PARALLEL ADAPTIVE WAVEFRONT ALGORITHMS SOLVING LYAPUNOV EQUATIONS FOR THE CHOLESKY FACTOR ON MESSAGE PASSING MULTIPROCESSORS*

JOSE M. CLAVER[+] and VICENTE HERNANDEZ[++]
[+]*Departamento de Informática, Universitat Jaume I, Campus de Penyeta Roja, Castellón 12071, Spain.*
[++]*Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Valencia 46071, Spain.*

December 9, 1998

Abstract

The order of the matrices involved in several algebraic problems decreases during the solution process. In these cases, parallel algorithms which use adaptive solving blocks sizes offer better performance results than the ones obtained on parallel algorithms using traditional constant block sizes. Recently, new parallel wavefront algorithms solving the Lyapunov equations for the Cholesky factor using Hammarling's method on message passing multiprocessors systems have been designed [5]. In this paper, new parallel adapative versions of these parallel algorithms are described and experimental results obtained on an SGI Power Challenge are presented.

1 Introduction

Lyapunov equations are related to a great variety of problems in control theory and signal processing. One of these problems is the design of balanced realizations of dynamic linear systems [12, 14, 18], of which the most extended application of this technique is model reduction [13, 17]. One approach is to compute the Cholesky factors of the solution of two Lyapunov equations and the SVD of the product of these factors. Other applications like the Hankel-norm approximation problem [6], the frequency domain approximation problem [15] and the solution of Riccati equations using Newton's method also require the solution of these equations.

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 to solve these equations [2, 7, 12], Hammarling's algorithm is specially appropiate, since it directly computes the Cholesky factor of the solution [8, 9]. Several wavefront based [16] algorithms have been implemented on shared memory multiprocessors using this method [4, 11]. More recently, parallel wavefront algorithms for solving Lyapunov equations using the same method on message passing multiprocessors systems have been developed [5]. These

*This research was partially supported by the CICYT grant TIC96-1062-C03-03

algorithms are based on previous wavefront algorithms to solve triangular linear systems [10], which have shown good efficiency in their solution. But, on algebraic problems in which the order of the matrices decrease on the solution process, the selection of the computational granurality is critical. In the last stages of the problem solution, there are waiting times when constant size solving blocks are used during all problem. The size of solving blocks must be lower in the last stages to avoid this situation. In this paper a parallel adaptive wavefront algorithm which selects the best block size in each moment of the problem solution is proposed.

In section 2 Hammarling's algorithm and its dependency graph are presented. Parallel wavefront algorithms are described in section 3. The parallel adaptive wavefront algorithms are presented in section 4. In section 5 experimental results on message passing multiprocessors are shown. Finally, the conclusions of this work are exposed.

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 [8]. If the eigenvalues of the matrix $\bar{A}$, $\{\lambda_1, \ldots, \lambda_n\}$, satisfy $|\lambda_i| < 1, i = 1, 2, \ldots, n$, then a unique and non-negative definite solution matrix $\bar{X}$ exist. 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 [8, 9].

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 1x1 and 2x2 diagonal blocks. Each 1x1 block contains a real eigenvalue of the coefficient matrix $\bar{A}$, as each 2x2 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 $X = Q^T \bar{X} Q$ and $B = Q^T \bar{B}$. 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 $G \in \mathbb{R}^{n \times n}$ 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*

Now, 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} and 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, and for simplicity, we assume that all the eigenvalues of S are real. This problem will be denoted 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 from 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

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

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

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

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. Fig. 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 $k = i, n$

$$\begin{bmatrix} G(k,i) & \mathbf{y}(k) \end{bmatrix} = \begin{bmatrix} G(k,i) & \mathbf{y}(k) \end{bmatrix} \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix}$$

end for

end for

end for

end *res_ser*

Fig. 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 $L(j:i-1, j), j \leq i$, 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, $G(j+1, j+1)$ must be previously used in iteration j to nullify the $(j+1)$-th element of $\mathbf{y}$. 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 dependencies for solving a 4x4 discrete-time Lyapunov equation is shown in Fig 2.

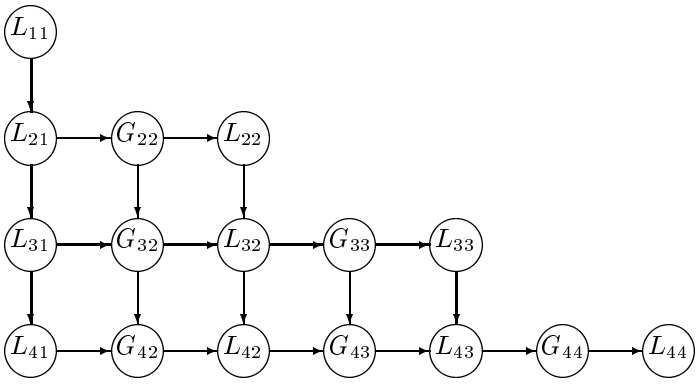

Fig. 2. *Data dependency graph for a* 4×4 *Lyapunov equation.*

It is important to ouline, 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 Fig. 3.

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

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

This idea was previously introduced by O'Leary [16] in the context of the Cholesky decompositon and it was used to design triangular linear system solvers on distributed memory multiprocessors [10]. In [4, 11] this strategy was used to solve Lyapunov equations by Hammarling's method on shared memory multiprocessors.

3 Wavefront algorithms

It can be observed, from the analysis of the previous section, that element $L(i,j)$ can be computed once 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. 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 L and, using the procedure *wf1(i,j)* described in Fig. 4, computes in parallel in each step all the elements $L(i,j)$ which belong to the same antidiagonal (see [5] for more details).

Procedure *wf1(i,j)*: Compute $L(i,j)$
  if $i = j$ then
    1. Compute the diagonal element $L(i,i)$.
  else
    2. Compute the subdiagonal element $L(i,j)$.
    3. Update G.
      3.1. Compute the vector $\mathbf{y}(i)$.
      3.2. Apply the previous Givens rotations $(\sin\theta_{j+1:i-1,j}, \cos\theta_{j+1:i-1,j}) = rot(\theta_{j+1:i-1,j})$
        $\begin{bmatrix} G(i, j+1:i-1) & \mathbf{y}(i) \end{bmatrix} = \begin{bmatrix} G(i, j+1:i-1) & \mathbf{y}(i) \end{bmatrix} \cdot rot(\theta_{j+1:i-1,j})$
      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)*.

Fig. 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 L. 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, the matrix L is solved by blocks of columns, that is, if n is the dimension of the problem and c is an integer such that $c = n/p$, each processor $P_i, i = 0, \ldots, p-1$, solves serially the columns $L(:,i), L(:,i+p), \ldots, L(:,i+cp)$. Then, the solution will be distributed cyclically by columns. This is specially appropriate for architectures like linear arrays or a rings of processors, solving in each step an antidiagonal of elements in a column block. Thus, in a column block $h = 0, \ldots, c-1$, the antidiagonal elements $i = hp, \ldots, n+p-1$ of L 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 and 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 updated 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 column of L is solved by one processor, the processor that computes the element $L(i,j)$ also owns the elements of the column j 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 between all the processors is proposed [10]. Fig. 5 shows this data distribution for four processors, $p = 4$.

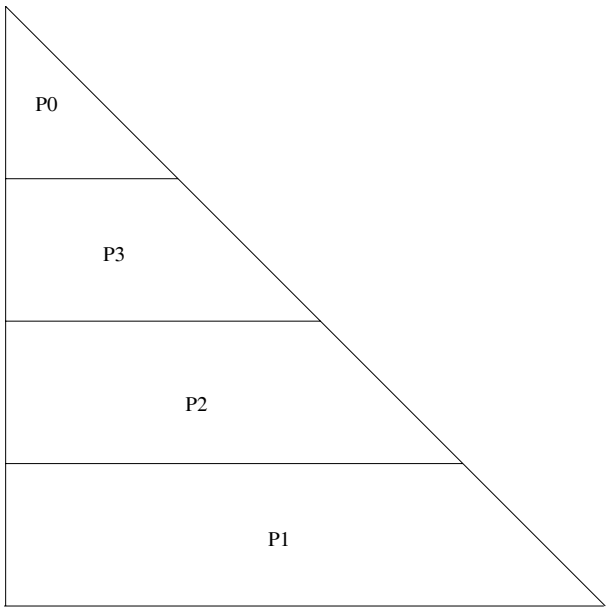

Fig. 5. *Data distribution of matrices S and G for the wavefront algorithm taking p=4).*

When the order of the matrix S is small, a copy of S can be stored in every processor and no exchange is required, since it is never modified.

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

$0,1$								
$0,2$	$1,3$							
$0,3$	$1,4$	$2,5$						
$0,4$	$1,5$	$2,6$	$0,10$					
$0,5$	$1,6$	$2,7$	$0,11$	$1,12$				
$0,6$	$1,7$	$2,8$	$0,12$	$1,13$	$2,14$			
$0,7$	$1,8$	$2,9$	$0,13$	$1,14$	$2,15$	$0,16$		
$0,8$	$1,9$	$2,10$	$0,14$	$1,15$	$2,16$	$0,17$	$1,18$	
$0,9$	$1,10$	$2,11$	$0,15$	$1,16$	$2,17$	$0,18$	$1,19$	$2,20$

Fig. 6. *Resolution sequence of L for the alwf1 algorithm(n=9, p=3).*

It is important to notice that the solution L is cyclically distributed by columns. The algorithm 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 $2n-2/p$, correspoonding to n/p pairs of rows of S and G. This amount of stored data are progresively reduced by one row. For this purpose two procedures are 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 rows *rowS* and *rowG* to the end of the buffer. The algorithm works using a pipeline structure (see Fig. 7).

Algorithm *alwf*1.
Memory: *Buffer((S(0,n),G(0,n)),...,(S(n/p-1,n),G(n/p-1,n))), L(n,n/p)*
Procedures: *send_head(Buffer), add_tail(Buffer,rowS,rowG)*
for $j = 0, n-1$
for $i = j, n-1$

```
        if j ∈ mycol
            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_if
    end_for
end_for
end alwf1.
```

Fig. 7.-The *alwf1 algorithm.*

In order to simplify the algorithm description, considerations about special conditions for the first and last steps of the problem are not considered in this figure.

3.1 *Increasing the computational granularity*

In order to increase the ratio of the time spent in computation *vs* data communications,we can increase the computational granularity. 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 approach will be, in some cases, the reduction of the parallelism in the problem, specially in the last steps. The increase of the computation 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 the latency of a particular machine.

In order to solve the problem using a coarse grain approach, the matrix L is partitioned in blocks of size $n_b \times m_b$. To simplify, the dimension n of the matrix L is assumed to be a multiple of n_b and m_b. Thus, L is partitioned in $(n/n_b) \times (n/m_b)$ blocks L_{ij}, where the block $L_{ij} = L(in_b : (i+1)n_b - 1, im_b : (j+1)m_b - 1)$ (elements $L(k,j)$, $k < j$, are zero). The procedure wfb which computes the block L_{ij} is shown in Fig. 8.

```
Procedure wfb(i,j,n_b,m_b): Compute the block L_ij
    for j' = jm_b,(j+1)mb-1
        for i' = inb,(i+1)n_b-1
            wf1(i',j')
        end for
    end for
end wfb.
```

Fig.8. *Procedure wfb(i, j).*

The procedure *wfb* may be vector (row or column) oriented [5]. This approach supposes high memory locality and consequently, a better performance.

The data distribution and the underlying topology for this case is the same as for the fine grain approach. In the coarse grain algorithm, block L_{ij} is computed by processor $P_{mod(j,p)}$, which update the block $G(in_b : (i+1)n_b - 1, (j+1)m_b : (i+1)n_b - 1)$ and sends them along with the block $S(in_b : (i+1)n_b - 1, (j+1)m_b : (i+1)n_b - 1)$ to processor $P_{mod(mod(j,p)+1,p)}$. Fig. 9 shows the solving sequence of L for $n = 9$, $p = 3$, $n_b = 2$ and $m_b = 1$.

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

Fig. 9. *Solving sequence of L for algorithm alwft (n=9, p=3, n_b=2, $m_b = 1$).*

The coarse grain algorithm (*alwfb*) works in a very similar way to the algorithm *alwf* (see Fig. 10). Each processor completely solves a column block of L in each step, i.e. processor P_i solves serially the column blocks

$$L(:, (i+cp)m_b : (i+cp+1)m_b - 1), \text{ where } c = 0, 1, \ldots, \tfrac{n}{m_b p} - 1.$$

The procedures *add_tail* and *send_buffer* act simultaneously in this case with n_b rows each time.

Algorithm *alwfb.*
Memory: *Buffer(S(n/p,n),G(n/p,n)), L(n/p,n)*
Functions: *send_head(Buffer,n_b), add_tail(Buffer, row_blockS, row_blockG)*
for $j = 0, n/m_b - 1$
for $i = 0, n/n_b - 1$
if $j \in mycolblock$
if $(i+1)n_b - 1 \geq jm_b$
if *block* $\in$ *diagonal*
receive$((S,G)(in_b : (i+1)n_b - 1, jm_b : (i+1)n_b - 1))$
send_head(Buffer,n_b)
wfb(i, j, n_b, m_b)
add_tail(Buffer,$((S,G)(\min((i+1)n_b, (j+1)m_b) : (i+1)n_b - 1,$
$\min((i+1)n_b, (j+1)m_b) : (i+1)n_b - 1))$
else
receive$((S,G)(in_b : (i+1)n_b - 1, jm_b : (i+1)n_b - 1))$
wft(i, j, n_b, m_b)
add_tail(Buffer,$((S,G)(in_b : (i+1)n_b - 1,$
$(j+1)m_b : (i+1)n_b - 1))$
s*end_head(Buffer,n_b)*
end if
end if
end if
end for
end for
end *alwfb.*

Fig. 10.- *Algorithm alwfb*

4 Adaptive wavefront algorithms

In order to improve the performance of wavefront block algorithms we have designed algorithms which have an adaptive value of the computational granularity during the problem solution. This approach has been carried out for shared memory multiprocessors by Hodel and Polla [11] and Claver

et al. [3, 4] but not for distributed memory multiprocessors.

Algorithm *alawfb*(n_{sb},m_{sb},n)
Memory: *Buffer(S(n/p,n),G(n/p,n)), L(n/p,n)*
Functions: *cond(j), send_head(Buffer,n_b), add_tail(Buffer, row_blockS, row_blockG)*
$m_b = m_{sb}$
$n_b = n_{sb}$
$nf_b = n/n_{sb}$
$mf_b = n/m_{sb}$
for $j = 0, mf_b - 1$
if $cond(j)$
$m_b = m_b/r$
$mf_b = rmf_b$
$j = rj$
$n_b = n_b/r$
$nf_b = rnf_b$
end if
for $i = 0, nf_b - 1$
if $j \in mycolblock$
if $(i+1)n_b - 1 \geq jm_b$
if $block \in diagonal$
receive$((S,G)(in_b : (i+1)n_b - 1, jm_b : (i+1)n_b - 1))$
if $(my\_id == p-1)\&(cond(j+1))$
for $k = 0, r-1$
send_head(Buffer,n_b/r)
end for
else
send_head(Buffer,n_b)
end if
wfb(i, j, n_b, m_b)
add_tail(Buffer,$((S,G)(\min((i+1)n_b, (j+1)m_b) : (i+1)n_b - 1,$
$\min((i+1)n_b, (j+1)m_b) : (i+1)n_b - 1))$
else
receive$((S,G)(in_b : (i+1)n_b - 1, jm_b : (i+1)n_b - 1))$
wfb(i, j, n_b, m_b)
add_tail(Buffer,$((S,G)(in_b : (i+1)n_b - 1,$
$(j+1)m_b : (i+1)n_b - 1))$
if $(my\_id == p-1)\&(cond(j+1))$
for $k = 0, r-1$
send_head(Buffer,n_b/r)
end for
else
send_head(Buffer,n_b)
end if
end if
end if
end if
end for
end for
end *alawfb*.

Fig. 11.- *Algorithm alawfb*

The proposed algorithm *alawfb* (see Fig. 11) adapts its solving block $n_b \times m_b$ each time the processors go to solve the next solving column block. The new solving block size will be, if this change is required, $(n_b/r) \times (m_b/r)$, where r is an integer $r = 1, 2, 3, \ldots$. We have called it the "reduction factor" of the solving blocks.

In order to decide the block partition, a boolean function *cond* is defined (see Fig. 12). The function *cond* has as parameters the current solving column block j and the number of processors. The function decides that the algorithm must reduce the solving block if the steps to solve the current column block are less than an *a priori* defined value called *condblock*.

Boolean **function** *cond*(j)
  if $n_b > n_{bmin}$
    if $(\frac{n-jn_b}{n_b} <$ *condblock*$)$ return $true$
    else return $false$
    end if
  else return $false$
  end if
end *cond*.

Fig. 12.- Boolean function *cond(j)*.

The algorithm *alwfb* adapts the solving block until a minimum block size, $n_{bmin} \times m_{bmin}$ is reached. This minimum solving block depends on the computer caracterstics.

5 Experimental Results

These parallel algorithms have been implemented on an SGI Power Challenge (PCh). This computer reflects one current tendency 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 parallel algorithms have been implemented using language C and the massage passing enviroment PVM. Communications are tuned and implemented through the main memory. All the algorithms were compiled with the maximum sequential optimization flags and all computations were carried out in double precision. The parallel algorithms were compared with the Hammarling's serial algorithm by blocks, in particular blocks of size 16x32. This block serial algorithm offers better performance that the non- blocked serial version. For example, on large problems ($n > 1000$), improvements of 100% have been achieved. That is due to the great size of secondary cache memory of the processors.

In algorithm *alwfb*, the solving 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 resulting matrix L is cyclically distributed among all the processors, optimizing the time spent on communications.

Fig. 13 shows the speed up results of algorithm *alwfb* for different values of q and n on the PCh using 8 processors. Notice that good performances are only obtained for problems of large orders. The large cache memory for each processor (2 MB) and the high cost of the communication set up give rise to this behaviour. Furthermore, the bus topology on the PCh creates a communication bottleneck, since only one message can be sent at each time. This one will become more serious as the number of processors is increased.

The value of q that obtains the best performance also depends on the number of processors and the order of the problem. It is important to observe that (see [5]), the value of q affects the performance more as the number of processors increases. This effect is ilustrated in Fig. 14, in which we present efficiency results of algorithm *alwfb* for *n=1200* on 2, 4, 6, 8 and 10 processors using different solving block sizes.

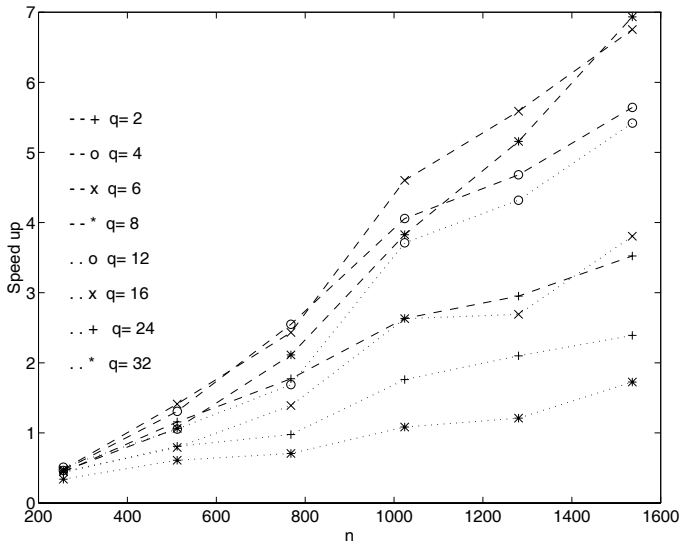

Fig. 13.- *Speed up of algorithm alwfb on the PCh using 8 processors and different values of q.*

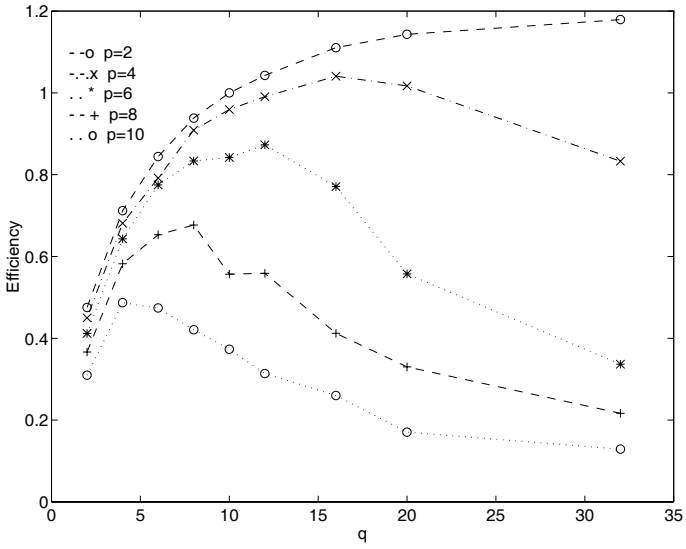

Fig. 14.- *Efficiency of algorithm alwfb on the PCh for n=1200, p=2,4,6,8,10 and different values of q.*

Fig. 15 shows the speed up results of adative algorithm *alawfb* for different values of n and for diferent initial values of q using 8 processors. The reduction factor used in these implementations is $r = 2$, due to the fact that it is the best value we have found in our experimental tests. The smallest block size used is q=2 or q=3, depending on the initial value taken for q. The condition value *condblock* that determines the block partition in the algorithm *alwfb* is, also due to the results obtained in the experimental tests, $condblock = \frac{3p}{2}$. Notice that algorithm *alawfb* presents, on problems of large orders, better performances than algorithm *alwfb*, but on small orders, similar behavior is observed. It is important to see that the performances of the algorithm are more independent of q than algorithm *alwfb*. Thus, it is possible to see that several speed up curves for different initial values of q are very close.

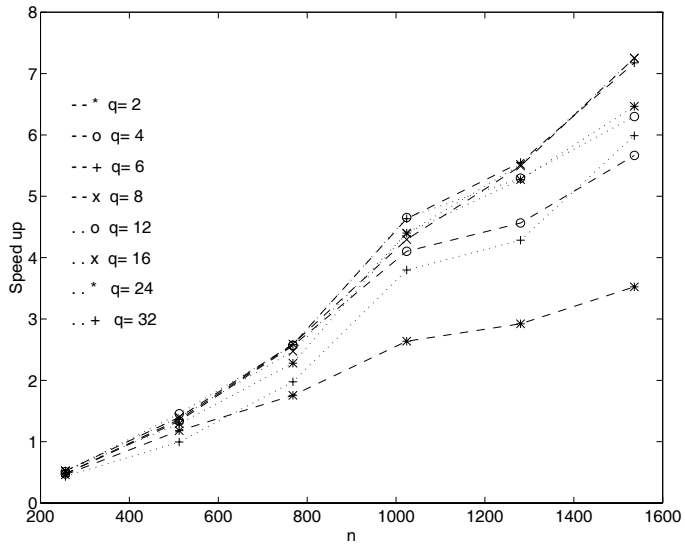

Fig. 15.- *Speed up of algorithm alawfb on the PCh for blocks $qp \times q$, where p=8 and q=2,4,6,8,12,16,24,32.*

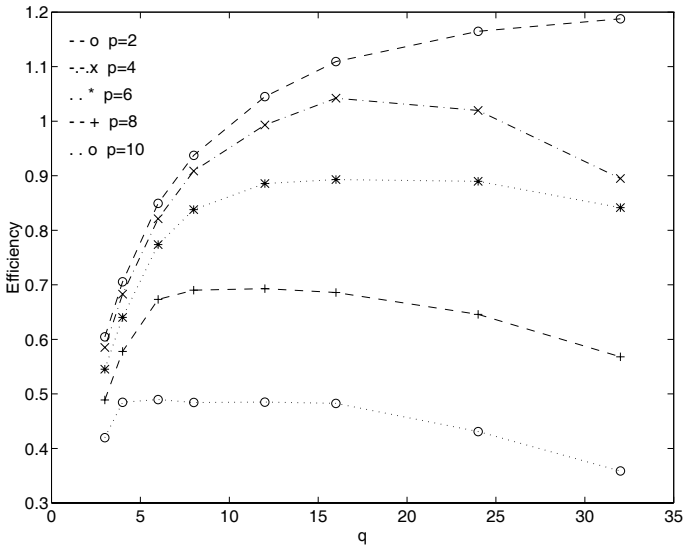

Fig. 16.- *Efficiency of algorithm alawfb on the PCh for n=1200, p=2,4,6,8,10 and different values of q.*

Fig. 16 shows this effect for $n = 1200$ and different number of processors. Notice that, for a wide range of initial values of q, efficiency results for algorithm *alawfb* are mantained. So, block selection affects to the performance less on the algorithm *alafb* than on the algorithm *alwfb*. Efficiencies great than one have been obtained, since these wavefront algorithms have practically no waits (see [5] for a datailed computational cost of this sort of algorithms) and a better management of cache memory is performed in parallel versions for large problems.

6 Conclusions

New parallel adaptive wavefront algorithms for the solution of large and dense discrete-time Lyapunov equations using Hammarling's method on message passing multiprocessors have been presented. These algorithms can be easily adapted to the continuous-time version of the Lyapunov equations.

This method is based in the use of adaptive wavefront of antidiagonals, obtaining good performances for large problems ($n > 1000$), in one of the most currently popular high performance computer, the SGI Power Challenge. The parallel algorithm *alawfb* offers better performances than the parallel algorithm *alwfb* due to the use of an adaptive solving block, and allows the system to obtain the best speed ups for a wide range of problem sizes n just with the same initial value of solving block selection. This behaviour is true for a different number of processors.

The performances of the algorithms grows with the order of problems tested, n. However, scalability problems for these algorithms are not solved when the number of processors is increased because bus topology causes a bottleneck for algorithms where simultaneous comunications are required.

References

[1] Z. Bai and J. Demmel, On a block implementation of Hessenberg multishift QR iteration, Int. Journal of High Speed Computing, Vol. 1, (1989), 97-112.

[2] R. H.Bartels and G. W. Stewart, Algorithm 432. Solution of the matrix equation AX+XB=C [F4], Comm. ACM 15, (1972) 820-826.

[3] J. M. Claver, Algoritmos de grano fino y medio para resolver la ecuacion de Lyapunov en un multiprocesador con memoria compartida, Tech. Rep. DI 08/11/94, Universitat Jaume I, (1994). (in spanish)

[4] J.M. Claver, V. Hernández and E.S. Quintana, Solving discrete-time Lyapunov equations for the Cholesky factor on a shared memory multiprocessor, Parallel Processing Letters, Vol. 6 No 3 (1996) 365-376.

[5] J. M. Claver and V. Hernandez, Parallel wavefront algorithms solving Lyapunov equations for the Cholesky factor on message passing multiprocessors, Tech. Rep. DI 01/04/97, Jaume I University, (1997).

[6] K. Glover, All optimal Hankel-norm approximations of linear multivariable systems and their L-error bounds, Int. Journal of Control 39,(1984), 1115-1193.

[7] G. H. Golub, S. Nash and C. Van Loan, A Hessenberg-Schur method for the problem AX+XB=C, IEEE Trans. A.C. Vol. 24, (1979) 909-913.

[8] S. J. Hammarling, Numerical solution of the stable, non-negative definite Lyapunov equation, IMA J. of Numarical Analysis 2, (1982) 303-323.

[9] S. J. Hammarling, Numerical solution of the discrete-time, convergent, non negative definite Lyapunov equation, System & Control Letters 17 (North Holland, 1991) 137-139.

[10] M. T. Heath and Charles H. Romine, Parallel solution of triangular systems on distributed-memory multiprocessors, SIAM J. Sci. Statist. Comput. Vol. 9 No.3, (1988) 558-588.

[11] A. S. Hodel and K. Polla, Parallel solution of large Lyapunov equations, SIAM J. Matrix Anal. Appl. 18, (1992) 1189-1203.

[12] A. J.Laub, Computation of balancing transformations, Proc. of the Joint Automate Control Conf. Vol. II, (1980).

[13] 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. A.C. 32, (1987) 115-122.

[14] B. C. Moore,Principal component analysis in linear systems: Controlability, observability, and model reduction, IEEE Trans. A.C. 26, (1981), 100-105.

[15] T.Mullis and R. A. Roberts, Synthesis of minimum roundoff noise fixed point digital filters, IEEE Trans. Circuits and Syst. 23, (1976) 551-562.

[16] D. P. O'Leary and G. W. Stewart, Data-flow algorithms for parallel matrix computations, Comm. ACM 28, (1986) 840-853.

[17] L. Pernebo and L. M. Silverman, Model reduction via balanced state space representations, IEEE Trans. A.C. 2, (1982) 382-387.

[18] K. Zhou, Frecuency-weighted L_∞ norm and optimal Hankel norm model reduction, IEEE Trans. A.C. Vol. 40, No 10, (1995) 1687-1699.