A COARSE-GRAIN PARALLEL SOLVER FOR PERIODIC RICCATI EQUATIONS

PETER BENNER

Zentrum für Technomathematik, Fachbereich 3/Mathematik und Informatik, Universität Bremen, D-28334 Bremen, Germany; benner@math.uni-bremen.de

RAFAEL MAYO, ENRIQUE S. QUINTANA-ORTí

Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12080–Castellón, Spain; {mayo,quintana}@icc.uji.es

VICENTE HERNÁNDEZ

Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 46071–Valencia, Spain; vhernand@dsic.upv.es

This paper describes the parallelization of a numerical solver for a class of quadratic matrix equations arising in the analysis and design of periodic linear control systems. Experimental results on a cluster of Intel PII processors, connected via a Gigabit switch and a Fast Ethernet switch, report the high performance and scalability of the approach.

1 Introduction

In this paper, we analyse the parallel solution of the discrete-time periodic Riccati equation (DPRE)

$$\begin{aligned} X_k = Q_k &+ A_k^T X_{k+1} A_k \\ &- A_k^T X_{k+1} B_k (R_k + B_k^T X_{k+1} B_k)^{-1} B_k^T X_{k+1} A_k, \end{aligned} \tag{1}$$

where $A_k \in \mathbb{R}^{n\times n}$, $B_k \in \mathbb{R}^{n\times m}$, $C_k \in \mathbb{R}^{r\times n}$, $Q_k \in \mathbb{R}^{n\times n}$, $R_k \in \mathbb{R}^{m\times m}$, and $X_k \in \mathbb{R}^{n\times n}$ is the sought-after solution. The equation is said to be periodic as for some integer $p > 0$, $A_k = A_{k+p}$, $B_k = B_{k+p}$, $C_k = C_{k+p}$, $Q_k = Q_{k+p}$, and $R_k = R_{k+p}$. Under mild conditions, the periodic symmetric positive semidefinite solution X_k is unique [7]. DPREs arise in the analysis and design of periodic linear control systems, e.g., the periodic linear-quadratic optimal control problem, model reduction of periodic linear systems, etc. [7] Periodic linear systems naturally arise when performing multirate sampling of continuous linear systems. Periodic systems with large state-space dimension n and/or large period arise, e.g., in the helicopter ground resonance damping problem (p=41) and the satellite attitude control problem (p=120) [6,12].

The need for parallel computing in this area can be seen from the fact that (1) represents a non-linear system with pn^2 unknowns. Reliable methods for solving these equations have a computational cost of $\mathcal{O}(pn^3)$ flops (floating-point arithmetic operations).

In this paper we analyse the parallelization of a coarse-grain DPRE solver, with a cost of $\mathcal{O}(p^2n^3)$ flops, introduced by Benner and Byers [3,4]. The resulting parallel algorithm only requires efficient point-to-point communication routines and a few high-performance numerical serial kernels for well-known linear algebra computations. A preliminary analysis of the parallel algorithm and basic experimental results using a native implementation of MPI on a Myrinet™ multistage interconnection network (MIN) were reported by the authors [5]. Here we extend these ideas and show that a careful overlapping of computation and communication allows us to obtain a parallel algorithm that can be ported to other commodity interconnection hardware (e.g., a Fast Ethernet switch) while maintaining similar parallel performance. A second DPRE solver with a lower computational cost ($\mathcal{O}(p\log_2(p)n^3)$ flops) is also known [4]. This algorithm presents a slightly irregular communication pattern, but preliminary research shows that a careful treatment of the communication also allows to overlap it with the computation. We do not analyse this second algorithm further in this paper.

In section 2 we briefly review a numerical solver for DPREs based on a reordering of a product of matrices associated with (1). The parallelization of the solver is analysed in section 3. In this section we also describe the use of non-blocking communication routines to overlap communication and computation. In section 4 we report the performance of our approach on a cluster of Intel Pentium-II processors, connected via a Myrinet MIN and a Fast Ethernet switch, using three different communication libraries. Finally, some concluding remarks are given in section 5.

2 Numerical solution of DPREs

Consider the $2n \times 2n$ periodic matrix pairs

$$L_k = \begin{bmatrix} A_k & 0 \\ -Q_k & I_n \end{bmatrix}, \quad M_k = \begin{bmatrix} I_n & B_k R_k^{-1} B_k^T \\ 0 & A_k^T \end{bmatrix}, \quad k = 0, 1, \ldots, p-1,$$

associated with the DPRE (1); here, I_n denotes the identity matrix of order n. If all the A_k are non-singular, the solutions of the DPREs are given by the d-stable invariant subspace of the periodic monodromy matrices [10]

$$\Pi_k = M_{k+p-1}^{-1} L_{k+p-1} M_{k+p-2}^{-1} L_{k+p-2} \cdots M_k^{-1} L_k, \quad k = 0, 1, \ldots, p-1. \quad (2)$$

Note that the monodromy relation still holds if (some of) the A_k are singular and the algorithm presented here can still be applied in that case [4,7,10].

The periodic QZ algorithm is a numerically sound DPRE solver that relies on an extension of the generalized Schur vectors method [10]. This method suffers from the same parallelization difficulties on a distributed parallel computer as the QR algorithm [9]. Although recent attempts have proposed improvements for the parallel QR algorithm [11], the parallelism and scalability of these algorithms are still far from those of traditional matrix factorizations [8].

In this paper we follow a different approach for solving DPREs without explicitly forming the monodromy matrices [3,4]. The approach relies on the following lemma.

Lemma. *Consider $Z, Y \in \mathbb{R}^{n\times n}$, with Y invertible, and let*

$$\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} Y \\ -Z \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{3}$$

be a QR factorization of $[Y^T, -Z^T]^T$; then $Q_{22}^{-1}Q_{21} \;=\; ZY^{-1}$.

The basic idea is to apply repeatedly the swapping procedure to the matrix products of the form $L_{i+1}M_i^{-1}$ in Π_k, $k = 0, 1, \ldots, p-1$, until we obtain reordered monodromy matrices of the form

$$\Pi_k = \hat{M}_k^{-1}\hat{L}_k = (\bar{M}_k \cdots \bar{M}_{k+p-1})^{-1}(\bar{L}_{k+p-1} \cdots \bar{L}_k), \quad k = 0, 1, \ldots, p-1, \tag{4}$$

without computing any explicit inverse. The solutions of the corresponding DPREs are then computed from the appropriate subspaces of the matrix pairs $(\hat{L}_k, \hat{M}_k)$, $k = 0, 1, \ldots, p-1$ [3,4].

Hereafter, we denote by $(Y, Z) \leftarrow swap(Y, Z)$ the application of the lemma to the matrix pair (Y, Z), where Y and Z are overwritten by Q_{22} and Q_{21}, respectively. The algorithm can be stated as follows [3,4]:

```
for k = 0, 1, ..., p-1
    Set L̂_k ← L_k , M̂_{(k+1) mod p} ← M_k
end
for t = 1, 2, ..., p-1
    for k = 0, 1, ..., p-1
        (L_{(k+t) mod p}, M_k) ← swap(L_{(k+t) mod p}, M_k)
        L̂_k ← L_{(k+t) mod p} L̂_k
        M̂_{(k+t+1) mod p} ← M_k M̂_{(k+t+1) mod p}
    end
end
for k = 0, 1, ..., p-1
    Solve the DPRE using (L̂_k, M̂_k)
end
```

The algorithm is only composed of QR factorizations and matrix products. The computational cost of the reordering procedure is $\mathcal{O}(p^2n^3)$ flops and $\mathcal{O}(pn^2)$ for workspace. The costs of the last loop are $\mathcal{O}(pn^3)$ flops and $\mathcal{O}(pn^2)$ for workspace.

3 Parallelization issues

Our coarse-grain parallel algorithm employs a logical linear array of processes, P_0, P_1,..., P_{p-1}, where p is the period of the system. In the algorithm the matrix pair (L_k, M_k) is initially stored in process P_k, $k = 0, 1, \ldots, p-1$. During the swapping stage, each swapping of a matrix pair is carried out locally in a different process. By the end of this stage, a reordered pair $(\hat{L}_k, \hat{M}_k)$, as in (4), is stored in process P_k, and the corresponding solution of the DPRE is computed locally. The parallel algorithm can be stated as follows.

```
In process $P_k$:
    Set $\hat{L}_k \leftarrow L_k$, $\hat{M}_{(k+1) \bmod p} \leftarrow M_k$
    Send $L_k$ to $P_{(k+p-1) \bmod p}$
    Receive $L_{(k+1) \bmod p}$ from $P_{(k+1) \bmod p}$
    for $t = 1, 2, \ldots, p-1$
        $(L_{(k+t) \bmod p}, M_k) \leftarrow swap(L_{(k+t) \bmod p}, M_k)$
        Send $\hat{M}_{(k+t) \bmod p}$ to $P_{(k+p-1) \bmod p}$
        $\hat{L}_k \leftarrow L_{(k+t) \bmod p}\hat{L}_k$
        Receive $\hat{M}_{(k+t+1) \bmod p}$ from $P_{(k+1) \bmod p}$
        Send $L_{(k+t) \bmod p}$ to $P_{(k+p-1) \bmod p}$
        $\hat{M}_{(k+t+1) \bmod p} \leftarrow M_k \hat{M}_{(k+t+1) \bmod p}$
        Receive $L_{(k+t+1) \bmod p}$ from $P_{(k+1) \bmod p}$
    end
    Solve the DPRE using $(\hat{L}_k, \hat{M}_k)$
```

This parallel algorithm requires efficient point-to-point communication routines, high-performance numerical serial kernels for the matrix product and the QR factorization, like those available in BLAS and LAPACK [2], and a serial spectral decomposition algorithm for obtaining the appropriate deflating subspace based, e.g., on the matrix disk function [3].

The algorithm presents a regular and local communication pattern as the only communications necessary are the left circular shifts of $\hat{M}_k$ and L_k.

Assume our system consists of n_p physical processors. In case $n_p > p$, there will be $n_p - p$ idle processors in the system and the speed-up of the algorithm will be bounded by p.

As the user can select the number of processors to use, in order to avoid idle processors, hereafter we assume that $p \geq n_p$. Thus, we map a cluster of $\lfloor (p-r-1)/n_p \rfloor + 1$ neighbour processes onto processor $\mathcal{P}_r$, $r = 0, 1, \ldots, n_p - 1$.

As the communication pattern only involves neighbour processes, the algorithm only requires the communication of n_p matrix pairs between neighbour processors at each iteration of loop t. The remaining $p - n_p$ transferences are actually performed inside a processor, and can be implemented as matrix copies.

Neglect now for a moment the communication time in the algorithm. The theoretical maximum speed-up of the algorithm is given by $S_p = \frac{p}{\lceil p/n_p \rceil}$.

In practice, we will only achieve this theoretical maximum in case we can overlap communication and computation in loop t. The degree of overlap depends on the efficiency of the communication subsystem and the computational performance of the processors. To derive a theoretical model for our parallel algorithm we will use a simplified variant of the "logGP" model [1]; here, we assume that the time required to send a message of length l is given by $\alpha+\beta l$. (Basically, the latency and overhead parameters of the logGP model are combined into α, while no distinction is made between the bandwidth, β^{-1}, for short and long messages.) We also define γ as the time required to perform a flop.

Using a non-blocking (buffered) communication `Send` routine in the algorithm, the transference of $\hat{M}$ needs to be overlapped with the computation of the matrix product $\hat{L}_k \leftarrow L_{(k+t) \bmod p}\hat{L}_k$. Analogously, the transference of L needs to be overlapped with the computation of $\hat{M}_{(k+t+1) \bmod p} \leftarrow M_k\hat{M}_{(k+t+1) \bmod p}$. Therefore, in our algorithm, the maximum speed-up is obtained for $2q^3\gamma \geq \alpha + \beta q^2$, where $q = 2n$ is the size of the matrices in our problem. Usually, $\beta \gg \gamma$ and from a certain "overlapping" threshold $\hat{q}$ communication will be completely overlapped with computation for all $q > \hat{q}$.

Notice that the algorithm is scalable in p as we only need to increase n_p so that p/n_p is constant to keep the performance. However, the algorithm is not scalable in n, as a single processor needs storage space for at least a single pair (L_k, M_k).

4 Experimental results

All our experiments were performed on a cluster of Intel™ Pentium-II processors at 300 MHz, with 128 MB of RAM each, using IEEE double-precision floating-point arithmetic ($\epsilon \approx 2.2 \times 10^{-16}$). An implementation of BLAS specially tuned for this architecture and Linux OS were employed. Performance experiments with the matrix product routine in BLAS (`DGEMM`) achieved 180 Mflops (millions of flops per second) on one processor; that roughly provides a parameter $\gamma \approx 0.0055$ μs.

Table 1. Performance of the communication libraries and hardware.

Library/Hw.	α (μs.)	β^{-1} (Mbps.)	
		Short messages	Long messages
GM/Myrinet	25.9	483.2	592.8
MPI/Myrinet	37.4	213.1	254.4
MPI/Ethernet	161.5	31.2	38.7

The cluster consists of 32 nodes connected by two different networks: a Myrinet MIN and a Fast Ethernet. Myrinet provides 1.28 Gbps, full-duplex links, and employs cut-through (wormhole) packet switching with source-based routing. Our other interconnection network consists of a well-known technology: Fast Ethernet.

We employed basic `Send` and `Receive` communication routines implemented in three different libraries. GM is a native application programming interface by Myricom for communication over Myrinet. We also used the MPI library developed for this architecture by Myricom on top of the GM API (avoids the overhead of using the TCP/IP protocol stack). The communication over the Fast Ethernet was performed using a standard implementation of MPI on top of TCP/IP. Table 1 reports the communication performance of these routines measured using a simple *ping-pong* test, both for short and long messages. The sizes of these messages, 20 and 500 KB respectively, were set to approximate the size of the smallest and the largest problem in our tests ($q = 2n$=50 and 250). The figures in the table for MPI/Ethernet should only be considered as rough approximations. We found a high variability in our experiments in this case due to the operation of the IP and ICMP protocols, contention, etc.

We could expect a parallel implementation based on the communication routines in the GM API to achieve a much higher efficiency. However, as long as communication and computation in our coarse-grain algorithm are completely overlapped, the communication performance will not play an important role in the execution time. From the figures in the table, the theoretical overlapping threshold $\hat{q}$ is at 5, 8, and 45 for GM/Myrinet, MPI/Myrinet, and MPI/Ethernet, respectively.

Figure 1 reports the speed-up of our solvers for problems of period p=8 and 16 using n_p=4, 8, and 16 processors. (Similar results were obtained for 24 and 32 processors.) The figure shows the high parallelism of our approach which obtains speed-ups close to the maximum in almost all cases.

The communication time only contributes to the execution time for the smallest problem sizes reported in the experiment for the MPI/Ethernet and the MPI/GM. We point out, among others, two reasons for the slight deviation of the experimental results and the model. In general, α and β depend on the message length. Furthermore, the computational cost of a flop depends on the problem size and the type of operation; e.g., the so-called Level 3 BLAS operations (basically, matrix product) exploit the hierarchical structure of the memory to achieve a lower γ.

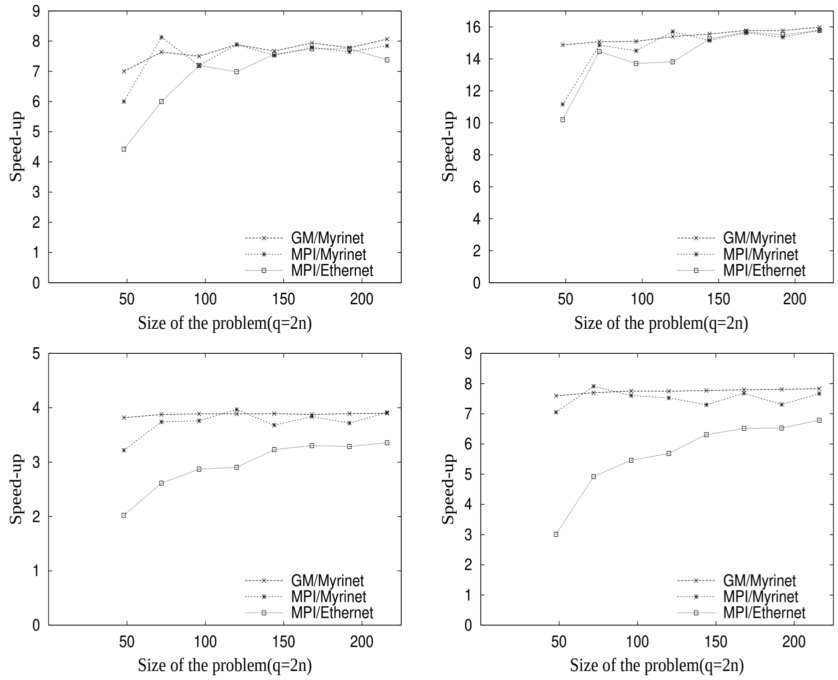

Figure 1. Speed-up of the DPRE solvers for $n_p = p = 8$ (top left), $n_p = p = 16$ (top right) $n_p = 4$, $p = 8$ (bottom left), and $n_p = 8$, $p = 16$ (bottom right).

5 Concluding remarks

We have analysed the parallelization of a numerical solver for DPREs on a cluster of personal computers connected using two different communication networks. The experimental results show that the parallel solution of

DPREs is computation-bounded, and speed-ups close to the maximum are easily achieved.

Acknowledgements

This research was supported by the Consellería de Cultura y Educación de la Generalidad Valenciana GV99-59-1-14, and the DAAD Programme *Acciones Integradas Hispano-Alemanas*.

References

1. A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. *LogGP: Incorporating long messages into the LogP model for parallel computation*, J. Parallel and Distributed Computing, 44(1):71-79, 1997.
2. E. Anderson *et al. LAPACK Users' Guide.* SIAM, Phil., PA, 1994.
3. P. Benner. *Contributions to the Numerical Solution of Algebraic Riccati Equations and Related Eigenvalue Problems.* Dissertation, Fak. f. Mathematik, TU Chemnitz–Zwickau, Chemnitz, FRG, 1997.
4. P. Benner and R. Byers. *Evaluating products of matrix pencils and collapsing matrix products.* To appear in *Numer. Linear Alg. Appl.*, 2000.
5. P. Benner, R. Mayo, E.S. Quintana-Ortí, and V. Hernández. *Solving Discrete-Time Periodic Riccati Equations on a Cluster.* Lecture Notes in Computer Sciences 1900, pp. 824–828. Springer–Verlag, 2000.
6. P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga. SLICOT - a subroutine library in systems and control theory. In B.N. Datta, editor, *Applied and Computational Control, Signals, and Circuits*, volume 1, chapter 10, pages 499–539. Birkhäuser, Boston, MA, 1999.
7. S. Bittanti, P. Colaneri, and G. De Nicolao. The periodic Riccati equation. In S. Bittanti, A.J. Laub, and J.C. Willems, editors, *The Riccati Equation*, pp. 127–162. Springer-Verlag, Berlin, 1991.
8. L.S. Blackford *et al. ScaLAPACK Users' Guide.* SIAM, Phil., PA, 1997.
9. G.H. Golub and C.F. Van Loan. *Matrix Computations.* John Hopkins University Press, Baltimore, 1989.
10. J.J. Hench and A.J. Laub. Numerical solution of the discrete-time periodic Riccati equation. *IEEE Trans. Aut. Cont.*, 39:1197–1210, 1994.
11. G. Henry, D. Watkins, and J. Dongarra. A parallel implementation of the non-symmetric QR algorithm for distributed-memory architectures. Technical Report CS-97-352, University of Tenneesee at Knoxville, 1997.
12. A. Varga and S. Pieters. *A computational approach for optimal periodic output feedback control. Automatica*, 34(4):477–481, 1998.