The Journal of Supercomputing, , 1–20 ( )

Parallel Wavefront Algorithms Solving Lyapunov Equations for the Cholesky Factor on Message Passing Multiprocessors *

JOSÉ M. CLAVER claver@inf.uji.es
Department of Informatics, Universitat Jaume I, Aptdo. 242, 12071-Castellón (SPAIN)

VICENTE HERNÁNDEZ vhernand@dsic.upv.es
Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Aptdo 22012, 46071-Valencia (SPAIN)

Received ; Revised
Editor:

Abstract. In this paper new parallel algorithms to solve the Lyapunov equations for the Cholesky factor using Hammarling's method on message passing multiprocessors are described. These algorithms are based on previous work carried out on the parallel solution of triangular linear systems by using row block data distribution and a wavefront of antidiagonals. The algorithms are theoretically analyzed and experimental results obtained on an SGI Power Challenge and a Cray T3D are presented.

Keywords: Control theory, linear matrix equations, Lyapunov equations, triangular linear systems, message passing mutiprocessors, wavefront algorithms.

1. Introduction

Lyapunov equations are related to a great variety of problems in control theory and signal processing. One of these problems is model reduction [15, 22] by means of the design of balanced realizations of dynamic linear systems [14, 19, 25]. For this purpose we must compute the Cholesky factors for the solutions of two Lyapunov equations and the SVD of the product of these factors. Other applications like the Hankel-norm approximation problem [7], the frequency domaine approximation problem [20] and the solution of Riccati equations using Newton's method also require that these equations be solved. We focus our study on the discrete-time Lyapunov equations

$$AXA^T - X + BB^T = 0,$$
$$A^TXA - X + C^TC = 0.$$

Among the different algorithms for solving these equations [2, 8, 14], the Hammarling's algorithm is specially appropiate, since it directly computes the Cholesky factor of the solution [9, 10]. Several wavefront algorithms using Hammarling's method [5, 6, 12] have been implemented on shared memory multiprocessors.

* This research was partially supported by the CICYT Project # TIC96-1062-C03-01-03

Recently, parallel algorithms for solving triangular linear systems on distributed memory multiprocessors have been developed. These algorithms are: fan-in/fan-out [24], wavefront [11] and cyclics [4, 16, 17]. Therefore, fan-in/fan-out [13] and cyclic algorithms [18, 23], based on the Schur method or the Hessenberg-Schur method have been developed for solving Sylvester and continuous-time Lyapunov equations using distributed memory multiprocessors. This paper is focused on wavefront algorithms, since they have shown good efficiency in the solution of triangular linear systems.

In section 2, Hammarling's algorithm and its data dependency graph are presented. Parallel algorithms with row block data distribution using wavefront of antidiagonals for the solver are described in section 3. A theoretical time analysis of the proposed algorithms is carried out in section 4. In section 5, experimental results on message passing multiprocessors are shown. Finally, the conclusions of the work are described.

2. Hammarling's Method.

The discrete-time Lyapunov equation we want to study is

$$\bar{A}\bar{X}\bar{A}^T - \bar{X} + \bar{B}\bar{B}^T = 0,$$

where $\bar{A}$ and $\bar{B}$ are the coefficient matrices $\bar{A} \in \mathbb{R}^{n\times n}$ and $\bar{B} \in \mathbb{R}^{n\times m}$, with $n \leq m$. When $n > m$, it is possible to apply the same algorithm as described in [9]. If the eigenvalues of the matrix $\bar{A}$, $\{\lambda_1, ..., \lambda_n\}$, satisfy $|\lambda_i| < 1, i = 1, 2, \ldots, n$, then a unique, non-negative definite solution matrix $\bar{X}$ exists. Therefore, it is possible to obtain the Cholesky decomposition of the solution $\bar{X}=\bar{L}\bar{L}^T$, where $\bar{L} \in \mathbb{R}^{n\times n}$ is a lower triangular matrix. However, this equation can also be solved directly for the Cholesky factor $\bar{L}$ by using Hammarling's algorithm [9, 10].

In the first step, the original Lyapunov equation is transformed into a simpler equation called the reduced Lyapunov equation. For this purpose, the real Schur decomposition of $\bar{A}$,

$$\bar{A}= QSQ^T,$$

is computed, where $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix and $S \in \mathbb{R}^{n\times n}$ is a block lower triangular matrix, with 1×1 and 2×2 diagonal blocks. Each 1 block contains a real eigenvalue of the coefficient matrix $\bar{A}$, as 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 [1].

Thus, the resulting equation is

$$SXS^T - X = -BB^T,$$

where $\bar{X}= QXQ^T$ and $\bar{B}= QB$. 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 $G \in \mathbb{R}^{n\times n}$ is lower triangular and $P \in \mathbb{R}^{m\times m}$ is orthogonal. So, from the solution L of the reduced equation

$$S\left(LL^T\right)S^T - \left(LL^T\right) = -GG^T,$$

the Cholesky factor of the original equation is obtained as $\bar{L}= QL$.

2.1. The Serial Algorithm

Following the method described by Hammarling, the matrices S, L and G are partitioned as

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

where s_{11} is either an scalar or a 2×2 block. If s_{11} is an scalar, l_{11} y g_{11} are also scalars, and $\mathbf{s}, \mathbf{l}$, and $\mathbf{g}$ are column vectors of $n-1$ elements. If s_{11} is a 2×2 block, l_{11} and g_{11} are 2×2 blocks and $\mathbf{s}, \mathbf{l}$, and $\mathbf{g}$ are $(n-2)\times 2$ blocks.

From now on, for the sake of simplicity, we assume that all the eigenvalues of S are real. This problem will be denoted as the *real case* of the Lyapunov equation. Hence, the next three equations are obtained:

$$\begin{array}{lcl} l_{11} & = & g_{11}/\sqrt{1-s_{11}^2}, \\ (s_{11}S_1 - I_{n-1})\,\mathbf{l} & = & -\alpha\mathbf{g} - \beta\mathbf{s} \quad \text{and} \\ S_1\left(L_1L_1^T\right)S_1^T - \left(L_1L_1^T\right) & = & -GG^T = -G_1G_1^T - \mathbf{y}\mathbf{y}^T, \end{array}$$

where

$$\alpha = g_{11}/l_{11},\ \beta = s_{11}l_{11},\ \mathbf{y} = \alpha\mathbf{v} - s_{11}\mathbf{g}, \mathbf{v} = S_1\mathbf{l} + sl_{11}$$

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. So, the lower triangular linear system for the second equation can be solved by forward substitution obtaining $\mathbf{l}$. Finally, the last equation is a discrete-time Lyapunov equation of order $n-1$ in which the matrix G is of the form

$$G = \left(\, G_1,\ \mathbf{y} \,\right),$$

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

$$G = \left(\, \bar{G}\ \ 0 \,\right)\bar{P},$$

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

$$S_1\left(L_1L_1^T\right)S_1^T - \left(L_1L_1^T\right) = -\bar{G}\bar{G}^T$$

can be treated again in the same way. This procedure can be repeated until the problem is completely solved. Figure 1 shows an algorithmic representation of Hammarling's method.

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

Figure 1. Hammarling's serial algorithm.

2.2. Study of the Data Dependencies

Hammarling's algorithm is column oriented. Thus, for solving the j-th column, it is necessary to know the elements $L(j : i-1, j), j \leq i$, before computing the element $L(i,j)$. Consider now the computation of the $(j+1)$-th column of L. The first element that must be computed is $L(j+1, j+1)$ but, according to step 1 of the serial algorithm, it is necessary to make previous use of $G(j+1, j+1)$ in the $j - th$ step in order to nullify the $(j+1)$-th element of $\mathbf{y}$. The next element to be computed is $L(j+2, j+1)$, which requires $L(j+1, j+1)$ and the updated element $G(j+2, j+1)$. Following this process, data dependency graph for solving a 4x4 discrete-time Lyapunov equation is shown in figure 2.

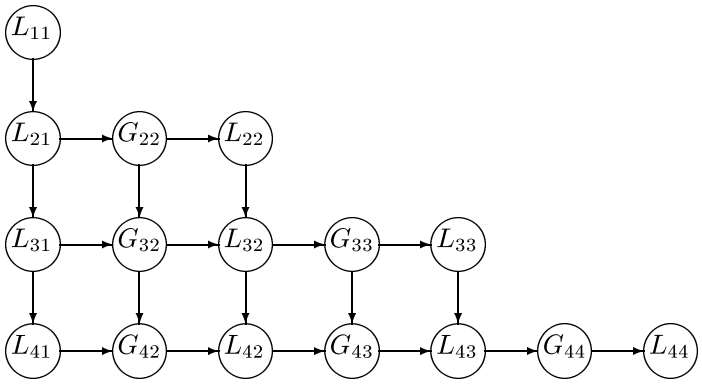

Figure 2. Data dependency graph for a 4×4 Lyapunov equation.

It is important to outline, from the analysis of the data dependencies, that the highest inherent parallelism is achieved when the elements on the same antidiagonal of L are computed simultaneously. The solving sequence is shown in figure 3.

This idea was previously introduced by O'Leary [21] in the context of the Cholesky decomposition problem and it was used to design triangular linear system solvers on distributed memory multiprocessors [11]. In [6, 12], this strategy was used to solve Lyapunov equations by Hammarling's method on shared memory multiprocessors.

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

Figure 3. Solving sequence by antidiagonals of L.

3. Wavefront algorithms

From the analysis of the previous section, it can be observed that element $L(i,j)$ can be computed if elements $L(1:i, 1:j-1)$ and $L(1:i-1, j)$ have been computed (the elements above the diagonal are zero), and elements $G(j:i,j)$ have been updated once. The total amount of antidiagonals for a matrix $S \in \mathbb{R}^{n \times n}$ is $adiag = 2n-1$. Therefore, our algorithm sweeps the $adiag$ antidiagonals of S and

G, using the procedure *wf1(i,j)* described in figure 4, and computes in parallel at each step all the elements $L(i,j)$ which belong to the same antidiagonal.

Procedure $wf1(i,j)$: Compute $L(i,j)$
if $(i = j)$ then
1. Compute the diagonal element.
$\alpha(j) := \sqrt{1 - S(i,i)^2}$
$L(i,i) := G(i,i)/\alpha(j)$
$\beta(j) := L(i,i) \cdot S(i,i)$
else
2. Compute the subdiagonal element.
$$L(i,j) := (-\alpha(j) \cdot G(i,j) - \beta(j) \cdot S(i,j) - S(j,j) \cdot \sum_{k=j+1}^{i-1} S(i,k) \cdot L(k,j)) / (S(i,i) \cdot S(j,j) - 1)$$
3. Update G.
3.1. Compute the vector $\mathbf{y}(i)$.
$$\mathbf{y}(i) := \alpha(j) \cdot (L(j,j) \cdot S(i,j) + \sum_{k=j+1}^{i} S(i,k) \cdot L(k,j)) - S(j,j) \cdot G(i,j)$$
3.2. Apply the previous rotations.
for $k = j+1 : i-1$
$$\begin{bmatrix} G(i,k) & \mathbf{y}(i) \end{bmatrix} := \begin{bmatrix} G(i,k) & \mathbf{y}(i) \end{bmatrix} \begin{bmatrix} \cos\theta_{kj} & \sin\theta_{kj} \\ -\sin\theta_{kj} & \cos\theta_{kj} \end{bmatrix}$$
end for
3.3. Compute and apply the Givens rotation $(\sin\theta_{ij}, \cos\theta_{ij})$ such that:
$$\begin{bmatrix} G(i,i) & \mathbf{y}(i) \end{bmatrix} \begin{bmatrix} \cos\theta_{ij} & \sin\theta_{ij} \\ -\sin\theta_{ij} & \cos\theta_{ij} \end{bmatrix} = \begin{bmatrix} * & 0 \end{bmatrix}$$
end if
end $wf1(i,j)$.

Figure 4. Procedure *wf1(i,j)*.

This algorithm reaches the theoretical highest degree of parallelism when 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 *adiag*. In practice, p is much smaller and more than one step is required to compute each antidiagonal (to simplify, we assume that n is a multiple of p). Taking into account the column orientation of Hammarling's algorithm, and in order to reduce the problems in memory access, matrix L is solved following blocks of columns. Thus, if n is the dimension of the problem and c is an integer such that $c = n/p$, the processor $P_i, i = 0, \ldots, p-1$, solves sequentially the columns $L(:,i), L(:,i+p), \ldots, L(:,i+cp)$. Then, the solution will be stored cyclically by columns. This is specially appropriate for architectures like linear arrays or ring of processors, solving concurrently at each step an antidiagonal of elements in a column block. Therefore, given a column block $h = 0, \ldots, c-1$, the antidiagonal elements $i = hp, \ldots, n+p-1$ of L corresponding to this block are

$$L_{i,hp}, L_{i-1,hp+1}, \ldots, L_{i-p+1,h(p+1)-1}$$

We don't consider elements over the major diagonal or which are out of the order of the solution.

In order to compute $L(i,j)$, the procedure *wf1* needs the rows $S(i,:)$ and $G(i,:)$, the diagonal elements $S(j,j), \ldots, S(i-1,i-1)$, the computed elements $L(j:i-1,j)$ and the rotations $(\sin\theta_{j+1,j}, \cos\theta_{j+1,j}), \ldots, (\sin\theta_{i-1,j}, \cos\theta_{i-1,j})$. Hence, as each processor completely solves a column of L, the processor which computes element $L(i,j)$ also posesses the elements of the $j-th$ column which were previously computed as well as their associated rotations. However, this processor needs the $j-th$ row of S and G, and the diagonal elements $j, \ldots, i-1$ of S. Then, a row block distribution for matrices S and G and a copy of the diagonal of S among all the processors is proposed [11]. Figure 5 shows this data distribution taking $p=4$.

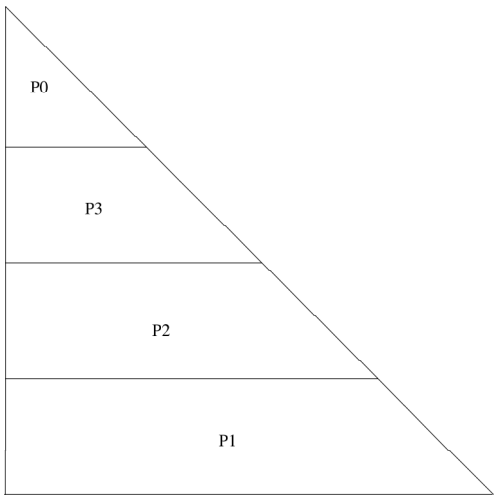

Figure 5. Data distribution of matrices S and G for the wavefront algorithm ($p=4$).

When the order of matrix S is small, a copy of S can be stored in every processor, since it is never modified.

If an unidirectional ring topology is selected, the processor $P_{mod(j,p)}$ computes element $L(i,j)$, updates row $G(i,j+1:i)$ and sends it along with row $S(i,j+1:i)$ to the processor $P_{mod(mod(j,p)+1,p)}$. In figure 6 the solving sequence for $n=9$ and $p=3$ is shown, where the processor number and the step in which it is computed is shown for each element $L(i,j)$.

It is important to notice that the solution L is cyclically distributed by columns. The algorithm *alwf1* uses a buffer in order to store the rows of S and G, adapting the data distribution during the solution process. The initial buffer size is $2n^2/p$, corresponding to pairs of rows of S and G. The amount of information stored is progressively being reduced by one row. For this purpose, two procedures are

$$
\begin{array}{ccccccccc}
0,1 & & & & & & & & \\
0,2 & 1,3 & & & & & & & \\
0,3 & 1,4 & 2,5 & & & & & & \\
0,4 & 1,5 & 2,6 & 0,10 & & & & & \\
0,5 & 1,6 & 2,7 & 0,11 & 1,12 & & & & \\
0,6 & 1,7 & 2,8 & 0,12 & 1,13 & 2,14 & & & \\
0,7 & 1,8 & 2,9 & 0,13 & 1,14 & 2,15 & 0,16 & & \\
0,8 & 1,9 & 2,10 & 0,14 & 1,15 & 2,16 & 0,17 & 1,18 & \\
0,9 & 1,10 & 2,11 & 0,15 & 1,16 & 2,17 & 0,18 & 1,19 & 2,20
\end{array}
$$

Figure 6. Solving sequence of L for the *alwf1* algorithm ($n = 9$, $p = 3$).

defined. The procedure *send_head(Buffer)* takes the rows of S and G , which are stored on the top of the buffer, sends them to the next processor and then deletes them. The procedure *add_tail(Buffer, rowS, rowG)* adds the rows *rowS* and *rowG* to the end of the buffer. The algorithm works using a pipeline structure (see figure 7). Special conditions for the first and last steps of the problem are not considered in this figure.

Algorithm $alwf1$.
Memory: *Buffer(S(n/p,n),G(n/p,n)), L(n/p,n)*
Procedures: *send_head(Buffer), add_tail(Buffer,rowS,rowG)*
for $j = 0 : n - 1$
if $j \in mycol$
for $i = j : n - 1$
if $(i = j)$
receive$(S(j, j), G(j, j))$
send_head(Buffer)
wf1(j,j)
else
receive(S(i,j:i),G(i,j:i))
wf1(i,j)
add_tail(Buffer,$S(i, j + 1 : i), G(i, j + 1 : i)$*)*
send_head (Buffer)
end if
end for
end if
end for
end *alwf1.*

Figure 7. Algorithm *alwf1.*

3.1. Increasing the computation grain.

In order to increase the ratio of the time spent in computation *vs* data communication, we can increase the computation grain. This procedure optimizes the data locality. This increase can be either oriented to vectors (*medium grain*) or blocks (*coarse grain*). The inherent consequence of this aproach will be, in some cases, the reduction of parallelism in the problem, especially on the last steps. The increase of the computational time depends not only on the number of processors and the order of the problem, but also on the computing speed of each node and the communications bandwidth and latency of a particular machine.

In order to solve the problem using a medium grain approach, the columns of L are partitioned in vectors of length t. To simplify, the dimension of the matrix L is assumed to be a multiple of t. Thus, the j-th column of L is partitioned in $f = n/t$ vectors $(v_{0j}, \ldots, v_{f-1,j})^T$, where $v_{ij} = (L_{it,j}, \ldots, L_{(i+1)t-1,j})$ (elements L_{kj}, $k < j$, are zero). The procedure wft which computes the vector v_{ij} of size t is shown in figure 8.

The data distribution and the underlying topology of this case are the same as for the fine grain approach. In the medium grain algorithm, the vector v_{ij} is computed by processor $P_{mod(j,p)}$, which updates rows $G(it : (i+1)t-1, j+1 : (i+1)t-1)$ and sends them along with rows $S(it : (i+1)t-1, j+1 : (i+1)t-1)$ to processor $P_{mod(mod(j,p)+1,p)}$. Figure 9 shows the solving sequence of L for $n = 9$, $p = 3$ and $t = 2$.

The medium grain algorithm (*alwft*) is very similar to the *alwf1* algorithm (see figure 10). The procedures *add_tail* and *send_buffer* act simultaneously in this case on t rows of S and G.

It is easy to extend the previous ideas from the medium grain algorithm to a coarse grain algorithm (*alwfb)* is oriented to blocks. In section 5 results obtained using this extension are shown. In this case, a block of size $n_b \times m_b$ is solved on each step.

Furthermore, other options may be considered in order to improve the performance of these algorithms, as the adaptive value of the computational grain. This approach has been carried out on shared memory multiprocessors by Hodel and Polla [12] and Claver *et al.* [5, 6], but it has not been considered in this paper.

4. Computational Cost.

In this section, the computational costs of Hammarling's serial algorithm and the new parallel wavefront algorithms are analyzed. All the eigenvalues of the coefficient matrix A are assumed to be real. Thus, the real Schur form, S, is lower triangular.

In the analysis of the algorithms we consider the following linear expression to model the communication cost between two processors

$$t = \sigma + M\tau,$$

where σ is the start up time, which is independent of the message length, τ is the cost of transmiting a word and M is the length of the message in words (in our case

Procedure $wft(i,j)$: Compute the vector v_{ij}.
$i' := it$
1. Compute the diagonal element.
if $(i' \leq j)$
$\alpha(j) := \sqrt{1 - S(j,j)^2}$
$L(j,j) := G(j,j)/\alpha(j)$
$\beta(j) := L(j,j) \cdot S(j,j)$
$i' := j + 1$
end if
$i'end := \min(i' + t - 1, n)$
2. Compute the subdiagonal elements of l.
for $l = i' : i'end$
$$L(l,j) := (-\alpha(j) \cdot G(l,j) - \beta(j) \cdot S(l,j) - S(j,j) \cdot \sum_{k=j+1}^{l} S(l,k) \cdot L(k,j)) \\ /(S(l,l) \cdot S(j,j) - 1)$$
end for
3. Compute the vector $\mathbf{y}$.
for $l = i' : i'end$
$$\mathbf{y}(l) := \alpha(j) \cdot (L(j,j) \cdot S(l,j) + \sum_{k=j+1}^{l} S(l,k) \cdot \bar{L}\,(k,j)) - S(j,j) \cdot G(l,j)$$
end for
4. Compute the partial Cholesky factor of G.
4.1. Apply the previous rotations.
for $l = i' : i'end$
for $k = j + 1 : i' - 1$
$$\left[\, G(l,k) \;\; \mathbf{y}(l) \,\right] := \left[\, G(l,k) \;\; \mathbf{y}(l) \,\right] \begin{bmatrix} \cos\theta_{kj} & \sin\theta_{kj} \\ -\sin\theta_{kj} & \cos\theta_{kj} \end{bmatrix}$$
end for
end for
4.2. Compute and apply the new rotations.
for $l = i' : i'end$
4.2.1. Compute the Givens rotation $(\sin\theta_{lj}, \cos\theta_{lj})$ such that:
$$\left[\, G(l,l) \;\; \mathbf{y}(l) \,\right] \begin{bmatrix} \cos\theta_{lj} & \sin\theta_{lj} \\ -\sin\theta_{lj} & \cos\theta_{lj} \end{bmatrix} = \left[\, * \;\; 0 \,\right]$$
4.2.1. Apply the Givens rotation.
for $k = l : i'end$
$$\left[\, G(k,l) \;\; \mathbf{y}(k) \,\right] := \left[\, G(k,l) \;\; \mathbf{y}(k) \,\right] \begin{bmatrix} \cos\theta_{lj} & \sin\theta_{lj} \\ -\sin\theta_{lj} & \cos\theta_{lj} \end{bmatrix}$$
end for
end for
end *wft.*

Figure 8. Procedure *wft(i,j).*

$0,1$
$0,1\ \ 1,2$
$0,2\ \ 1,3\ \ 2,4$
$0,2\ \ 1,3\ \ 2,4\ \ 0,6$
$0,3\ \ 1,4\ \ 2,5\ \ 0,7\ \ 1,8$
$0,3\ \ 1,4\ \ 2,5\ \ 0,7\ \ 1,8\ \ 2,9$
$0,4\ \ 1,5\ \ 2,6\ \ 0,8\ \ 1,9\ \ 2,10\ \ 0,11$
$0,4\ \ 1,5\ \ 2,6\ \ 0,8\ \ 1,9\ \ 2,10\ \ 0,11\ \ 1,12$
$0,5\ \ 1,6\ \ 2,7\ \ 0,9\ \ 1,10\ \ 2,11\ \ 0,12\ \ 1,13\ \ 2,14$

Figure 9. Solving sequence of L for algorithm *alwft* ($n = 9$, $p = 3$, $t = 2$).

Algorithm *alwft.*
Memory: *Buffer(S(n/p,n),G(n/p,n)), L(n/p,n)*
Functions: *send_head(Buffer,t), add_tail(Buffer, row_blockS, row_blockG)*
for $j = 0 : n - 1$
if ($j \in mycol$)
for $i = 0 : n/t - 1$
if ($(i+1)t - 1 \geq j$)
if ($vector \in diagonal$)
receive$((S,G)(j : (i+1)t - 1, j : (i+1)t - 1))$
send_head(Buffer,t)
wft(i,j)
add_tail(Buffer,$((S,G)(j+1 : (i+1)t - 1, j+1 : (i+1)t - 1))$
else
receive$((S,G)(it : (i+1)t - 1, j : (i+1)t - 1))$
wft(i,j)
add_tail(Buffer,$((S,G)(it : (i+1)t - 1, j+1 : (i+1)t - 1))$
s*end_head(Buffer,t)*
end if
end if
end for
end if
end for
end *alwft.*

Figure 10. Algorithm *alwft.*

a word is the unit needed to store a float point number). For simplicity, the values σ and τ are normalized with respect to the computational cost, that is, they will be expressed in flops. A flop is defined as the time spent in performing a floating point operation. A simplified asynchronous model in which only reception time cost has been considered.

4.1. Hammarling's serial algorithm.

The Hammarling's serial algorithm, as shown in figure 1, is divided into four stages. The cost of each one of these stages is

$$\begin{array}{lll} C_{s1} = 5n, & & C_{s2} = \frac{n^3}{3} + \frac{5n^2}{2} - \frac{17n}{2}, \\ C_{s3} = \frac{n^3}{3} + \frac{5n^2}{2} - \frac{17n}{6} & \text{and} & C_{s4} = n^3 + 3n^2 - 4n. \end{array}$$

Thus, the total cost is $C_{res\_ser} = \frac{5n^3}{3} + 8n^2 + O(n)$. The influence of cache memories has not been analyzed.

4.2. Wavefront algorithms.

We will define $C_A(i,j)$ as the arithmetic cost to compute element $L(i,j)$, $C_C(i,j)$ as the communication cost related to $L(i,j)$ and $C_W(i,j)$ as the time spent waiting for data needed to compute $L(i,j)$. Thus, the total cost of $alwf1$ for $L(i,j)$ is

$$C_{alwf1}(i,j) = C_A(i,j) + C_C(i,j) + C_W(i,j),$$

where $C_A(i,j) = 10i - 10j + 24$, $C_C(i,j)$ is the communication cost spent transmitting rows $S(i,j:i)$ and $G(j,j:i)$,

$$C_C(i,j) = \sigma + 2(i-j+1)\tau,$$

and $C_W(i,j)$ is the waiting time cost between the computations of $L(i,j)$ and $L(i+1,j-1)$, defined by

$$C_W(i,j) = \max\{0, C_A(i,j) - C_A(i+1,j-1)\},$$

in which $C(i,j) - C(i+1,j-1)$ is theoretically always negative. Hence, costs due to waiting states are not considered.

Since processor 0 has the highest global cost, an upper limit for the total cost of the algorithm *alwf1* can be defined as

$$C_{alwf1} = \sum_{l=1}^{n/p} \sum_{i=j}^{n} C_{alwf1}(i,j) = \frac{5n^3}{3p}(1+\frac{\tau}{5}) + \frac{n^2}{2}(\frac{17}{p} + \frac{\sigma}{p} + \tau(1+\frac{1}{p}) + 5) + O(n),$$

where $j = (l-1)p + 1$.

Analyzing the expression of the global cost, and considering cuadratic and lower terms negligible, the upper limit of the efficiency for the $alwf1$ algorithm will be

$$\frac{1}{1+\frac{\tau}{5}},$$

which only depends on the communications parameter τ.

The total cost in algorithm *alwft*, $C_{alwft}(i,j)$, in which L is partitioned in vectors v_{ij} of size t, is

$$C_{alwft}(i:i+t-1,j) = \sum_{k=i}^{i+t-1} C_A(k,j) + C_C(i:i+k-1,j) + C_W(i:i+k-1,j)$$

where $C(k,j) = 10k-10j+24$. The cost $C_C(i:i+t-1,j)$ is the communication time needed to send the submatrices $S(i:i+t-1, j:i+t-1)$ and $G(i:i+t-1, j:i+t-1)$, which is defined by

$$C_C(i:i+t-1,j) = \sigma + 2\sum_{k=i}^{i+t-1}(k-j+1)\tau,$$

and $C_W(i:i+t-1,j)$ is the time cost due to waits between the computation of vectors $L(i:i+t-1,j)$ and $L(i+t:i+2t-1,j-1)$,

$$C_W(i:i+t-1,j) = \max\{0, \sum_{k=i}^{i+t-1}(C_A(k,j) - C_A(k+t,j-1))\}.$$

As in algorithm $alwf1$, the cost due to waiting states is not considered.

The upper limit for the global cost of this algorithm is defined by

$$C_{alwft} = \sum_{l=1}^{n/p}\sum_{q=j/t}^{n/t} C_{alwft}(i,j) = \frac{5n^3}{3p}(1+\frac{\tau}{5}) + \frac{n^2}{2}(\frac{19}{p} + \frac{\sigma}{pt} + \tau(1+\frac{1}{p}) + 10) + O(n),$$

where $j = (l-1)p+1$ and $i = (q-1)t+1$.

When the resolution grain is incremented, the cubic order term for the time cost in the algorithm *alwft* is the same as in algorithm *alwf1*. Only the square order cost, $\frac{\sigma}{pt}$, which depends on t is reduced. The lower communication cost and the bigger computation grain produce the highest locality and data reuse. This effect depends on the vector size and can be easily generalized for block algorithms (coarse grain algorithms). However, as the grain size is increased, the inherent parallelism of the problem is reduced. Hence, the grain size is a compromise which depends strongly on computer characteristics.

5. Experimental Results

These parallel algorithms have been implemented on two different machines, an SGI Power Challenge (PCh) and a Cray T3D. These computers reflect two current tendencies in the construction of high performance computers. The PCh is a shared memory multiprocessor (the main memory has 1 GB) with 12 superscalar R10000 processors at 200 MHz, which have 64kB and 2MB of primary and secondary cache memory, respectively.

The Cray T3D is a distributed, but globally shared, memory multiprocessor with 256 superscalar Alpha DEC-21064 processors at 150 MHz. Each processor has a cache memory of 32 KB and a main memory of 64 MB, composing a global memory of 16 GB. The network topology between nodes is a tridimensional torus.

The parallel algorithms have been implemented using language C and the message passing enviroment PVM. On the PCh, communications are implemented using shared memory segments. All the algorithms were compiled with the maximum sequential optimization flags and all computations were carried out in double

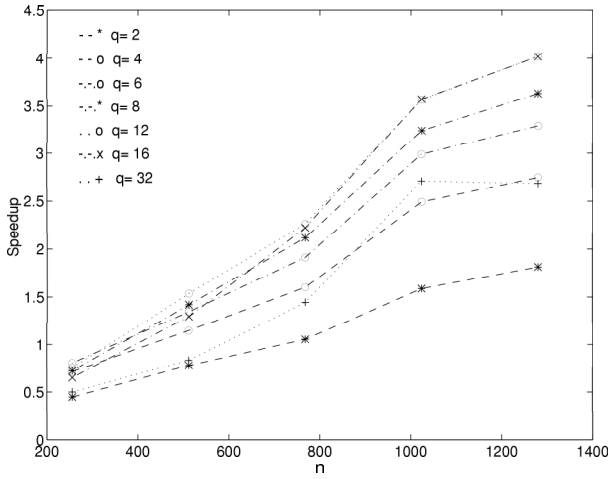

Figure 11. Speedup of the algotihm *alwfb* on the PCh using 4 processors, for different problem orders and blocks sizes $qp \times q$.

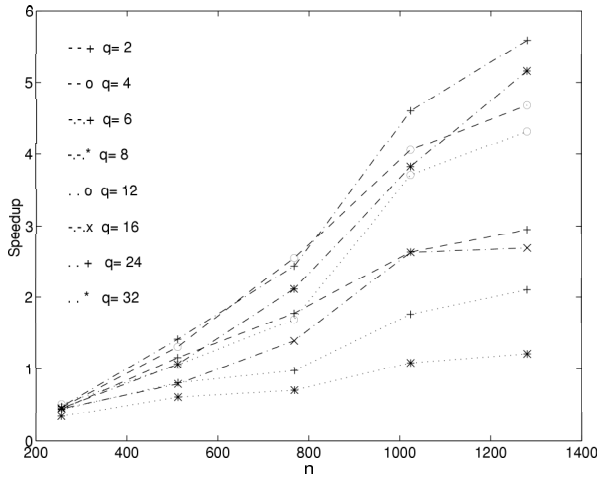

Figure 12. Speedup of the algorithm *alwfb* on the PCh using 8 processors, for different problem orders and blocks sizes $qp \times q$.

precision. The parallel algorithms were compared with the Hammarling's serial

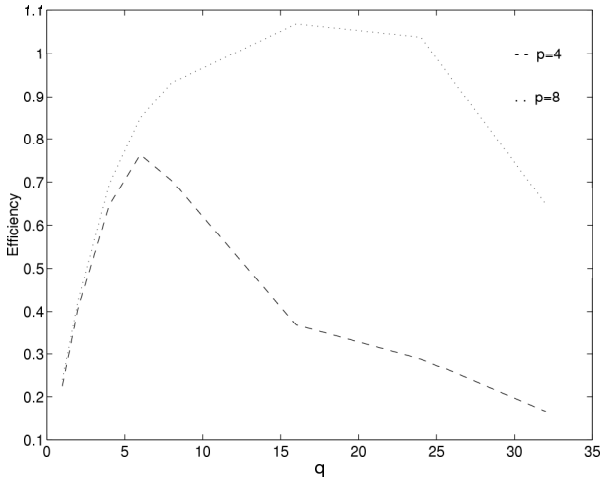

Figure 13. Efficiency of the algorithm *alwfb* on the PCh for $n = 1280$, using 4 and 8 processors, and for different values of q.

algorithm by blocks, in particular blocks of size 16×32 on the PCh and 16×16 on the T3D. This block serial algorithm offers better performance that the non-blocked serial version. For example, on large problems ($n > 1000$), improvements of 100% on the PCh and 30% on the T3D, have been achieved. The results are specially important on the PCh, due to the great size of secondary cache memory of processors.

The algorithm *alwf1* obtained low-performance results due to its small computation grain, and they are not presented in this paper. In the coarse grain algorithm *alwfb*, the block sizes selected are $qp \times q$, where p is the number of processors and $q = 1, 2, 3, \ldots$. We chose these block sizes because the solution matrix L is cyclically distributed among all the processors, optimizing the time spent on communications. When $q = 1$, the behavior the algorithm *alfwb* is equal to algorithm *alwft* with $t = p$.

Figures 11 and 12 show the speedup results for different values of q and n using the PCh for 4 and 8 processors, respectively. Notice that good performances are obtained only for problems with large orders. Large cache memories and the high cost of the communications set up give rise to this behavior. Furthermore, the bus topology on the PCh creates a communications bottleneck, since only one message can be sent at each time. This behavior will become more serious as the number of processors is increased.

The best performance is obtained for a value of q which depends on the number of processors and the order of the problem. For $n = 1280$, the best performance is obtained for $q = 16$ using 4 processors, and for $q = 6$ using 8 processors on the PCh (see figure 13).

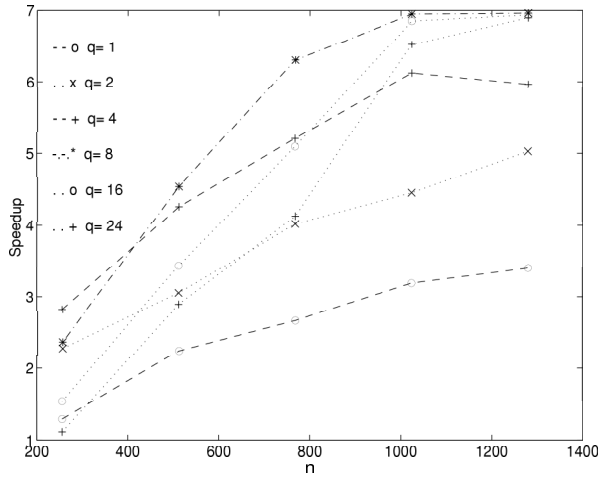

Figure 14. Speedup of the algorithm *alwfb* on the T3D using 4 processors, for different problem orders and blocks $qp \times q$.

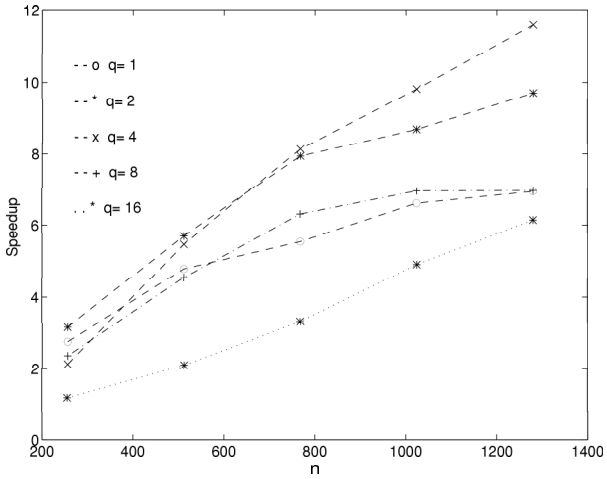

Figure 15. Speedup of the algorithm *alwfb* on the T3D using 8 processors, for different problem orders and blocks $qp \times q$.

Figures 14 and 15 show the speedup results for different values of q and n on the T3D using 4 and 8 processors, respectively. In this case, the results are better

than the ones obtained on the PCh. That is because the cache memory size is lower than the one used on the PCh, hence the performance of the sequential algorithm decreases more quickly when the order of the problem is increased. Also, the topology of the T3D allows the communication of several messages to overlap at the same time.

As in the PCh, the value of q that obtains the best performance also depends on the number of processors and the order of the problem. For $n = 1024$, the best performance is obtained for $q = 8$ using 4 processors and for $q = 4$ using 8 processors (see figure 16). It is important to see that (see figures 13 and 16) for both the PCh and the T3D, the value of q affects the performance more as the number of processors increases. This effect is reflected in figure 17.

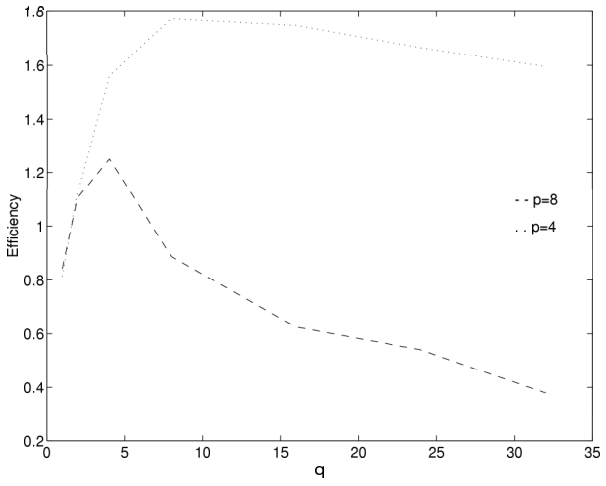

Figure 16. Efficiency of the algorithm *alwfb* on the T3D for $n = 1024$, using 4 and 8 processors and for different values of q.

Efficiencies greater than one have been obtained for both the PCh and T3D, since our algorithm has practically no waits and a better management of the cache memory is performed in the parallel version for large problems. However, as the number of processors is increased, the efficiency of the performance is reduced. This effect is the consequence of the presence of idle processors during the processing of the first and last columns of the problem (see figure 17).

6. Conclusions

New parallel algorithms for the solution of large and dense discrete-time Lyapunov equations using Hammarling's method on message passing multiprocessors have

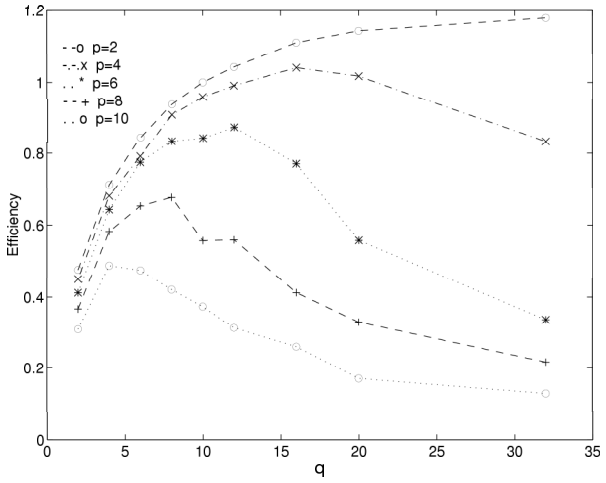

Figure 17. Efficiency of the algorithm *alwfb* on the PCh for $n = 1200$, using different number of processors and values of q.

been presented. These algorithms can be easily adapted to the continuous-time version of the Lyapunov equations.

The key to these algorithms is the use of a wavefront of antidiagonals. This technique was previously used in a similar way to solve triangular linear systems [11]. This method has been generalized for blocks of size $qp \times q$, obtaining good performances for large problems ($n > 1000$) in two of the most currently popular high performance computers, the Cray T3D and the SGI Power Challenge. Due to the properties of cache memory management of these computers, efficiency results great than one have been obtained in some cases.

The performances of the algorithm *alwfb* improve as the order of problems tested n. However, scalability problems for these algorithms are not solved as the number of processors is increased. Nowadays, we are working on the implementation of adaptive algorithms in order to obtain better scalability behavior of parallel algorithms so that the election of the best block to solve the problem does not depend on its order.

Acknowledgments

We thank the Ecole Politecnique Fédérale de Lausanne (EPFL), in Switzerland, for the use of the Cray T3D.

References

1. Bai, Z. and Demmel, J., "On a block implementation of Hessenberg multishift QR iteration", Int. Journal of High Speed Computing, Vol. 1, 1989, pp. 97-112.
2. Bartels, R.H. and Stewart, G. W., "Algorithm 432. Solution of the matrix equation $AX + XB = C$ [F4]", Comm. ACM 15, 1972, pp. 820-826.
3. Bishof, C. and Van Loan,C., "The WY representation for products of Householder matrices", SIAM J. Sci. Statist. Comput., 8, 1987, pp. s2-s13.
4. Chamberlain, R. M., "An algorithm for Lu factorization with partial pivoting on the hypercube", Tech. Rept. CCs 86/11, Dept. of Science and Technology, Chr. Michelson Institute, 1986, Bergen, Norway.
5. Claver, J. M., "Algoritmos de grano fino y medio para resolver la ecuacion de Lyapunov en un multiprocesador con memoria compartida", Technical Report DI 08/11/94, Universitat Jaume I, 1994, Castellón, Spain. (in spanish)
6. Claver, J. M., Hernández, V. and Quintana, E. S., "Solving discrete-time Lyapunov equations for the Cholesky factor on a shared memory multiprocessor", Parallel Processing Letters, Vol. 6, No 3, 1996, pp. 365-376.
7. Glover, K., "All optimal Hankel-norm approximations of linear multivariable systems and their L-error bounds", Int. Journal of Control, Vol. 39, 1984, pp. 1115-1193.
8. Golub, G. H., Nash S. and Van Loan, C., "A Hessenberg-Schur method for the problem $AX + XB = C$", IEEE Trans. A.C., Vol. 24, 1979, pp. 909-913.
9. Hammarling, S. J., "Numerical solution of the stable, non-negative definite Lyapunov equation", IMA J. of Numerical Analysis, Vol. 2, 1982, pp. 303-323.
10. Hammarling, S. J., "Numerical solution of the discrete-time, convergent, non negative definite Lyapunov equation", System & Control Letters, 1991, pp. 137-139.
11. Heath, M. T. and Romine, C. H., "Parallel solution of triangular systems on distributed-memory multiprocessors", SIAM J. Sci. Statist. Comput., Vol. 9, No. 3, 1988, pp. 558-588.
12. Hodel, A. S and Polla, K., "Parallel solution of large Lyapunov equations", SIAM J. Matrix Anal. Appl., Vol. 18, 1992, pp. 1189-1203.
13. Kågström, B. and Poromaa, P., "Distributed block algorithms for the triangular Sylvester equation with condition estimator, Hypercube and Distributed Computers, 1989, pp. 1189-1203.
14. Laub, A. J., "Computation of balancing transformations", Proc. of the Joint Automate Control Conf., Vol. II, 1980.
15. Laub, A. J., Heat, M. T., Paige G. C. and Ward, R. C., "Computations of system Balancing transformations and other applications of simultaneous diagonalization algorithms", IEEE Trans. A.C., Vol. 32, 1987, pp. 115-122.
16. Li, G. and Coleman, T. F., "A new method for solving triangular systems on distributed memory message-passing multiprocessors", Tech. Rept. TR 87-812, Dept. of Computer Science, Cornell University, 1987, New York.
17. Li, G. and Coleman, T. F., "A parallel triangular solver for a distributed-memory multiprocessor", SIAM J. Scientific & Statistical Computing, Vol. 9, 1988, pp. 485-502.
18. Marqués, M., Van de Geijn, R. and Hernández, V., "Parallel algorithms for the triangular Sylvester equation", Third SIAM Conference on Linear Algebra in Signals, Systems and Control, 1993, Seattle.
19. Moore, B. C., "Principal component analysis in linear systems: Controlability, observability, and model reduction", IEEE Trans. A.C., Vol. 26, 1981, pp. 100-105.
20. Mullis, T. and Roberts, R. A., "Synthesis of minimum roundoff noise fixed point digital filters", IEEE Trans. Circuits and Syst. Vol. 23, 1976, pp. 551-562.
21. O'Leary, D. P. and Stewart, G. W., "Data-flow algorithms for parallel matrix computations", Comm. ACM, Vol. 28, 1986, pp. 840-853.
22. Pernebo, L. and Silverman, L. M., "Model reduction via balanced state space representations", IEEE Trans. A.C., Vol. 2, 1982, pp. 382-387.
23. Quintana, E. S., Marqués, M. and Hernandez, V., "Algoritmos por bloques para resolver la ecuación matricial de Sylvester en un Anillo de procesadores", IV Jornadas sobre Paralelismo, 1993, La Coruña, Spain, pp. 1-14. (in spanish)

24. Romine, C. H. and Ortega, J. M., "Parallel solution of triangular systems of equations", Tech. Rep. RM-86-05, Dept. of Applied Mathematics, University of Virginia, 1986, Virginia.
25. Zhou, K., "Frecuency-weighted L_∞ norm and optimal Hankel norm model reduction", IEEE Trans. A.C., Vol. 40, No 10, 1995, pp. 1687-1699.

The Journal of Supercomputing, , 21–20 ( )

Contributing Authors

José M. Claver received his M.S. degree in Physics from the University of Valencia in 1984. From 1985 to 1991, he was an staff member of the Electronics and Computer Architecture Department at the University of Castilla-La Mancha as an Assistant Professor. From 1988 to 1990, he was head of the above Departament and his interest shifted to parallel computing. Since 1991 he is an Assistant Professor at the Informatics Department of the University Jaume I of Castellón, Spain. His research interest include parallel and distributed computing, control and linear systems, numerical linear algebra and computer architecture.

Vicente Hernández received his M.S. degree and Ph.D. degree in Mathematics from the University of Valencia in 1972 and 1979, respectively. He has taught in the School of Mathematics of the University of Valencia, 1973, in the Departament Department of Mathematics of the University of Navarra at San Sebastian from 1973 to 1979, and in the Applied Mathematics Departament of the Valencia University of Technology from 1979 to 1986. Since 1986, he is Professor of Computer Sciences at the Departament of Information Systems and Computation of the Valencia University of Technology. He also served as Chairman of the above Department and Dean of the Faculty of Computer Sciences of the same University. His research interest include control and linear systems, numerical linear algebra and parallel computing.