Theoretical Computer Science 412 (2011) 1484–1491

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Parallel QR processing of Generalized Sylvester matrices

M. Kourniotis *, M. Mitrouli, D. Triantafyllou

Department of Mathematics, University of Athens, Panepistimioplois, 15784 Athens, Greece

ARTICLE INFO

Keywords:
Generalized Sylvester
Parallel QR factorization
Rank
ScaLapack
Numerical methods

ABSTRACT

In this paper, we develop a parallel QR factorization for the generalized Sylvester matrix. We also propose a significant faster evaluation of the QR applied to a modified version of the initial matrix. This decomposition reveals useful information such as the rank of the matrix and the greatest common divisor of the polynomials formed from its coefficients. We explicitly demonstrate the parallel implementation of the proposed methods and compare them with the serial ones. Numerical experiments are also presented showing the speed of the parallel algorithms.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

In image processing, image convolution (blurring) and deconvolution (deblurring) [6], it is needed to depict a picture to a matrix. More specifically, to every pixel of a digital picture corresponds three integers according to the depth of the red, green and blue (RGB). From these integers, polynomials can be constructed, the degree of which depends on the megapixels of the initial image. The matrix of an 640×480 picture corresponds to 640 univariate polynomials of degree 479 each. From the coefficients of these polynomials is formed the generalized Sylvester matrix (GSM), the size of which can be significantly big. For a standard picture can be up to $409\,600 \times 1120$. Appropriate processing of the GSM, such as the specification of its QR factorization, can produce useful information about the initial polynomial set concerning the determination of their greatest common divisor (GCD). This is necessary for the deblurring of the image. When GSM of large sizes are encountered, their QR factorization demands many flops, which frequently makes the whole process inefficient. In these cases, the use of serial programming is rather discouraging. The parallel evaluation of the procedures can lead to fast and accurate algorithms. The general idea of the parallel QR factorization of GSM is connected with the cooperation of many processors through a process grid on which the initial matrix is distributed, in order to be transformed into an upper triangular form R. In every level of the algorithm, communications between the processors, data's broadcast and local computations are used in order to achieve the best combination of the time processing of the processors and the communication between them.

The following definitions [7] are used to evaluate the performance of a parallel algorithm. Let us suppose that we want to apply a parallel processing of data using P processors.

Definition 1. The speedup of a parallel implementation is defined as

$$S_P = \frac{T_1}{T_P},$$

where T_1 is the execution time of the sequential algorithm and T_P is the execution time of the parallel algorithm.

Speedup refers to how faster a parallel algorithm is in comparison with the corresponding sequential one. When $S_P = P$, we have the so-called linear speedup or ideal speedup. When running an algorithm with linear speedup, doubling the number of processors doubles the speed. This is considered to be a very good scalability [9,12].

* Corresponding author. Tel.: +30 210 7276390; fax: +30 210 7276398.
E-mail addresses: mkournio@yahoo.com (M. Kourniotis), mmitroul@math.uoa.gr (M. Mitrouli), dtriant@math.uoa.gr (D. Triantafyllou).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.11.051

Definition 2. The efficiency of a parallel implementation is defined as

$$E_P = \frac{1}{P} \cdot \frac{T_1}{T_P} = \frac{S_P}{P}.$$

Efficiency is a performance metric with values between 0 and 1, estimating how well utilized the processors are in solving the problem, compared to the effort wasted in communication and synchronization. Algorithms with linear speedup and algorithms running on a single processor have an efficiency of 1.

The paper is organized as follows. In Section 2, we give a brief review of the GSM and we present the block QR algorithm. We describe the parallel QR factorization as executed through scaLapack and we apply it to classical generalized Sylvester matrices (CGSM). In Section 3, we analyse the parallel QR factorization for a modified generalized Sylvester matrix (MGSM). This parallel implementation reduces significantly the required floating point operations and results to a fast and stable algorithm. Numerical results showing the speedup in the performance of the method are also given. In Section 4, a comparison of the required execution time of all proposed methods is described. Concluding remarks concerning the serial and parallel implementation of the algorithms are also presented.

2. Parallel and serial QR implementation of classical generalized Sylvester matrices (CGSM)

In this section, we present the parallel triangularization of CGSM.

2.1. Classical generalized Sylvester matrix

Let

$$a(s) = a_n s^n + a_{n-1} s^{n-2} + a_1 s^{n-3} + \cdots + a_0,$$
$$b_i(s) = b_{i,p} s^p + b_{i,p-1} s^{p-1} + b_{i,p-2} s^{p-2} + \cdots + b_{i,0}, \quad i = 1, 2, \ldots, m,$$

be $m + 1$ polynomials of maximal degrees n and p, with $p \leq n$. From their coefficients, we form

$$S_0 = \begin{bmatrix} a_n & a_{n-1} & a_{n-2} & \ldots & a_0 & 0 & \ldots & 0 & 0 \\ 0 & a_n & a_{n-1} & \ldots & a_0 & 0 & \ldots & 0 & 0 \\ . & . & . & \ldots & . & . & \ldots & . & . \\ 0 & 0 & 0 & a_n & \ldots & . & . & a_1 & a_0 \end{bmatrix}$$

and

$$S_i = \begin{bmatrix} b_{i,p} & b_{i,p-1} & \ldots & b_{i,0} & 0 & \ldots & 0 \\ 0 & b_{i,p} & \ldots & b_{i,1} & b_{i,0} & \ldots & 0 \\ . & . & . & \ldots & . & \ldots & . \\ 0 & 0 & 0 & b_{i,p} & \ldots & . & b_{i,0} \end{bmatrix},$$

where S_0 is an $p \times (n + p)$ matrix corresponding to $a(s)$ and S_i an $n \times (n + p)$ matrix corresponding to $b_i(s)$, $i = 1, 2, \ldots m$.

The CGSM [1] is an $(mn + p) \times (n + p)$ matrix defined by

$$S = \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_m \end{bmatrix}.$$

In several applications the upper triangularization of S is needed. This can be achieved using the parallel QR factorization.

Theorem 1 (*QR Factorization of the Classical Generalized Sylvester Matrix (CGSM)*)**.** *Let $S = [\underline{s}_1, \underline{s}_2, \ldots, \underline{s}_{n+p}] \in \mathbb{R}^{(mn+p)\times(n+p)}$ be a CGSM. Then there always exists an orthogonal matrix $Q = [\underline{q}_1, \underline{q}_2, \ldots, \underline{q}_{mn+p}] \in \mathbb{R}^{(mn+p)\times(mn+p)}$ and an upper triangular matrix $R \in \mathbb{R}^{(mn+p)\times(n+p)}$, such that*

$$S = QR$$

and

$$span\{\underline{s}_1, \underline{s}_2, \ldots, \underline{s}_k\} = span\{\underline{q}_1, \underline{q}_2, \ldots, \underline{q}_k\}, \quad k = 1, \ldots, n + p$$

where $Q = H_1 * H_2 * \cdots * H_{n+p}$, H_i the ith Householder reflection matrix and $\underline{s}_i$, $\underline{q}_i$ the ith column of S and Q, respectively. If Q_1 is the left $(mn + p) \times (n + p)$ part of Q, then $range(S) = range(Q_1)$, $S = Q_1 R_1$ and $rank(S) = rank(R_1)$, where R_1 is the upper $(n + p) \times (n + p)$ part of R [3].

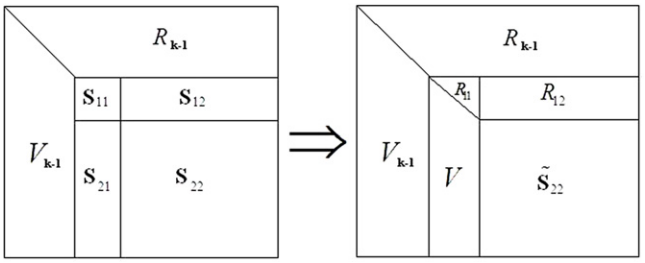

Fig. 1. The block QR algorithm.

Implementation

If we apply to S the QR-factorization, the number of non-zero rows of R gives the rank of the matrix and the last $m - r$ columns of Q form its null space. The required flops for the serial QR-factorization of S, applied directly to the whole of it, are $(n+p)^2(mn+p-\frac{n+p}{3})$. In order to simplify the previous formula, we set n, m, p to be of order of n. Then, the required complexity of the above classical procedure is $O(n^4)$ flops, which means that the method is inefficient [3].

2.2. The block QR algorithm

Let S be an $(mn+p)\times(n+p)$ CGSM. Our aim is to find its parallel QR factorization $S = QR$, where Q is $(mn+p)\times(mn+p)$ orthogonal and R is $(mn+p)\times(n+p)$ upper triangular matrix. Throughout the process, S will supposed to be block partitioned.

In first step $(k = 1)$, we separate the initial matrix of size $m_1 \times n_1 = (mn+p)\times(n+p)$ to blocks of size $m_b \times n_b$, with $m_b \geq n_b$. We triangularize the upper left block and we update the others.

In the second step, the size of the matrix will be $m_2 \times n_2 = (m_1 - n_b)\times(n_1 - n_b)$, in the third step $m_3 \times n_3 = (m_2 - 2\cdot n_b)\times(n_2 - 2\cdot n_b)$ and so on.

In the kth step of the algorithm, we form the factorization of the submatrix $S_k \in \mathbb{R}^{m_k \times n_k}$, where $m_k = m_1 - k\cdot n_b$, $n_k = n_1 - k\cdot n_b$, as follows:

$$S_k = \begin{bmatrix} S_1 & S_2 \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} = Q_k \cdot \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix},$$

where
$S_{11} = n_b \times n_b$,
$S_{12} = n_b \times (n_k - n_b)$,
$S_{21} = (m_k - n_b)\times n_b$,
$S_{22} = (m_k - n_b)\times(n_k - n_b)$
$Q_k = \prod_{i=1}^{n_b} H_i$, H_i Householder matrices of the form $H_i = I - t_i v_i v_i^T$, v_i the Householder vector with $i-1$ zero entries, $t_i = \frac{2}{v_i^T v_i}$ scalar.

In matrix form we have, $Q = H_1 \cdot H_2 \cdot \ldots \cdot H_{n_b} = I - V\cdot T\cdot V^T$, where T is $n_b \times n_b$ upper triangular and V is $m_k \times n_b$, whose ith column contains the vector v_i. This matrix is saved in the zero part of S_1. The updating procedure of S_2 is as follows:

$$\tilde{S}_2 = \begin{bmatrix} \tilde{S}_{12} \\ \tilde{S}_{22} \end{bmatrix} \leftarrow \begin{bmatrix} R_{12} \\ R_{22} \end{bmatrix} = (Q_k)^T \cdot S_2 = (I - V\cdot T^T \cdot V^T)\cdot S_2.$$

This step is described in Fig. 1.

The procedure of triangularizing the matrix S ends when k reaches a value j for which it holds $n_b \geq m_j$.

The implementation

We briefly describe the high level implementation of the parallel QR factorization of S_k. We assume that S is distributed according to the two-dimensional block circled pattern over a process grid $C \times D$. More specifically, each block row/column of the partitioned matrix lays in a cyclic way on a process row/column of the process grid, respectively.

- Perform in parallel:
  - Triangularize S_{11} and compute matrix V
  - Compute the triangular factor T
  - Update S_2

followed by broadcasts among the process rows and columns and local matrix-to-matrix multiplications.

Table 1
Parallel times of QR factorization for square matrices 10 000 × 10 000 on Uranus HP.

Block size	Process grid		
	4 × 2	3 × 2	2 × 2
10	348.50	443.30	706.53
25	220.32	288.81	423.91
50	167.18	214.18	337.77
80	154.54	205.96	306.03
100	160.59	204.06	312.68
250	203.92	273.16	389.47
500	280.96	363.69	508.66

Table 2
Parallel and serial times of QR factorization for square matrices on Uranus HP.

Matrix size	Serial	Parallel	S_P	E_P
10 000	1201	154.5	7.77	0.971
5 000	148.37	19.81	7.49	0.936
2 000	9.511	1.74	5.47	0.68
1 000	1.28	0.27	4.74	0.59
500	0.19	0.077	2.47	0.30

Table 3
Parallel and serial QR factorization for CGSM on Aegean HP V2600.

n	m	p	$(mn+p)\times(n+p)$	Serial	Parallel	S_P	E_P
100	100	100	10 100 × 200	6.23	0.93	6.69	0.84
200	100	200	20 200 × 400	31.37	7.40	4.23	0.52
200	100	300	30 200 × 500	64.17	17.30	3.70	0.46
400	100	400	40 400 × 800	177.97	37.77	4.71	0.59
500	100	500	50 500 × 1000	320.58	68.77	4.66	0.58
600	100	600	60 600 × 1200	527.35	103.38	5.10	0.63

Accuracy

The factorization computed with the parallel routine is the exact factorization of the matrix $S + E$ with

$$\|E\|_2 = \epsilon \cdot p(m, n) \cdot \|S\|_2 ,$$

where ϵ is the machine precision and $p(m, n)$ is a modest function of m and n [5].

The introduced relative error (*Rel*) is

$$Rel = \frac{\|S - (S + E)\|_2}{\|S\|_2} = \frac{\|E\|_2}{\|S\|_2} \leq \epsilon \cdot p(m, n).$$

Thus, it is always stable as it holds for the serial QR factorization. To reduce possible ill conditioning of the upper triangular part, we can include a kind of smooth scaling as the one proposed in [11].

Numerical examples

Next some results concerning the implementation and the speed of the QR factorization for CGSM are presented.

Some parameters of the algorithm

Scalapack's parallel routine PDGEQRF [8] performs the block QR algorithm described above. It applies a two-dimensional block cyclic decomposition. More specifically, it splits the matrix in equal size blocks and distributes them on a process grid $C \times D$ that consists of $C * D$ processors.

Initially, we implemented the parallel QR routine on square matrices of order 10 000 using 8, 6 and 4 processors, selecting each time various block sizes. The time results (in seconds) are presented in Table 1. From the attained results, is found that the best choice for faster parallel-time implementation is the use of a 4 × 2 process grid consisting of 8 processors with partition block size $m_b = n_b = 80$. In Table 2, we used square matrices of size 500–10 000 whereas, in Table 3, we tested CGSM up to sizes 60 600 × 1200 formed by polynomials with random selected coefficients. For these examples, we executed the respective serial and parallel block QR factorizations (with Lapack's routine DGEQRF [2]) comparing them according to their speedup and efficiency. For this parallel implementation, we used the system Uranus (HP Superdome) and a second parallel system Aegean (HP V2600) from the Computer Center at the University of Athens.

In Fig. 2, the required time for serial and parallel implementation of the CGSM given in Table 3 is presented.

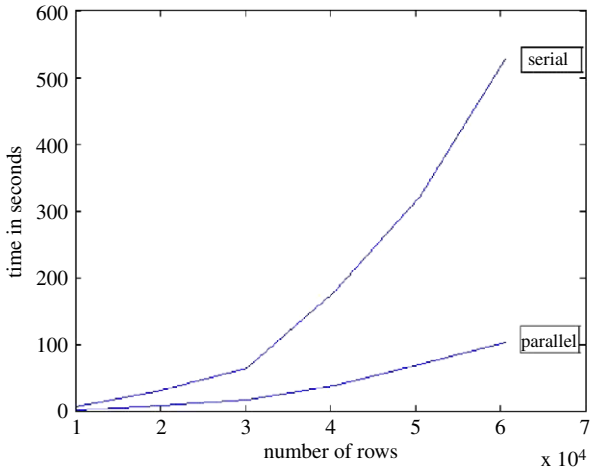

Fig. 2. CGSM: serial vs. parallel.

3. Parallel QR implementation of modified generalized Sylvester matrix (MGSM)

Next, we describe a modification of the CGSM [10] and we propose its parallel QR factorization.

We reorder the rows of CGSM by collecting the jth row of every block S_i, $i = 1, \ldots, m$ constructing in this way n same blocks B. We move the initial block S_0 to the bottom. The resulting matrix is called MGSM, is denoted by S^* and has the following form:

$$S = \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_m \end{bmatrix} \to S^* = \begin{bmatrix} B\underline{00} & \mathbb{O} & & \\ \underline{0}B\underline{0} & \mathbb{O} & & \\ \underline{00}B & \mathbb{O} & & \\ & \ddots & \ddots & \\ & \mathbb{O} & & \underline{0}B \\ & S_0 & & \end{bmatrix},$$

where $\mathbb{O}$ is a zero matrix, $\underline{0}$ is a zero column vector, $\underline{00}$ are two zero column vectors and so on. We now start processing S^* according to the following steps:

Step 1. We triangularize the submatrix B using the parallel QR factorization described in the previous section. B is partitioned in blocks that are distributed according to the two-dimensional block decomposition on a process grid $C \times D$. We calculate in parallel the first QR factorization of submatrix B ($B = Q_1 R_1$). Next B is replaced by R_1, which will be of upper triangular or trapezoidal form according to the number and the degrees of the given polynomials. In this way, the QR factorization is performed only once causing a significant reduction to the required time. This results to

$$\begin{bmatrix} Q_1^T & \mathbb{O} & \mathbb{O} & \ldots & \mathbb{O} & \mathbb{O} \\ \mathbb{O} & Q_1^T & \mathbb{O} & \ldots & \mathbb{O} & \mathbb{O} \\ \mathbb{O} & \mathbb{O} & Q_1^T & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \mathbb{O} & \mathbb{O} & \mathbb{O} & \ldots & Q_1^T & \mathbb{O} \\ \mathbb{O} & \mathbb{O} & \mathbb{O} & \ldots & \mathbb{O} & I \end{bmatrix} \cdot \begin{bmatrix} B\underline{00} & \mathbb{O} & & \\ \underline{0}B\underline{0} & \mathbb{O} & & \\ \underline{00}B & \mathbb{O} & & \\ & \ddots & \ddots & \\ & \mathbb{O} & & \underline{0}B \\ & S_0 & & \end{bmatrix} \to \begin{bmatrix} R_1\underline{00} & \mathbb{O} & & \\ \underline{0}R_1\underline{0} & \mathbb{O} & & \\ \underline{00}R_1 & \mathbb{O} & & \\ & \ddots & \ddots & \\ & \mathbb{O} & & \underline{0}R_1 \\ & S_0 & & \end{bmatrix} = (S^*)^{(1)}.$$

The remaining same blocks are updated with R_1 without any other calculations.

Step 2. We triangularize in parallel the two first blocks $\begin{bmatrix} R_1\underline{0} \\ \underline{0}R_1 \end{bmatrix} =: B_1$ of $(S^*)^{(1)}$. Next, we update again the entries of the remaining same blocks without making any other calculations.

If the number of same blocks B is odd, we move the last one under S_0 and the matrix has the following form:

$$\begin{bmatrix} B_1\underline{00} & \mathbb{O} & \\ \underline{00}B_1 & \mathbb{O} & \\ & \ddots & \ddots \\ & \mathbb{O} & B_1 \\ & S_0 & \\ & \mathbb{O} & \underline{0}R_1 \end{bmatrix} = (S_M^*)^{(1)}.$$

Step i, $i = 3, \ldots, k-1$. We triangularize in parallel the two first same blocks of $(S^*)^{(i)}$ ($(S^*)^{(i)}$ is the updated MGSM in the ith step of the factorization) or $(S_M^*)^{(i)}$ and we update the remaining blocks without making any other calculations, since the blocks are the same. The number of the same blocks are reducing per half in every step since the number of their rows are doubled. If the number of the same blocks R_1 is odd, the last one will not be updated and will be moved to the bottom of S^*. Because the number of the same blocks is initially equal to n and this number is reduced per half in every step, the maximum number of the needed parallel QR factorizations applied to the first two same blocks every time is at most $[\log_2(n)]$. We continue triangularizing the blocks until the same blocks in the $(S^*)^{(k)}$ be three or less, in some step k, where $k \leq [\log_2(n)]$.

Step k. The modified matrix in this step will have the following form:

$$\begin{bmatrix} R_k & & & \\ R_k & & & \\ R_k & & & \\ & S_0 & & \\ & & R_{i_1} & \\ & & \cdots & \\ & & R_{i_\rho} & \end{bmatrix} = (S_M^*)^{(1)},$$

where ρ is the number of steps where odd blocks were appearing each time and the last of them was moved to the bottom of the matrix. The number of R_k's is at most three. We triangularize the three (or less) R_k's using Givens transformations, zeroing specific elements which are below the main diagonal or implementing in parallel the QR factorization only for the rows whose first non-zero element is below the main diagonal. We get the following form:

$$\begin{bmatrix} R & \\ S_0 & \\ & R_{i_1} \\ & \cdots \\ & R_{i_\rho} \end{bmatrix},$$

where R is the upper triangular, S_0 is a parallelogram and R_{i_j}'s, $j = 1, \ldots, \rho$ are upper triangular.

Step k + 1. We apply one more time in parallel the QR factorization only for the rows whose first non-zero element is below the main diagonal. Using R we zero only the entries of S_0 and R_{i_j}'s, $j = 1, \ldots, \rho$, which are below the main diagonal and we update the corresponding rows.

These steps result to the following factorization.

Theorem 2 (*Skinny QR factorization of the MGSM [10]*)**.** *Let*

$$S^* = \begin{bmatrix} B\underline{00} & \mathbb{O} & & \\ \underline{0B0} & \mathbb{O} & & \\ \underline{00B} & \mathbb{O} & & \\ & \ddots & \ddots & \\ & \mathbb{O} & & \underline{0B} \\ & S_0 & & \end{bmatrix}$$

be the MGSM written in block form. There exists orthogonal matrix $Q \in \mathbb{R}^{(mn+p)\times(mn+p)}$ *such that* $QS^* = R$, *where* $R = \begin{bmatrix} R_1 \\ \mathbb{O} \end{bmatrix} \in \mathbb{R}^{(mn+p)\times(n+p)}$ *and* $R_1 \in \mathbb{R}^{(n+p)\times(n+p)}$, *upper triangular.*

Numerical results

Next some results concerning the parallel QR factorization for MGSM and their comparison with the parallel factorization for CGSM are presented.

Implementation

We applied in parallel the procedure described above for MGSM using the parameters from the previous section. In Table 4, we present the required time (in seconds) for the parallel and the serial execution for triangularizing a MGSM.

In Fig. 3, the required time for serial and parallel implementation of the MGSM given in Table 4 is presented.

Comparison of serial and parallel implementation

The serial evaluation of the proposed in [10] factorization of MGSM requires $O(16\log_2(n)\cdot n^3)$ flops, which means that the complexity is one order less than the $O((n+p)^2(mn+p-\frac{n+p}{3}))$ flops of the classical procedure. If an algorithm based on displacement structure is used, in the case where $m = n = p$ are required $O(r\cdot n^3)$ flops, where r is a demanded number of steps, in order to obtain a proper generator using the partial triangularization method described in [4]. For our algorithm, this value of r is bounded from $16\log_2(n)$.

Table 4
Parallel and serial QR factorization for MGSM.

n	m	p	$(mn+p)\times(n+p)$	Serial	Parallel	S_P	E_P
100	100	100	10 100 × 200	0.68	0.15	4.53	0.57
200	100	200	20 200 × 400	1.79	0.48	3.72	0.47
200	100	300	30 200 × 500	2.64	0.63	4.19	0.52
400	100	400	40 400 × 800	11.22	1.68	6.88	0.84
500	100	500	50 500 × 1000	25.94	3.59	7.23	0.90
600	100	600	60 600 × 1200	39.1	5.19	7.53	0.94

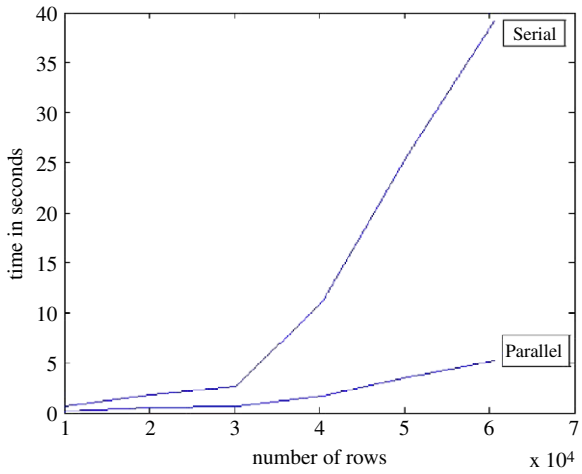

Fig. 3. MGSM: serial vs. parallel.

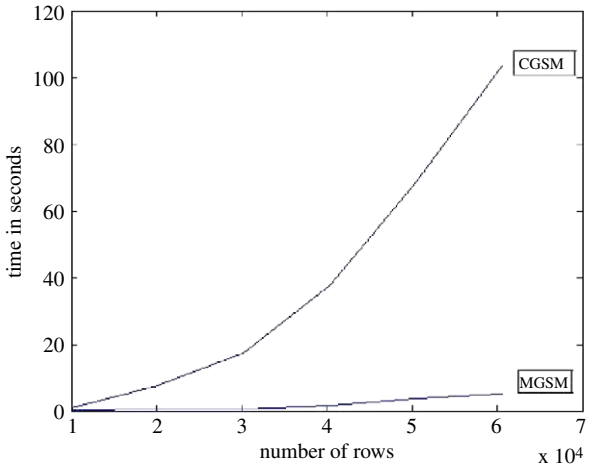

Fig. 4. Parallel implementation: CGSM vs. MGSM.

Applying in parallel the factorization of CGSM, it seems to be quite faster than the serial one. The parallel factorization of MGSM is significantly more efficient than the serial one. For matrices of small order this is not so evident, but as we proceed to matrices of larger orders the efficiency can increase up to 94%.

In Fig. 4, the required time (in seconds) for the parallel factorization of CGSM and MGSM given in Tables 3 and 4 are presented.

From this figure, it is clear that the parallel QR factorization of MGSM is significantly more efficient than that of CGSM.

4. Conclusions

In this paper, we presented the parallel QR factorization for CGSM. We also introduced a parallel QR factorization applied to the MGSM. The parallel implementation of the QR factorization of CGSM or MGSM is significantly faster than the serial one. The main benefit of our parallel algorithm is that it takes advantage of the special form of MGSM and performs parallel

QR factorizations to specific blocks of seriously smaller sizes than those of the initial matrix, causing an important reduction to the required computations. Another main advantage of our procedure is that it exploits the special form of MGSM and handles in parallel not all the blocks of the initial matrix but only one at every step, updating simultaneously the others without any additional floating point operations. This is the basic reason for the significant reduction of the required time that MGSM attains versus the classical one. In addition, parallel MGSM achieves a very good scalability since its speedup becomes almost linear as the sizes of the matrices increase.

Concerning a comparison between CGSM and MGSM as we can see from the times given in Tables 3 and 4, the parallel factorization of MGSM is significantly faster than that of CGSM. We must notice that the serial factorization of MGSM is even faster than the parallel factorization CGSM establishing the superiority of MGSM. The difference in time between them becomes larger as the size of the initial matrices become larger too. In Table 4, we see that the parallel factorization of MGSM is about 20 times faster than the parallel CGSM and about 100 times faster than the serial one.

Acknowledgements

We would like to thank the referees for their valuable comments and suggestions that contributed to a significant improvement of the presentation of this paper.

The authors are grateful to Dr. M. Drakopoulos for providing a lot of useful information by personal communication.

An allocation of CPU time from the Computer Center at the University of Athens is gratefully acknowledged.

This research was financial supported by Kapodistrias 70/4/5760 of NKUA/SARG.

References

[1] S. Barnett, Greatest common divisor from generalized Sylvester resultant matrices, Linear and Multilinear Algebra 8 (1980) 271–279.
[2] J. Choi, J.J. Dongarra, S. Ostrouchov, A.P. Peetitet, D.W. Walker, R Clint Whaley/The Design and Implementation of the ScaLapack LU, QR, and Cholesky Factorization Routines.
[3] B.N. Datta, Numerical Linear Algebra and Applications, SIAM Publications, Philadelphia, 2010.
[4] T. Kailath, J. Chun, Generalized displacement structure for block-Toeplitz, Toeplitz-block and Toeplitz-derived matrices, SIAM Journal on Matrix Analysis and Applications 15 (1) (1994) 114–128.
[5] NAG Parallel Library, http://www.nag.co.uk/numeric/FD/manual/pdf/F08/f08aefp_fd03.pdf.
[6] S.U. Pillai, B. Liang, Blind image deconvolution using a robust GCD approach, IEEE Transactions on Image Processing 8 (2) (1999) 295–301.
[7] S. Ragsdale, Parallel Programming, McGraw-Hill, 1991, pp. 20–21.
[8] The Scalapack Users Guide, www.netlib.org.
[9] X.H. Sun, J.L. Gustafson, Toward a better parallel performance metric, Parallel Computing 17 (1991) 1093–1109.
[10] D. Triantafyllou, M. Mitrouli, On rank and null space computation of the generalized Sylvester matrix, Numerical Algorithms 54 (2010) 297–324.
[11] R. Vandebril, M.V. Barel, N. Mastronardi, A parallel QR-factorization/solver of quasiseparable matrices, ETNA 30 (2008) 144–167.
[12] W. Ware, The ultimate computer, IEEE Spectrum 9 (1972) 84–91.