$ flops)
    $\hat{A} := (I_n + \hat{W})^{-1} A_k$                   $((2/3 + 2)n^3$ flops)
    $Y_k := \hat{A} X_k A_k$                                $(4n^3$ flops)
    $X_{k+1} := X_k + Y_k$
    if not convergence
        $G_{k+1} := G_k + A_k G_k (I_n + \hat{W}^T)^{-1} A_k^T$   $(6n^3$ flops)
        $A_{k+1} := A_k \hat{A}$                            $(2n^3$ flops)
    end if
```

The implementation evaluated in this work executes the most time consuming operations in the GPU while operations that exhibit a fine-grain parallelism are performed in the CPU. Whenever it is possible, both processors concurrently perform their tasks, reporting significant time savings. Finally, the computation of inverses is replaced by the use of the LU factorization of the related matrix.

3 A mixed-precision ARE solver

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

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

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 of a cheap method to compute X_0 and an efficient Lyapunov solver. The initial approximation, X_0, can be efficiently obtained executing some steps of GESDA, which can even be performed using SP arithmetic. This way, the solver benefits from the larger performance that the hardware offers in SP arithmetic computations (Intel CPUs are 2× faster and this factor is larger for NVIDIA GPUs). An economic Lyapunov solver was presented in [5]. The solver implements the Sign Function and relies on a tuned CPU-GPU matrix inversion kernel.

		Mojigata	Hetfield
GPU	Processor	NVIDIA K40 “Kepler” GK110B	NVIDIA TitanX “Maxwell” GM220
	# Cores	2,880	3,072
	Memory	12 GB GDDR5	12 GB GDDR5
CPU	Processor	i7-4770	i7-6700
	# Cores	4	4
	Frequency	3.40 GHz	3.40 GHz
	Main memory	16 GB DDR3	64 GB DDR3
SW	Compiler	`icc` 14.0.0	`icc` 14.0.0
	CUDA Version	6.5	8.0

Table 1: Platforms employed in the evaluation

4 Experimental evaluation

The evaluation focuses on two aspects, the time to solution and the energy consumption. The test-cases evaluated were extracted from the Oberwolfach[5] benchmark collection. In particular, two instances of the steel profile (with $n = 1,357$ and $5,177$) and another from the flow meter problem ($n = 9,669$). Although the three cases permit the use of a low-rank solver, only the Sign Function implementation takes advantage of this feature. Table 1 describes the hardware employed in this evaluation. A remarkable difference between both platforms is that the performance of the GPU in Hetfield is 32× larger when using SP arithmetic than using DP, while this factor reduces to 3× in Mojigata. However, when using hybrid (CPU-GPU) variants these ratios can be smoothed.

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

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

$$\mathrm{RRes} = \left\|\mathcal{R}_c\left(X^*\right)\right\|_F / (\left\|Q\right\|_F + 2\left\|A\right\|_F\left\|X^*\right\|_F + \left\|G\right\|_F\left\|A\right\|_F^2). \quad (3)$$

The results summarized in Tables 2 and 3 show that both solvers behave differently in the two platforms. While in Hetfield the Sign Function solver is clearly faster, Mojigata seems to slightly favor the SDA solver. This behavior is explained by noting that the implementation in SDA is more suitable to the GPU architecture, and the GPU in Mojigata is more powerful (when using DP arithmetic). On the other hand, the Sign Function implementation features a better CPU-GPU load-balance.

		Mojigata			Hetfield			
problem	# steps	sign func.	solver	total	sign func.	solver	total	rel. res.
rail_1357	10	3.63	0.37	4.05	4.78	0.30	5.09	1.51E-17
rail_5177	11	71.81	10.45	82.81	154.52	11.76	166.55	4.96E-17
flow_9669	13	411.34	74.90	488.19	1093.64	72.59	1167.12	2.12E-10

Table 2: Run-times (in sec.) of the Sign Function solver.

For the mixed-precision scheme, we execute the proposal modifying the number of SDA and iterative refinement and steps. At every step of the iterative refinement, a Lyapunov equation is solved via the Sign Function method. Once again we fixed the parameters so that a comparable accuracy is reached. The results, summarized in Table 4, demonstrate

[5] Available at https://portal.uni-freiburg.de/imteksimulation/downloads/benchmark

PROBLEM	# STEPS	MOJIGATA	HETFIELD	REL. RES.
RAIL_1357	24	1.85	6.37	4.96E-16
RAIL_5177	27	75.89	316.67	4.61E-16
FLOW_9669	24	401.79	1805.26	7.99E-12

Table 3: Run-times (in sec.) of the SDA solver.

that the mixed-precision strategy is more effective in HETFIELD, where the GPU exhibits a higher performance in SP arithmetic. But it also outperforms the DP solvers (except for the small case of SDA) on MOJIGATA. Regarding the parametrization of the solver, the experiments show 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.

	#STEPS			MOJIGATA			HETFIELD			
PROBLEM	GESDA	Newton	Lyap.	GESDA	It. ref.	TOTAL	GESDA	It. ref.	TOTAL	Rel. res.
RAIL_1357	10	1	10	0.7	1.5	2.2	0.5	1.9	2.4	1.75E-14
	10	2	8	0.7	2.1	2.9	0.6	2.6	3.2	1.62E-14
	15	1	9	0.9	1.3	2.3	0.6	1.8	2.4	9.10E-15
	15	2	8	0.9	2.2	3.2	0.7	3.0	3.8	1.78E-15
	20	1	9	1.2	1.3	2.5	0.8	2.0	2.8	6.92E-16
	20	2	8	1.1	2.2	3.3	0.9	3.2	4.1	3.70E-16
RAIL_5177	10	1	10	19.8	30.1	50.0	13.0	61.0	74.0	6.53E-15
	10	2	9	19.0	42.4	61.41	12.6	104.3	117.0	4.99E-15
	15	1	10	26.6	23.7	50.08	17.4	61.8	79.1	1.44E-15
	15	2	9	26.4	44.2	70.60	17.3	104.9	122.3	1.00E-15
	20	1	10	33.3	30.2	63.56	21.8	62.0	83.8	7.42E-16
	20	2	9	33.3	39.4	72.66	21.8	105.2	127.0	1.31E-16
FLOW_9669	10	1	7	111.4	113.7	225.12	72.9	291.5	364.4	2.44E-09
	10	2	7	107.6	167.4	275.0	70.7	527.4	598.1	3.94E-13
	15	1	7	149.1	100.4	249.5	99.7	291.7	391.4	2.15E-09
	15	2	7	151.5	164.0	315.5	101.0	529.5	630.6	3.73E-13
	20	1	10	189.1	149.8	339.0	130.5	374.7	505.2	5.39E-10
	20	2	8	178.9	187.0	365.9	129.4	585.3	714.8	7.21E-15

Table 4: Run-times (in sec.) and relative residuals reported by the mixed-precision solver.

In a second experiment, we measure the energy consumption related with the best configuration of each method in MOJIGATA. Tables 5 and 6 show the energy consumption of the three solvers. In the mixed-precision case, the SP and DP stages are distinguished. From the reported results, it can be noticed that the savings in energy consumption of the mixed-precision method is slightly higher than the run-time counterpart. Furthermore, the improvement seems to increase with the dimension of the problem.

SOLVER	PROBLEM	# STEPS	TIME (S)	ENERGY (J)
SIGN FUNC.	RAIL_1357	10	4.0	563.88
	RAIL_5177	11	85.65	14493.7
	FLOW_9669	13	487.29	94516.5
SDA	RAIL_1357	24	2.16	437.2
	RAIL_5177	27	78.12	17730.2
	FLOW_9669	24	467.22	101077.1

Table 5: Energy and run-time evaluation of the double precision solvers.

PROBLEM	#STEPS			GESDA		It. Refinement		Total
	GESDA	Newton	Lyap.	TIME (S)	ENERGY (J)	TIME (S)	ENERGY (J)	Energy (J)
RAIL_1357	10	2	9	0.71	118.3	2.04	309.4	427.7
RAIL_5177	10	2	10	19.55	3342.7	33.72	6554.0	9896.7
FLOW_9669	10	2	7	104.95	16459.0	115.97	24536.0	40995.0

Table 6: Energy and run-time evaluation of the mixed precision solver.

5 Concluding remarks

In this work we presented and evaluated a mixed precision variant for the solution of the Algebraic Riccati Equation. The experimental evaluation showed that the mixed-precision variant offers an interesting reduction on the execution time, compared to traditional methods like the Sign Function and SDA. In addition, the savings in energy consumption reported are even more important than the savings in run-time, what reinforces the convenience of the mixed-precision solver over DP solvers.

As future work, we will study a low rank variant of the mixed precision solver. Specifically, a combination of a low rank variant of the SDA or the Sign Function algorithm to obtain the initial solution and a low rank Lyapunov solver in the iterative refinement.

References

[1] *Matrix Equation Sparse Solver (MESS) library (www. mpi-magdeburg. mpg. de/ projects/ mess/).*

[2] *PLiC library (www3. uji. es/ ~quintana/ plic/ plic/).*

[3] P. Benner and R. Byers, *An exact line search method for solving generalized continuous-time algebraic Riccati equations*, IEEE Trans. Autom. Control., 43 (1998), pp. 101–107.

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

[5] P. Benner, P. Ezzatti, E. Quintana-Ortí, and A. Remón, *Matrix inversion on CPU-GPU platforms with applications in control theory*, Concurrency and Computation: Practice and Experience, 25 (2013), pp. 1170–1182.

[6] D. Bini, B. Iannazzo, and B. Meini, *Numerical Solution of Algebraic Riccati Equations*, Society for Industrial and Applied Mathematics, 2011.

[7] E.-W. Chu, H.-Y. Fan, and W.-W. Lin, *A structure-preserving doubling algorithm for continuous-time algebraic riccati equations*, Linear Algebra and its Applications, 396 (2005), pp. 55–80.

[8] P. Lancaster and L. Rodman, *Algebraic Riccati Equations*, Oxford University Press, 1995.