Strategies for Parallelizing the Solution of Rational Matrix Equations

José M. Badía[1], Peter Benner[2], Maribel Castillo[1], Heike Faßbender[3], Rafael Mayo[1], Enrique S. Quintana-Ortí[1], and Gregorio Quintana-Ortí[1]

[1] Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain; *E-mail:* {*badia,castillo,mayo,quintana,gquintan*}*@icc.uji.es*.

[2] Fakultät für Mathematik, Technische Universität Chemnitz, 09107 Chemnitz, Germany; *E-mail: benner@mathematik.tu-chemnitz.de*.

[3] Technische Universität Braunschweig, Institut *Computational Mathematics*, 38106 Braunschweig, Germany; *E-mail: h.fassbender@tu-bs.de*

In this paper we apply different strategies to parallelize a structure-preserving doubling method for the rational matrix equation $X = Q + LX^{-1}L^T$. Several levels of parallelism are exploited to enhance performance, and standard sequential and parallel linear algebra libraries are used to obtain portable and efficient implementations of the algorithms. Experimental results on a shared-memory multiprocessor show that a coarse-grain approach which combines two MPI processes with a multithreaded implementation of BLAS in general yields the highest performance.

1 Introduction

The nonlinear matrix equation

$$X = Q + LX^{-1}L^T, \tag{1}$$

where $Q \in \mathbb{R}^{n\times n}$ is symmetric positive definite, $L \in \mathbb{R}^{n\times n}$ is nonsingular, and $X \in \mathbb{R}^{n\times n}$ is the sought-after solution, arises in the analysis of stationary Gaussian reciprocal processes over a finite interval. The solution of certain 1-D stochastic boundary-value systems are reciprocal processes. For instance, the steady-state distribution of the temperature along a heated ring or beam subjected to random loads along its length can be modeled in terms of such reciprocal processes[1].

The problem considered here is to find the (unique) largest positive definite symmetric solution X_+ of (1) when n is of large dimension ($n \approx O(1,000) - O(10,000)$). Given the cubic computational cost of numerically reliable solvers[2], the use of parallel computers is necessary for large-scale rational matrix equations (RMEs). Several methods have been proposed for solving (1) in the past[2–4]. Among these, we select the method based on the solution of a discrete-time algebraic Riccati equation (DARE) via a structure-preserving doubling algorithm (SDA), because of its parallel properties.

In this paper we compare different strategies to parallelize the solution of the RME (1) via the SDA. The strategies attack the problem at three levels of granularity, or combinations of those, and rely on different programming models and standard libraries. In many cases, the additional effort required to analyze the parallelism of the problem from different perspectives, so that combining two or more levels of granularity becomes feasible, yields a higher parallel efficiency of the algorithm.

The paper is structured as follows. In Section 2 we review the SDA for the RME (1). In Section 3 we describe several different approaches for the parallelization of the method. The results in Section 4 report the performance of these strategies on a shared-memory multiprocessor consisting of 16 Intel Itanium2 processors. Finally, some concluding remarks follow in Section 5.

2 Numerical Solution of Rational Matrix Equations via a Structure-Preserving Doubling Algorithm

The solution X of

$$X = Q + LX^{-1}L^T$$

satisfies[3]

$$G \begin{bmatrix} I \\ X \end{bmatrix} = H \begin{bmatrix} I \\ X \end{bmatrix} W \tag{2}$$

for some matrix $W \in \mathbb{R}^{n\times n}$, where

$$G = \begin{bmatrix} L^T & 0 \\ -Q & I \end{bmatrix}, \qquad H = \begin{bmatrix} 0 & I \\ L & 0 \end{bmatrix}.$$

Hence, the desired solution X can be computed via an appropriate deflating subspace of $G - \lambda H$. The following idea[4] delivers a fast solver based on these ideas.

Assume that X is the unique symmetric positive definite solution of (1). Then it satisfies (2) with $W = X^{-1}L^T$. Let

$$\widehat{L} = LQ^{-1}L, \quad \widehat{Q} = Q + LQ^{-1}L^T, \quad \widehat{P} = L^TQ^{-1}L,$$

and

$$\widehat{X} = X + \widehat{P}.$$

Then it follows that

$$\widehat{G} \begin{bmatrix} I \\ \widehat{X} \end{bmatrix} = \widehat{H} \begin{bmatrix} I \\ \widehat{X} \end{bmatrix} W^2, \tag{3}$$

where

$$\widehat{G} = \begin{bmatrix} \widehat{L}^T & 0 \\ \widehat{Q} + \widehat{P} & -I \end{bmatrix}, \qquad \widehat{H} = \begin{bmatrix} 0 & I \\ \widehat{L} & 0 \end{bmatrix}.$$

It is easy to see that $\widehat{X}$ satisfies (3) if and only if the equation

$$\widehat{X} = (\widehat{Q} + \widehat{P}) - \widehat{L}\widehat{X}^{-1}\widehat{L}^T \tag{4}$$

has a symmetric positive definite solution $\widehat{X}$.

The doubling algorithm[4] was recently introduced to compute the solution $\widehat{X}$ of (4) using an appropriate doubling transformation for the symplectic pencil (3). Applying this special doubling transformation repeatedly, the SDA in Algorithm 1 is derived[4].

Algorithm 1 SDA

$L_0 = \widehat{L}\,,\ Q_0 = \widehat{Q} + \widehat{P}\,,\ P_0 = 0$
for $i = 0, 1, 2, \ldots$
1. $Q_i - P_i = C_i^T C_i$
2. $A_C = L_i^T C_i^{-1}$
3. $C_A = C_i^{-T} L_i^T$
4. $Q_{i+1} = Q_i - C_A^T C_A$
5. $P_{i+1} = P_i + A_C A_C^T$
6. $L_{i+1} = A_C C_A$

until convergence

As the matrices $Q_i - P_i$, $i = 0, 1, 2, \ldots$ are positive definite[4], the iterations are all well-defined and the sequence Q_{i+1} will converge to $\widehat{X}$. Thus, the unique symmetric positive definite solution to (1) can be obtained by computing

$$X_+ = \widehat{X} - \widehat{P}. \tag{5}$$

The SDA has nice numerical behavior, with a quadratic convergence rate, low computational cost, and fair numerical stability. The algorithm requires about $6.3n^3$ floating-point arithmetic (flops) operations per iteration step when implemented as follows: first a Cholesky factorization $(Q_i - P_i) = C_i^T C_i$ is computed ($\frac{n^3}{3}$ flops), then $L_i^T C_i^{-1}$ and $C_i^{-T} L_i^T$ are solved (each triangular linear system requires n^3 flops), and finally $L_{i+1},\ Q_{i+1},\ P_{i+1}$ are computed using these solutions ($4n^3$ flops if the symmetry of Q_{i+1} and P_{i+1} is exploited). Hence, one iteration step requires $\frac{19}{3}n^3 \approx 6.3n^3$ flops.

The iteration for the SDA is basically composed of traditional dense linear algebra operations such as the Cholesky factorization, solution of triangular linear systems, and matrix products. On serial computers these operations can be efficiently performed using (basic) kernels in BLAS and (more advanced) routines in LAPACK[5].

3 Parallelization

We can parallelize the SDA at different levels (see Figure 1):

- At the highest level, several operations of the sequential algorithm can be executed concurrently. Thus, e.g., the computations of A_C and C_A are independent operations which can be performed simultaneously.

- At an intermediate level, parallel implementations of linear algebra kernels and routines in ScaLAPACK[6] can be used to extract parallelism from each operation. As an example, the parallel routine `pdpotrf` in ScaLAPACK can be used to compute the Cholesky factorization $Q_i - P_i = C_i^T C_i$.

- At the lowest level, multithreaded implementations of BLAS can be used to extract parallelism from the calls to BLAS performed from within each operation, as for example, in invocations to BLAS from the routine for the Cholesky factorization.

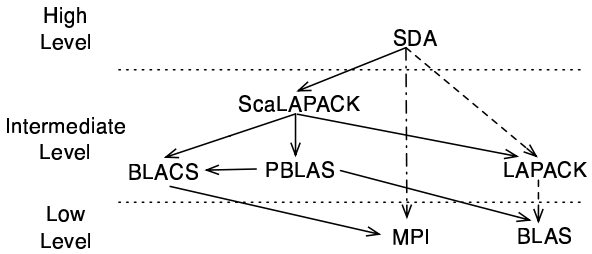

Figure 1. Parallel levels and libraries exploited by the different strategies for the parallelization of the SDA.

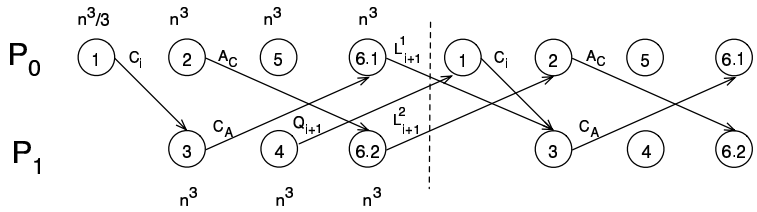

Figure 2. Execution of two consecutive iterations of the 2P parallelization of the SDA by two processes, P_0 and P_1.

Exploiting the two highest levels of parallelism is more natural on platforms with distributed-memory, including multicomputers and clusters. On the other hand, multi-threaded implementations of BLAS are only suited for architectures with shared address space as, e.g., SMPs or multicore processors.

Following these possibilities, we have developed several parallel implementations of the SDA:

- **Message-passing, two processes (2P)**. An analysis of the data dependency graph of the algorithm shows that there are two groups of operations which can be performed concurrently. In this first approach, one MPI process is employed per group.

  Figure 2 illustrates the execution of two consecutive iterations of the SDA (see Algorithm 1) in this approach. The nodes are labeled with their corresponding number of the operation of the SDA, and the computational cost of each operation is included. The arrows represent communication of intermediate results. In order to balance the computational load, we divide the matrix product in step 6 between the two processes. Thus, each process computes half of the columns of the matrix L_{i+1}, which needs then to be communicated to the other process.

  A serial implementation of BLAS and LAPACK is used on each process to perform the different operations of the algorithm.

- **Message-passing multiprocesses (MP)**. We have also developed a message-passing implementation that computes all operations in the SDA using the parallel routines in ScaLAPACK.

- **Multithreading (MT)**. In this case the *parallel* algorithm is reduced to a "serial" code that uses BLAS and LAPACK, and extracts all parallelism from a multithreaded implementation of BLAS.
- **Hybrid (HB)**. Different hybrid versions of the parallel algorithm can be obtained by combining two of the levels of parallelism described earlier. In particular, we have implemented and tested two hybrid algorithms:
  - **Two-process hybrid algorithm (HB2P)**. Each of the two processes in the 2P implementation invokes the kernels from a multithreaded version of BLAS to take advantage of a system with more than two processors.
  - **Multiprocess hybrid algorithm (HBMP)**. Each of the MPI processes executing the ScaLAPACK routines invokes kernels from a multithreaded implementation of BLAS.

4 Numerical Experiments

All the experiments presented in this section were performed on a SGI Altix 350. This is a (CC-NUMA) shared-memory multiprocessor composed of 16 Intel Itanium2 processors running at 1.5 GHz. The processors share 32 GBytes of RAM via a *SGI NUMAlink* interconnect. Two different (multithreaded) implementations of BLAS were used on this platform:

- The SGI *Scientific Computing Software library* (SCSL).
- The *Math Kernel Library* (MKL) 8.1 from Intel.

We have also used the implementation of ScaLAPACK provided by SGI. For simplicity, we only report results for the parallel implementations combined with the optimal implementation of BLAS for each case. All the speed-ups reported next have been measured with respect to an optimized sequential implementation of the SDA.

Figures 3 and 4 show the speed-up of the parallel implementations of the SDA. The results of the HB2P parallel implementation for two processors correspond to those of the 2P algorithm, while the results of the HB2P algorithm for more than two processors correspond to the hybrid version that combines two MPI processes with different numbers of threads. Thus, for example, the results of HB2P on 10 processors involve two processes, each running five threads to exploit a multithreaded implementation of BLAS. Although we have also tested the HBMP parallel implementation, results are not reported for this option as the performance was always lower than those of the MP and MT parallel implementations.

The figures show that, as could be expected, the efficiency (speed-up divided by the number or processors) decreases with the number of processors while the speed-up increases with the equation size. For equations of "small" size, algorithms MP and HB2P obtain similar speed-ups but, as the equation size is increased, the hybrid algorithm outperforms the message-passing version. The difference between these two algorithms remains almost constant regardless of the number of processors. In general, the highest speed-ups are delivered by the algorithm HB2P. On the other hand, the MT algorithm exhibits scalability problems with the number of processors (Figure 3) and, when more than 4 processors

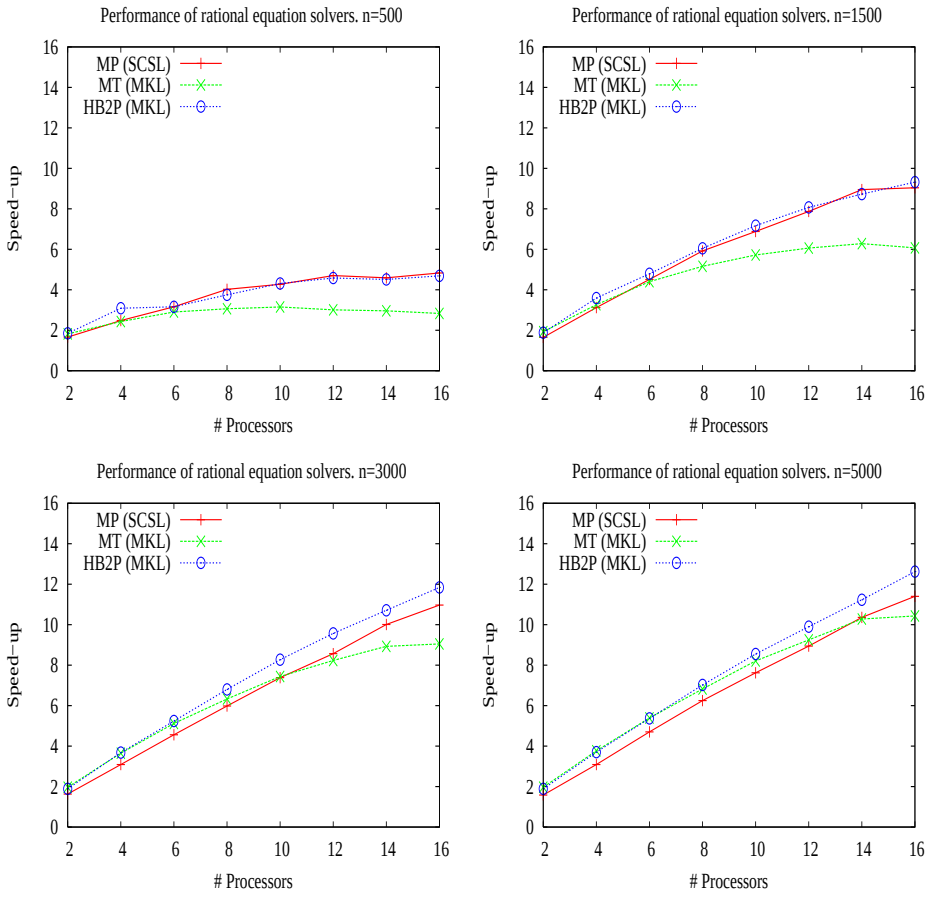

Figure 3. Speed-ups of the parallel implementations MP, MT, 2P (=HB2P on 2 processors), and HB2P of the SDA for n=500, 1500, 3000, 5000, and varying number of processors.

are employed, requires equations of large size to yield a performance comparable to those of the other two approaches (Figure 4).

Possible causes of these results are the following:

- In general, exploiting the parallelism at a higher level identifies more work to be "shared" among the computational resources and should deliver higher performances.

- In the MT algorithm, all threads/processors must access a unique copy of the data (matrices and vectors) in the main memory. Thus, the coherence hardware of this architecture is likely to be stressed to keep track of data that is shared. The overall cost of data transference between cache memories becomes higher as the number of processors is increased, explaining the lower performance of the MT algorithm when the number of processors is large.

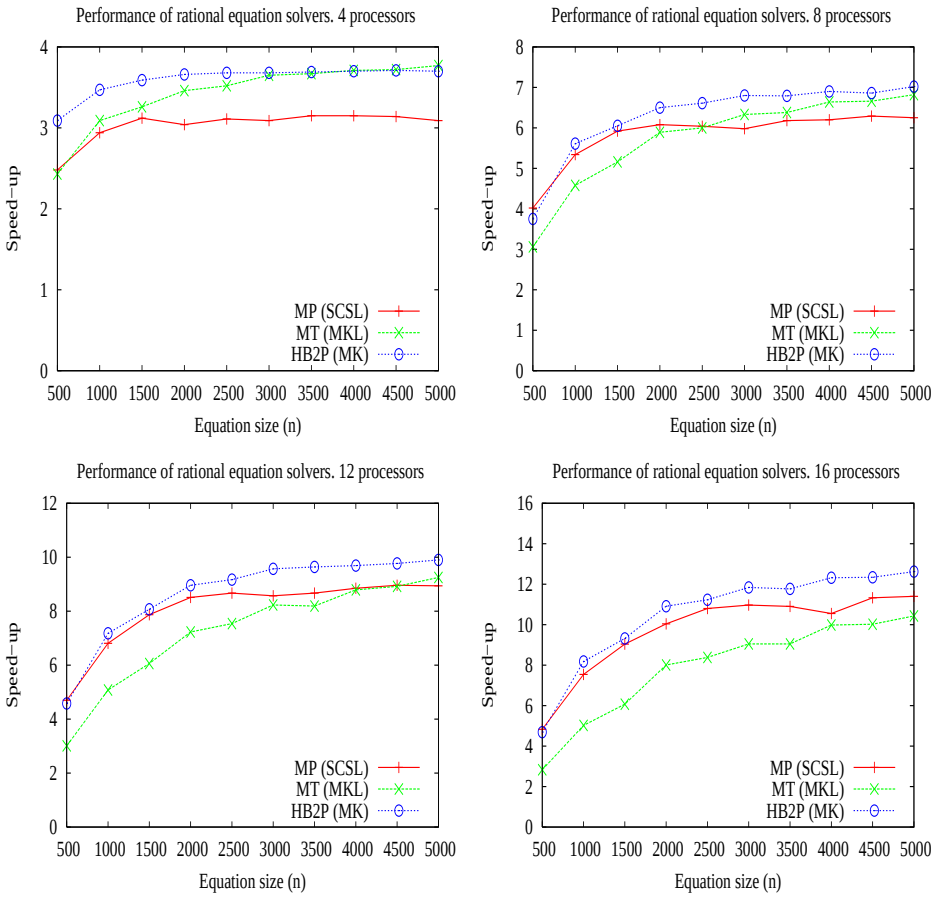

Figure 4. Speed-ups of the parallel implementations MP, MT, 2P (=HB2P on 2 processors), and HB2P of the SDA on 4, 8, 12, 16 processors and varying dimension of the equation.

- In the MP algorithm, the data is partitioned among the processes. As the SGI Altix 350 is a CC-NUMA platform, the data is also likely to be distributed so that data accessed from one process lies in the "local" memory of the corresponding processor. Thus, as most of the accesses performed by the algorithm are satisfied by the local memory, this solution incurs a lower memory access overhead than the MT algorithm when the number of processors is large. On the other hand, when the number of processors is small, sharing data between threads, as in the MT algorithm, is more efficient than doing so via processes, as in the MP algorithm.

- The hybrid algorithm increases the data locality of memory accesses performed by the threads since, in this approach, the amount of data shared by each thread group is roughly halved with respect to the MT algorithm.

5 Conclusions

In this paper we have implemented five different parallel strategies to solve the RME (1) via a structure-preserving doubling algorithm. The algorithms pursue the solution of the equation at different levels of parallelization: at the highest level we have used MPI processes to execute the different operations of the sequential algorithm concurrently. At the lowest level we have exploited a multithreaded version of the BLAS library to execute each basic linear algebra kernel in parallel.

Experiments on a shared-memory multiprocessor show that the best results are obtained with an hybrid parallel algorithm that combines MPI processes with threads. This algorithm obtains high speed-ups and scales linearly up to 16 processors. High performance is also obtained with the multiprocess version of the algorithm which employs only routines from ScaLAPACK. These results confirm that parallelizing the solution of the problem at different levels is worth the effort.

All the parallel algorithms are implemented using standard sequential and parallel linear algebra libraries and MPI, ensuring the portability of the codes.

Acknowledgments

José M. Badía, Maribel Castillo, Rafael Mayo, Enrique S. Quintana-Ortí, and Gregorio Quintana-Ortí were supported by the CICYT project TIN2005-09037-C02-02 and FEDER. These authors and Peter Benner were also supported by the DAAD Acciones Integradas project HA2005-0081 (Spain), D/05/25675 (Germany).

References

1. B.C. Levy, R. Frezza, and A.J. Kerner. Modeling and estimation of discrete-time Gaussian reciprocal processes. *IEEE Trans. Automat. Control*, AC(90):1013–1023, 1990.
2. P. Benner and H. Faßbender. On the solution of the rational matrix equation $X = Q + LX^{-1}L^T$. *EURASIP J. Adv. Signal Proc.*, Article ID 21850, 10 pp., 2007.
3. A. Ferrante and B.B. Levy. Hermitian solutions of the equation $X = Q + NX^{-1}N^*$. *Linear Algebra Appl.*, 247:359–373, 1996.
4. W.-W Lin and S.-F Xu. Convergence analysis of structure-preserving doubling algorithms for Riccati-type matrix equations. *SIAM J. Matrix Anal. Appl.*, 28:26–39, 2006.
5. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.
6. L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997.