Parallel Processing Letters

SOLVING DISCRETE-TIME LYAPUNOV EQUATIONS FOR THE CHOLESKY FACTOR ON A SHARED MEMORY MULTIPROCESSOR*

JOSE M. CLAVER†
Dpto. de Informática, Universitat Jaume I, Aptdo. 242, 12071-Castellón (SPAIN),

VICENTE HERNANDEZ and ENRIQUE S. QUINTANA‡
Dpto. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Aptdo. 22012, 46071-Valencia (SPAIN).

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

ABSTRACT

In this paper we study the parallel solution of the discrete-time Lyapunov equation. Two parallel fine and medium grain algorithms for solving dense and large order equations $\hat{A}\hat{X}\hat{A}^T - \hat{X} + \hat{B}\hat{B}^T = 0$ on a shared memory multiprocessor are presented. They are based on Hammarling's method and directly obtain the Cholesky factor of the solution. The parallel algorithms work following an antidiagonal wavefront. In order to improve the performance, column-block-oriented and wrap-around algorithms are used. Finally, combined fine and medium grain parallel algorithms are presented.

Keywords: Control theory, linear matrix equations, Lyapunov matrix equations, triangular linear systems, Givens rotations, shared memory multiprocessors.

1. Introduction

Discrete-time Lyapunov equations are related to several problems in control theory and signal processing such as balanced realizations [1, 2] and model reduction of dynamic linear systems [3, 4]. The key to these computational problems is to obtain the balanced transformation and the solution of the balanced realization problem. In particular, in order to compute a balanced transformation, three problems need to be solved. These are the solution of Lyapunov equations, the computation of the

* This research was partially supported by the ESPRIT III Basic Research Programme of the EC under contract No. 9072 (Project GEPPCOM).
† Supported by the *Fundació Caixa-Castelló (No. A-35-IN).*
‡ Supported by the *Conselleria de Educació i Ciència* de la *Generalitat Valenciana.*

Cholesky factor of the solution and the computation of the singular value decomposition (SVD) of a product of matrices. Other applications of these equations are the Hankel-norm approximation problem [5] and the frequency domain approximation problem [7].

The discrete-time Lyapunov equations,

$$\hat{A}\hat{X}\hat{A}^T - \hat{X} + \hat{B}\hat{B}^T = 0 \text{ and } \hat{A}^T\hat{X}\hat{A} - \hat{X} + \hat{C}^T\hat{C} = 0, \tag{1}$$

appear in the computation of balanced transformations of discrete-time linear systems. Among the different algorithms for solving these equations (see [2, 6, 9]) Hammarling's algorithm is specially appropriate, since in this method the Cholesky factor of the solution is directly computed [10, 11]. In this paper we present parallel algorithms with different grain size of parallelism for solving (1) on a shared memory multiprocessor (SMM). These algorithms are based on previous works described in [12, 13]. In particular, parallel shared memory algorithms for solving the continuous-time Lyapunov equation are presented in [13]. An adaptative technique is also described in [13] to improve the performance of the algorithms. However, numerical results of this technique are not given. Our algorithms are applied to the discrete-time case of the Lyapunov equation and seem to be simpler than those. Furthermore, a combination of fine and medium grain algorithms, which improves the performance, is developed and numerical results are given in our paper and [15].

In section 2 Hammarling's algorithm and its data dependency graph are presented. From this graph, an analysis of the parallelism of the method is carried out. In sections 3 and 4, fine and medium grain parallel algorithms are described, respectively. A theoretical time analysis of the proposed algorithms is carried out in section 5. The experimental results obtained on a SMM are shown in section 6. Finally, in section 7 the conclusions of this paper are presented.

2. Hammarling's Method

We focus our study on the discrete-time Lyapunov equation

$$\hat{A}\hat{X}\hat{A}^T - \hat{X} + \hat{B}\hat{B}^T = 0, \tag{2}$$

where the coefficient matrix $\hat{A} \in \Re^{n\times n}$ and $\hat{B} \in \Re^{n\times m}$ with $n \leq m$. When $m < n$, it is possible to apply the same algorithm as described in [10]. If the eigenvalues of the coefficient matrix $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ satisfy $|\lambda_i| < 1$, $i = 1, 2, \ldots, n$, then the solution matrix $\hat{X} \in \Re^{n\times n}$ exists and is unique and non-negative definite. Therefore, it is possible to obtain its Cholesky decomposition $\hat{X} = \hat{L}\hat{L}^T$. However, equation (2) can also be solved directly for the Cholesky factor $\hat{L}$ by Hammarling's algorithm [10, 11]. Below we summarize this algorithm.

First, the original equation (2) is transformed to a simpler form called *reduced Lyapunov equation*. For this purpose, the real Schur decomposition

$$\hat{A} = QSQ^T$$

is computed. In this decomposition, $Q \in \Re^{n\times n}$ is an orthogonal matrix and $S \in \Re^{n\times n}$ is a block lower triangular matrix, with 1×1 or 2×2 diagonal blocks. Each

1×1 block contains a real eigenvalue of the coefficient matrix, whereas each 2×2 block is associated with a pair of complex conjugate eigenvalues. Block algorithms for computing the real Schur decomposition on high performance computers are described in [16].

From this decomposition, the reduced equation is

$$SXS^T - X + BB^T = 0, \tag{3}$$

where $X = Q^T \hat{X} Q$ and $B = Q^T \hat{B}$. Next, the product BB^T is reduced to a simpler form by computing the LQ factorization of B,

$$B = \begin{pmatrix} G & 0 \end{pmatrix} P,$$

where $G \in \Re^{n \times n}$ is lower triangular and $P \in \Re^{m \times m}$ is orthogonal. Therefore, from the solution L of the final reduced Lyapunov equation,

$$S(LL^T)S^T - (LL^T) + GG^T = 0, \tag{4}$$

the Cholesky factor of the original equation (2) is computed as $\hat{L} = QL$.

2.1. The Serial Algorithm

Now we will show how equation (4) is solved. Following Hammarling's algorithm we partition the matrices S, L and G as

$$S = \begin{pmatrix} s_{11} & 0 \\ s & S_1 \end{pmatrix}, L = \begin{pmatrix} l_{11} & 0 \\ l & L_1 \end{pmatrix} \quad \text{and} \quad G = \begin{pmatrix} g_{11} & 0 \\ g & G_1 \end{pmatrix} \tag{5}$$

where s_{11} is either a scalar or a 2×2 block. In the first case, l_{11} and g_{11} are also scalars and s, l and g are column vectors of $n-1$ elements. In the second case, l_{11} and g_{11} will be 2×2 blocks and s, l and g will be $(n-2) \times 2$ blocks.

From now on, and for simplicity, we assume that all the eigenvalues of S are real. We will call this the *real case* of the Lyapunov equation. When some of the eigenvalues of S are complex the equation can be solved in a similar way by means of a block generalization of the algorithms for the *real case* [11]. The sizes of the blocks which appear in the *complex case* are 1×1, 1×2, 2×1 and 2×2.

From (4) and (5) the three following equations are obtained

$$\begin{array}{lcll} l_{11} & = & g_{11}/\sqrt{1 - s_{11}^2}, & \\ (s_{11}S_1 - I_{n-1})l & = & -\alpha g - \beta s & \text{and} \\ S_1(L_1L_1^T)S_1^T - (L_1L_1^T) & = & -GG^T = -G_1G_1^T - yy^T & \end{array} \tag{6}$$

where

$$\alpha = g_{11}/l_{11}, \quad \beta = s_{11}l_{11}, \quad y = \alpha v + s_{11}g, \quad v = S_1 l + s l_{11}$$

and I_{n-1} stands for the identity matrix of order $n-1$.

The diagonal element l_{11} is directly computed from the first equation in (6). Then, the lower triangular linear system in the second equation can be solved by

forward substitution and the vector l is obtained. Finally, the last equation is a discrete-time Lyapunov equation of order $n-1$ where the matrix G is of the form

$$G = \begin{pmatrix} G_1 & y \end{pmatrix},$$

i.e., a block matrix composed of an $n-1 \times n-1$ lower triangular matrix, G_1, and a $n-1$ column vector y. Therefore, it is possible to obtain the Cholesky decomposition of the product GG^T using the LQ factorization

$$G = \begin{pmatrix} \bar{G} & 0 \end{pmatrix} \bar{P},$$

where $\bar{P} \in \Re^{n \times n}$ is orthogonal and $\bar{G} \in \Re^{(n-1)\times(n-1)}$ is lower triangular. Thus, the new reduced Lyapunov equation

$$S_1(L_1 L_1^T)S_1^T - (L_1 L_1^T) + \bar{G}\bar{G}^T = 0, \tag{7}$$

can be treated again in the same way until the problem is completely solved. An algorithmic representation of Hammarling's method is shown in Fig. 1.

Algorithm *serial_solver*:
`do` $j = 1, n$
`1. Compute the diagonal element`
$\alpha = \sqrt{1 - S(j,j)^2}; \quad L(j,j) = G(j,j)/\alpha; \quad \beta = S(j,j)L(j,j)$
`2. Solve the lower triangular linear system for` l
`do` $i = j+1, n$
$L(i,j) = \Big(-\alpha G(i,j) - \beta S(i,j) - \sum_{k=j+1}^{i-1} S(i,k)L(k,j)S(j,j) \Big) / (S(i,i)S(j,j) - 1)$
`end do`
`3. Compute the vector` y `(using` $G(:,j)$ `for update)`
`do` $i = j+1, n$
$y(i) = \alpha \Big(L(j,j)S(i,j) + \sum_{k=j+1}^{i} S(i,k)L(k,j) \Big) - S(j,j)G(i,j)$
`end do`
`4. Compute the Cholesky factor of the matrix` G `of order` $n-j$
`do` $i = j+1, n$
`4.1. Compute a Givens rotation` $(\sin\theta_i, \cos\theta_i)$ `such that`
$$\begin{bmatrix} G(i,i) & y(i) \end{bmatrix} \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix} = \begin{bmatrix} * & 0 \end{bmatrix}$$
`4.2 Apply the Givens rotation`
`do` $k = i+1, n$
$$\begin{bmatrix} G(k,i) & y(k) \end{bmatrix} = \begin{bmatrix} G(k,i) & y(k) \end{bmatrix} \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix}$$
`end do`
`end do`
`end do`
end *serial_solver*

Fig. 1. *Hammarling's serial algorithm.*

2.2. Study of the Data Dependencies

Hammarling's algorithm is column oriented. Consider the j-th column of L, it is necessary to know the elements $L(j:i-1,j)$ prior to computing the element $L(i,j)$. Consider now the computation of the $(j+1)$-th column of L. The first element that should be computed is $L(j+1,j+1)$ but, according to step 1 of the serial algorithm, $G(j+1,j+1)$ should be previously used in iteration j to nullify the $(j+1)$-th element of y. The next element to be computed is $L(j+2,j+1)$, which requires the updated element $G(j+2,j+1)$. Following this process, the data dependency graph for solving a 4×4 discrete-time Lyapunov equation is shown in Fig. 2. (In [13] the data dependency graph for the continuous-time Lyapunov equation is shown).

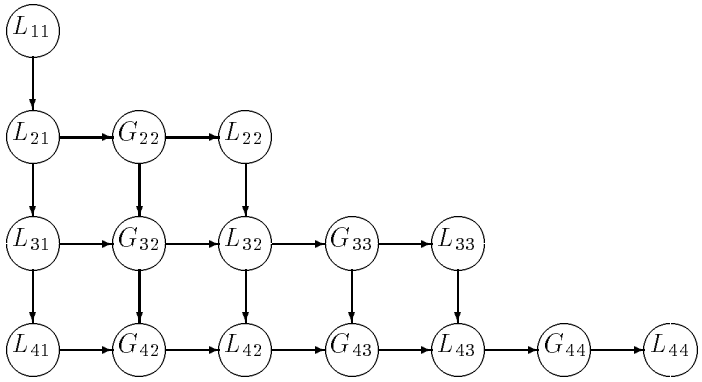

Fig. 2. *Data dependency graph for a* 4×4 *Lyapunov equation.*

From the analysis of the data dependencies, it is possible to observe that the highest inherent parallelism is achieved when the elements on the same antidiagonal of L are computed simultaneously (The same situation occurs in the continuous-time Lyapunov equation [12]). The solving sequence is shown in Fig. 3.

$$\begin{bmatrix}
1 & & & & & & & \\
2 & 3 & & & & & & \\
3 & 4 & 5 & & & & & \\
4 & 5 & 6 & 7 & & & & \\
5 & 6 & 7 & 8 & 9 & & & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & & \\
n-1 & n & n+1 & n+2 & n+3 & \dots & 2n-3 & \\
n & n+1 & n+2 & n+3 & n+4 & \dots & 2n-2 & 2n-1
\end{bmatrix}$$

Fig. 3. *Resolution sequence by antidiagonal elements of* L.

This idea was previously introduced in [8] where it was used to design triangular linear systems solvers on distributed memory multiprocessors. Here it is used to compute simultaneously the solution of triangular linear systems and LQ decompositions. Two approaches can be followed from the idea of a wavefront of antidiagonals. The first one is fine grain size oriented and leads to an algorithm

where the elements of a complete antidiagonal of L are computed at each step. From this approach, an algorithm is obtained which is appropriate for scalar parallel architectures. The second one is medium grain size oriented. In this one, the columns of the matrix L are partitioned in subvectors of length t (fixed or variable) and the computation is carried out so that an antidiagonal of these subvectors is computed at each step. This algorithm is specially appropriate for vector multiprocessors.

3. Fine Grain Parallel Algorithms

From the analysis of the previous section we notice that the element $L(i,j)$ may be computed after the elements $L(1:i, 1:j-1)$ and $L(1:i-1, j)$ have been computed, and the elements $G(j:i, j)$ have been updated (note that the elements above the diagonal are zero). Therefore, the algorithm sweeps the $2n-1$ antidiagonals of L and, using the procedure *pfgle* described in Fig. 4, computes in parallel the elements $L(i,j)$ which belong to the same antidiagonal in each step.

Procedure *pfgle(i,j)*: Compute $L(i,j)$
`if` $i = j$ `then`

1. Compute the diagonal element

$$\alpha(j) = \sqrt{1 - S(j,j)^2}; \quad L(j,j) = G(j,j)/\alpha(j); \quad \beta(j) = S(j,j)L(j,j)$$

`else`

2. Compute the subdiagonal element of l

$$L(i,j) = \Big(-\alpha(j)G(i,j) - \beta(j)S(i,j) - \textstyle\sum_{k=j+1}^{i-1} S(i,k)L(k,j)S(j,j) \Big) / (S(i,i)S(j,j) - 1)$$

3. Update G

3.1. Compute the scalar $\bar{y}$ (using $G(i,j)$ for update)

$$\bar{y} = \alpha(j) \Big(L(j,j)S(i,j) + \textstyle\sum_{k=j+1}^{i} S(i,k)L(k,j) \Big) - S(j,j)G(i,j)$$

3.2. Apply the previous rotations

do $k = j+1, i-1$

$$\begin{bmatrix} G(i,k) & \bar{y} \end{bmatrix} = \begin{bmatrix} G(i,k) & \bar{y} \end{bmatrix} \begin{bmatrix} \cos\theta_{kj} & \sin\theta_{kj} \\ -\sin\theta_{kj} & \cos\theta_{kj} \end{bmatrix}$$

end do

3.3. Compute a Givens rotation $(\sin\theta_{ij}, \cos\theta_{ij})$ such that

$$\begin{bmatrix} G(i,i) & y \end{bmatrix} \begin{bmatrix} \cos\theta_{ij} & \sin\theta_{ij} \\ -\sin\theta_{ij} & \cos\theta_{ij} \end{bmatrix} = \begin{bmatrix} * & 0 \end{bmatrix}$$

`end if`
end *pfgle(i,j)*

Fig. 4. *Procedure pfgle(i,j) for the fine grain algorithm.*

This algorithm has the highest degree of parallelism if the number of processors satisfies $p \geq n/2$. In this case, the number of steps required to compute the solution is equal to the number of antidiagonals of L. However, in practice, p is much smaller and more than one step is required to compute each antidiagonal. Furthermore, on an architecture with a hierarchical structure of the memory, the locality of the data for large matrices has to be considered. In order to avoid the problems in memory access and taking into account the column orientation of Hammarling's algorithm,

matrix L is partitioned by blocks of columns. Obviously, the best performance is obtained when the number of columns in each block, c, is a multiple of p. Thus, given a block h, the elements in the antidiagonal $i = (h-1)c+1, \ldots, n+c-1$ of L, corresponding to this block are

$$L_{i,h(c-1)+1}, L_{i-1,h(c-1)+2}, \ldots, L_{i-t,hc-1}.$$

In order to increase the performance the elements L_{kl} are "wrapped around" the next column block when $n < k \leq n+c-1$. Thus, the loss of efficiency at the end of each column block is reduced. An example of the execution sequence for a 10×10 Lyapunov equation is shown in Fig. 5.

$$\begin{bmatrix}
1 & & & & & & & & & \\
2 & 3 & & & & & & & & \\
3 & 4 & 5 & & & & & & & \\
4 & 5 & 6 & 11^{*} & {}^{(*)} & & & & & \\
5 & 6 & 7 & 12^{*} & 13 & & & & & \\
6 & 7 & 8 & 13 & 14 & 15 & & & & \\
7 & 8 & 9 & 14 & 15 & 16 & 18^{*} & {}^{(*)} & & \\
8 & 9 & 10 & 15 & 16 & 17 & 19^{*} & 20 & & {}^{(*)} \\
9 & 10 & 11 & 16 & 17 & 18 & 20 & 21 & 22 & {}^{(*)} \\
10 & 11 & 12 & 17 & 18 & 19 & 21 & 22 & 23 & 24
\end{bmatrix}$$

* Elements computed on wrap around.
$^{(*)}$ Zero elements; not computed on wrap around.

Fig. 5. *Resolution sequence of L for the fine grain algorithm (n=10,c=3,p=3).*

The accumulation of Givens rotations in this algorithm requires a larger computational and storage cost than other methods [13] though, in some cases, this larger cost is justified [14]. The Givens rotations corresponding to a column block are stored in $2n \times c$ words. Once a new block of L is completely computed and the rest of matrix G has been updated, the block of Givens rotations is no longer required.

4. Medium Grain Parallel Algorithms

Consider a partition of the columns of L in vectors of length t. To simplify, we assume that the dimension of the matrix is a multiple of t. Then, the j-th column of L is partitioned in $f = n/t$ vectors $(v_{1j}, v_{2j}, \ldots, v_{fj})$ where $v_{ij} = (L_{(i-1)t+1,j}, L_{(i-1)t+2,j}, \ldots, L_{it,j})^T$ (note that L_{kj}, $k < j$, are zero).

The resolution sequence and the problems of locality in this case are the same as explained in section 3 though now, the size of the grain is bigger. The procedure *pmgle* which computes a vector $v_{i',j}$ of elements of L is shown in Fig. 6.

This algorithm is specially appropriate for vector processors (or SMM with vector units). In such case, a correct selection of the vector length t is essential. In order to obtain the best performances, t must be a multiple of the dimension of the vector register. In this way, the traffic between the main memory or the cache memories and the vector registers is reduced and their use is optimized. As in the

fine grain case, the algorithm works by blocks of columns and the wrap around technique is implemented in order to improve the performance. To simplify, we chose the number of columns per block c equal to t and multiple of the number of processors, $c = t = k \cdot p$. Different combinations will be shown in section 6.

Procedure *pmgle*(i', j): `Compute` v`(`i`',`j`).`
$i = (i' - 1)t + 1$
`1. Compute the diagonal element`
`if` $i \leq j$ `then`
$\alpha(j) = \sqrt{1 - S(j,j)^2}$; $L(j,j) = G(j,j)/\alpha(j)$; $\beta(j) = S(j,j)L(j,j)$
$i = j + 1$
`end if`
$iend = i + t - 1$
`if` $n < iend$ `then` $iend = n$
`2. Compute the subdiagonal elements of` l
`do` $l = i, iend$

$$L(l,j) = \Big(-\alpha(j)G(l,j) - \beta(j)S(l,j) - \textstyle\sum_{k=j+1}^{l-1} S(l,k)L(k,j)S(j,j) \Big) / (S(l,l)S(j,j) - 1)$$

`end do`
`3. Compute vector` y `(using` $G(:,j)$ `for update)`
`do` $l = i, iend$

$$y(l) = \alpha(j) \Big(L(j,j)S(l,j) + \textstyle\sum_{k=j+1}^{l} S(l,k)L(k,j) \Big) - S(j,j)G(l,j)$$

`end do`
`4. Compute the partial Cholesky factor of G`
`4.1. Apply the previous rotations`
`do` $l = i, iend$
`do` $k = j + 1, i - 1$

$$\begin{bmatrix} G(l,k) & y(l) \end{bmatrix} = \begin{bmatrix} G(i,k) & y(l) \end{bmatrix} \begin{bmatrix} \cos\theta_{kj} & \sin\theta_{kj} \\ -\sin\theta_{kj} & \cos\theta_{kj} \end{bmatrix}$$

`end do`
`end do`
`4.2 Compute and apply new rotations`
`do` $l = i, iend$
`4.2.1. Compute a Givens rotation` $(\cos\theta_{lj}, \sin\theta_{lj})$ `such that`

$$\begin{bmatrix} G(l,l) & y(l) \end{bmatrix} \begin{bmatrix} \cos\theta_{lj} & \sin\theta_{lj} \\ -\sin\theta_{lj} & \cos\theta_{lj} \end{bmatrix} = \begin{bmatrix} * & 0 \end{bmatrix}$$

`4.2.1. Apply the Givens rotation`
`do` $k = l, iend$

$$\begin{bmatrix} G(k,l) & y(k) \end{bmatrix} = \begin{bmatrix} G(k,l) & y(k) \end{bmatrix} \begin{bmatrix} \cos\theta_{lj} & \sin\theta_{lj} \\ -\sin\theta_{lj} & \cos\theta_{lj} \end{bmatrix}$$

`end do`
`end do`
end *pmgle*

Fig. 6. *Procedure pmgle(v(i, j)) for the medium grain algorithm.*

Unless n is a multiple of t, the medium grain algorithms loose some efficiency

when computing the last vector of each column. Some efficiency will also be lost when computing the last blocks of columns of the equation since, in this case, the parallelism is reduced. The larger t is the greater will be the loss of efficiency. In order to improve the performance, when computing the last column blocks of the equation, an adaptative value t can be choosen in each step of the process [13]. A different approach consists of an algorithm which combines fine and medium grain algorithms depending on the situation [15].

5. Time Analysis

In this section, the computational costs of Hammarling's serial algorithm and our parallel algorithms are analyzed. All the eigenvalues of the coefficient matrix A are assumed to be real. Therefore, the real Schur form, S, only has 1×1 diagonal blocks, i.e., it is lower triangular.

Hammarling's serial algorithm (*serial_solver*), as shown in Fig. 1, is divided in 4 stages. The cost (in floating point operations) of each one of these stages is

$$C_{s1} = 5n, \qquad C_{s2} = \frac{n^3}{2} + \frac{3n^2}{2} - 2n,$$
$$C_{s3} = \frac{n^3}{3} + \frac{5n^2}{2} - \frac{17n}{6} \quad \text{and} \quad C_{s4} = n^3 + 3n^2 - 4n.$$

Thus, the total cost is $C_T = \frac{11n^3}{6} + O(n^2)$. If a vector processor is available and a pipeline is applied directly, the theoretical cost is: $\frac{11\beta n^3}{6} + \frac{(14+3\alpha)n^2}{2} + O(n)$, where α is the start up of the vector unit and β the time cost for each operation, obviously with $\beta \leq 1$. This cost is obtained from the analysis of the loops and data dependencies in each stage. The theoretical cost for a direct parallelization of these algorithms on a SMM with p processors is $\frac{11n^3}{6p} + O(n^2)$ for the scalar algorithm and $\frac{11\beta n^3}{6p} + O(n^2)$ for the pipelined algorithm.

The fine grain algorithm (*fgle*) has the same cost as the cost of the scalar *serial_solver*. The cost of this algorithm using p processors is $\frac{11n^3}{6p} + O(n^2)$.

The costs (scalar, vector and parallel) obtained for the medium grain algorithm (*mgle*) have the same expressions as those for the *serial_solver*.

6. Experimental Results

The parallel algorithms have been implemented on a shared memory multiprocessor, the Alliant FX/80, using Fortran 77 with language extensions. This parallel computer has 8 processors and a three-leveled memory. The main memory has 8 banks of 8 Mbytes and is connected by a high-performance bus to two cache memories of 256K bytes each. Furthermore, each processor has its own cache instructions and a set of vector registers which form the third level of the memory (32 registers of 8 bytes each). The vector units are divided in 4 stages. All the algorithms were compiled with the optimization flags $-Og$. The additional flags $-v$ (vector) and $-c$ (concurrent) were used for the vector and parallel algorithms respectively and both flags for the parallel vector algorithms.

All the computations were carried out in double precision and the results of the parallel algorithms were compared to those obtained with Hammarling's serial algo-

rithm. The data matrices were randomly generated. The Matrix S was generated as a lower triangular matrix (only real eigenvalues). The block column sizes tested were 8, 16, 32 and 64; The vector length varied among the same values. The size of the problem, n varied from 20 to 500.

Fig. 7 shows the *Speedup* for the parallel fine grain algorithm with 8 processors. The parallel algorithm is compared to the *serial_solver* executed in 1 processor. Only scalar and concurrent optimizations were used in both algorithms. The *Speedup* for any block column size is close to 6.5 for $n \geq 300$.

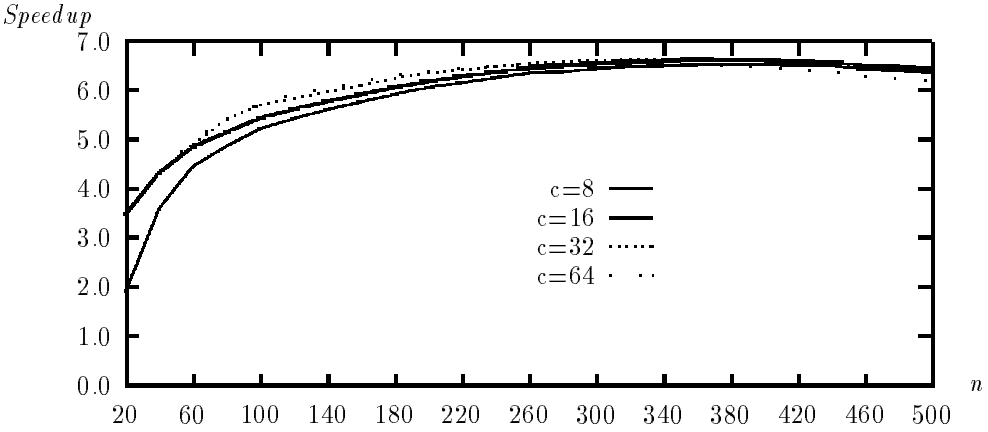

Fig. 7. *Speedup for the fine grain algorithm on 8 processors.*

Fig. 8 shows the *Speedup* for the parallel medium grain algorithm with 8 processors. In this figure, the parallel algorithm is compared to the *serial_solver* executed in 1 processor and vector and concurrent optimizations were used in both algorithms.

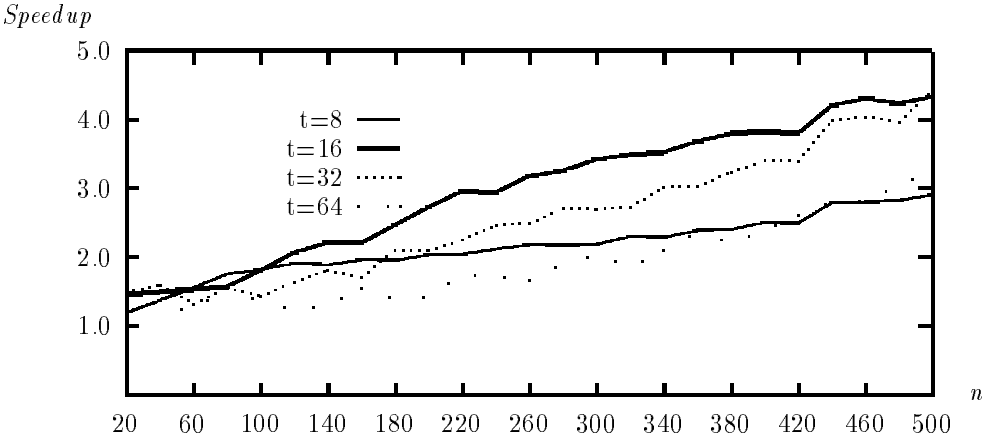

Fig. 8. *Speedup for the medium grain algorithm on 8 processors.*

For parallel medium grain algorithms we obtain a non-uniform behavior unlike the fine grain algorithms (see Fig. 8). The reason is that we chose t to be equal to the number of columns in each column block and the size of the number of columns in a column block a multiple of p. Thus, when t is less than the size of the vector

register, we have an inefficient use of the vector unit, though a higher locality in the accesses to the memory. The opposite situation occurs when t is greater than p. Therefore, the best performances will be obtained when both factors are balanced. In our tests this was achieved for $t = 16$ and 32.

We have tested the use of a tuned BLAS for the Alliant FX/80 in this algorithm, but only BLAS-1 and BLAS-2 could be applied and no important improvement of performance was obtained.

The speedup of the medium grain algorithm is lower for small size equations. In this case, the parallelism of the algorithm is low and frequently the processors are idle. The same situation occurs when computing the last columns of the solution of a larger size equation though, in this case, the effects are not so visible.

Therefore, a combination of fine and medium grain algorithms has been developed. In this algorithm the first r columns of the solution are computed using the medium grain algorithm and then, the fine grain algorithm is used in the last $n - r$ columns. The parameter r depends on size of the vector registers, the number of processors and the value of t. The combination of fine and medium grain algorithms achieves the best performances for any value of n. Fig. 9 shows these results for $t = 16$ and 32. In this figure the combined fine and medium grain algorithm is compared to the *serial_solver*. Vector and concurrent optimizations were used in both algorithms.

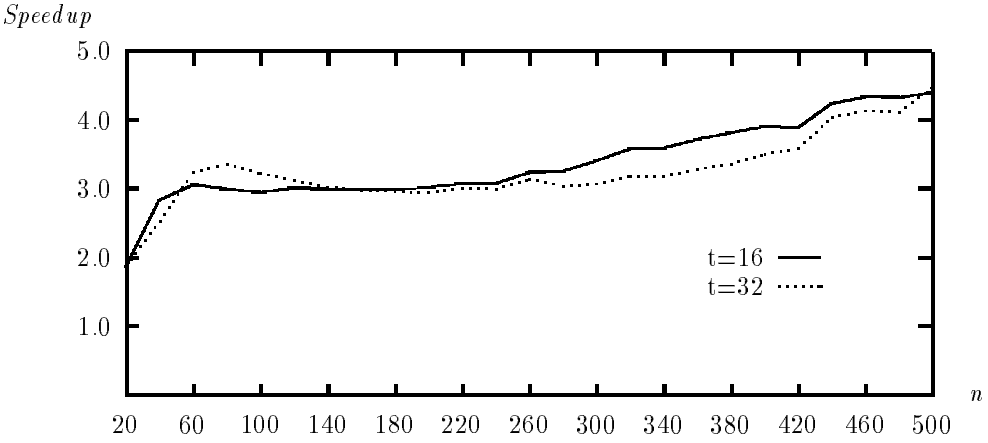

Fig. 9. *Speedup for the combined fine and medium grain algorithm on 8 processors.*

7. Conclusions

We have presented two parallel approaches for the resolution of discrete-time Lyapunov equations based on Hammarling's algorithm. Our study has been focused on large and dense discrete-time equations.

Fine grain algorithms, without vectorization, and using 8 processors, achieve an efficiency of 0.84 for $n = 500$. The *Speedup* of the fine grain algorithms tends to stabilize for large Lyapunov equations (around $n = 300$).

For medium grain algorithms and the sizes of the problems tested, the *Speedup* increases linearly with n. When vectorization is used and $n = 500$, these algorithms

achieve an efficiency of 0.56 on 8 processors.

Furthermore, for almost any dimension of the problem, combined fine and medium grain algorithms offer a better performance due to their higher degree of parallelism when the size of the problem is small and in the last stages of the algorithm.

References

1. B. C. Moore, *Principal component analysis in linear systems: controllability, observability, and model reduction*, IEEE Trans. **AC-26**, (1981) 100-105.
2. A. J.Laub, *Computation of balancing transformations*, Proc. of the Joint Automate Control Conf. Vol. II, (1980)
3. A. J. Laub, M. T. Heat, G. C. Paige, R. C. Ward, *Computations of system balancing transformations and other applications of simultaneous diagonalization algorithms*, IEEE Trans. **AC-32**, (1987) 115-122.
4. L. Pernebo and L. M. Silverman, *Model reduction via balanced state space representations*, IEEE Trans. **AC-2**, (1982) 382-387.
5. K. Glover, *All optimal Hankel-norm approximations of linear multivariable systems and their L-error bounds*, Int. Journal of Control **39**, (1984) 1115-1193.
6. R. H. Bartels and G. W. Stewart, *Algorithm 432. Solution of the matrix equation* $AX + XB = C$, Comm. of the ACM **15**, (1972) 820-826.
7. T. Mullis and R. A. Roberts, *Synthesis of minimum roundoff noise fixed point digital filters*, IEEE Trans. Circuits and Syst. **23**, (1976) 551-562.
8. M. T. Heath and C. H. Romine, *Parallel solution of triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput. **9**, (1988) 558-588.
9. G. H. Golub, S. Nash and C. Van Loan, *A Hessenberg-Schur method for the problem* $AX + XB = C$, IEEE Trans. **AC-24**, (1979) 909-913.
10. S. J. Hammarling, *Numerical solution of the stable, non-negative definite Lyapunov equation*, IMA J. of Numerical Analysis **2**, (1982) 303-323.
11. S. J. Hammarling, *Numerical solution of the discrete-time, convergent, non-negative definite Lyapunov equation*, Systems & Control Letters **17** (North Holland, 1991) 137-139.
12. D. P. O'leary and G. W. Stewart, *Data-flow algorithms for parallel matrix computations*, Comm. of the ACM **28**, (1986) 840-853.
13. A. S. Hodel and K. Polla, *Parallel solution of large Lyapunov equations*, SIAM J. Matrix Anal. Appl. **18**, (1992) 1189-1203.
14. C. Bishof and C. Van Loan, *The WY representation for products of Householder matrices*, SIAM J. Sci. Statist. Comput. **8**, (1987) s2-s13.
15. J. M. Claver, *Algoritmos paralelos de grano fino y medio para resolver la ecuacion de Lyapunov en un multiprocesador con memoria compartida*, TR 08/11/94, Universitat Jaume I, (1994).
16. Z. Bai and J. Demmel, *On a block implementation of Hessenberg multishift QR iteration*, Int. Journal of High Speed Computing **1**, (1989), 97-112.