UNIVERSIDAD POLITÉCNICA DE VALENCIA
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Algoritmos paralelos y distribuidos para resolver ecuaciones matriciales de Lyapunov en problemas de reducción de modelos

Tesis Doctoral presentada por José M. Claver Iborra

Dirigida por: Dr. D. Vicente Hernández García

Dr. D. Enrique S. Quintana Ortí

Valencia, junio de 1998

Agradecimientos

Quiero expresar en estas breves líneas mi agradecimiento a las muchas personas que me prestaron su ayuda y apoyo durante la realización de esta tesis.

En primer lugar quiero agradecer a mis directores, los Doctores Vicente Hernández y Enrique S. Quintana su inestimable apoyo. En particular, agradecer a Vicente Hernández que me introdujera en este campo de investigación, su magistral forma de enseñar y su constante preocupación por mi trabajo y a Enrique S. Quintana su constante atención y sus consejos durante las últimas etapas de esta tesis.

A mis colegas del departamento de Informática de la Universitat Jaume I y especialmente a mis compañeros del grupo de investigación. He aprendido mucho de ellos y siempre han estado dispuestos a prestarme su ayuda.

A mis padres por su incondicional apoyo y ánimo. Y por último, mi especial agradecimiento a mi mujer, María José, por creer siempre en mi y estar a mi lado en todo momento.

Índice

Capítulo 1

El problema de la reducción de modelos. Conceptos fundamentales

1.1 Introducción

En la actualidad se está realizando un gran esfuerzo por parte de la comunidad científico-técnica para la resolución de problemas de ingeniería dentro de unos límites determinados de tiempo, precisión y coste. Estos problemas tienen cada vez mayor dimensión, en algunos casos demasiado grande como para ser resueltos dentro de los límites de tiempo y precisión deseables, y mayor complejidad. La necesidad de resolver estos problemas ocasiona la creciente utilización de sistemas informáticos paralelos y distribuidos [Perrot 92]. Por una parte se diseñan sistemas *hardware* con mayores prestaciones y que puedan ejecutar el código desarrollado de una forma más eficiente. Por otra, se desarrollan nuevos algoritmos numéricos [Kailath 80, Petkov 91] y estrategias de programación más adecuadas para obtener el mayor provecho de los computadores ya existentes [Dowd 93].

Uno de los temas cruciales en ingeniería es la modelización de sistemas dinámicos complejos. Sin embargo, para que sea factible el tratamiento de problemas reales, son necesarias aproximaciones que lo transformen en un problema más simple. Un sistema dinámico complejo suele ser no lineal, distribuido y variable en el tiempo, por lo que una de las primeras aproximaciones clásicamente realizadas es su transformación en un sistema lineal invariante en el tiempo. Los sistemas lineales invariantes en el tiempo resultantes de esta aproximación inicial suelen ser sistemas de gran tamaño que poseen un gran número de variables de estado [Skelton 88]. Por ello es necesario buscar modelos matemáticos más simples que aproximen al máximo el comportamiento del sistema original. Este modelo, que poseerá menor número de estados que el sistema original, se denomina *modelo reducido* o modelo de orden reducido y al procedimiento utilizado para conseguirlo reducción de modelos [Fortuna 92].

La reducción de modelos, que era un problema abierto en teoría de sistemas hace unos años, actualmente es uno de sus temas fundamentales [Skelton 88]. La aproximación usual para obtener modelos de orden reducido suele ser la misma para sistemas en tiempo continuo que para sistemas en

tiempo discreto. Nos referiremos normalmente al primer tipo de sistemas, ya que en la mayoría de los casos las conclusiones son extensibles sin excesiva dificultad para el caso de tiempo discreto.

1.1.1 Objetivos y justificación

Las ecuaciones de Lyapunov tienen una importancia fundamental en muchos algoritmos de análisis y síntesis en teoría de control. Éstas aparecen de forma natural en problemas de control lineal regidos por ecuaciones diferenciales ordinarias de primer orden autónomas lineales (EDO). Como muchos métodos de control no lineal utilizan el sistema lineal obtenido de la linealización de EDO alrededor de un punto de trabajo, estos métodos también requieren la solución segura y eficiente de este tipo de ecuaciones. Las ecuaciones de Lyapunov generalizadas aparecen de forma natural en los sistemas de control regidos por EDO de segundo orden, sistemas descriptores, o ecuaciones en derivadas parciales (EDP). En [Gajic 95, Mehrmann 91, Petkov 91, Rosen 95, Sima 94, Varga 95, Zhou 96] pueden encontrarse algunas referencias recientes de aplicaciones de ecuaciones de Lyapunov en temas de control. En particular, la reducción de modelos para problemas de control de gran tamaño se ha convertido en uno de los temas más importantes en teoría de sistemas y control en los últimos años. Muchos de los algoritmos propuestos para ello necesitan resolver una o más ecuaciones de Lyapunov (ver, por ejemplo, las referencias [Fortuna 88, Helmke 94, Safonov 88]).

La necesidad de computación paralela en esta área puede deducirse del hecho de que para un sistema con una dimensión en el espacio de estados de orden 1000, la resolución de la ecuación de Lyapunov asociada a éste representa la resolución de un sistema de ecuaciones de 500500 incógnitas (una vez explotada la simetría de la solución). Sistemas de tal dimensión regidas por EDO son comunes en aplicaciones de ingeniería química, aparecen de manera frecuente para sistemas de segundo orden y representan grandes retículos cuando proceden de la discretización de EDP [Gardiner 88, Laub 88, Rosen 95].

El método de Bartels-Stewart [Bartels 72] es el método directo más utilizado para la resolución de las ecuaciones de Lyapunov y es numéricamente estable. Se trata del primer método basado en transformaciones ortogonales que se diseñó para resolver las ecuaciones de Sylvester y Stein y, como particularización, para las ecuaciones de Lyapunov en tiempo continuo y discreto.

En muchas aplicaciones, se requiere el factor de Cholesky de la solución de las ecuaciones de Lyapunov antes que la solución en sí misma [Hammarling 82, Helmke 94, Laub 87]. Esta situación se da en algunos de los métodos de reducción de modelos más utilizados aplicados en el espacio de estados [Laub 87, Safonov 89, Varga 91]. El método de Hammarling [Hammarling 82, Hammarling 91] está basado en el método de Bartels-Stewart y permite obtener directamente el factor de Cholesky de la solución de las ecuaciones de Lyapunov. Por ello uno de los objetivos que esta tesis es el diseño de algoritmos paralelos de éste método y su implementación en las diversas arquitecturas de computadores de altas prestaciones actuales, así como el estudio de las prestaciones obtenidas.

La primera etapa de los métodos de Bartels-Stewart y Hammarling es la simplicación o reducción de la ecuación. En ésta se suele utilizar el algoritmo iterativo QR [Francis 61]. Se trata de un algoritmo que transforma una matriz cuadrada a la forma real de Schur y que tiene un alto coste computacional. Algunos estudios experimentales, basados en distribuciones por bloques, muestran las dificultades en la paralelización del algoritmo QR (con desplazamiento doble implícito) en computadores paralelos [Henry 96] aunque se han propuesto alternativas para incrementar su eficiencia [Henry 97, Watkins 91]. En el caso de las ecuaciones de Lyapunov generalizadas se necesita utilizar en la primera etapa de reducción el algoritmo QZ. Actualmente no existe ninguna implementación paralela de este algoritmo debido a su alta complejidad. Pero, como el algoritmo QR y QZ están compuestos por el mismo tipo de operaciones de grano fino, son de esperar parecidos resultados de paralelismo y escalabilidad para el algoritmo QZ. Sólo la segunda etapa de los métodos

de Bartels-Stewart y Hammarling ofrecen, por sus características, la posibilidad de obtener buenos resultados en computadores paralelos.

Una alternativa a los métodos anteriores es la utilización de métodos basados en la función signo matricial que no requieren de una reducción inicial de la ecuación [Roberts 80]. La función signo matricial fue introducida por Roberts como un método fiable para resolver la ecuación algebraica de Riccati. Se trata de un método que no es numéricamente estable pero que en la práctica se comporta de forma estable en la mayoría de los casos. Recientemente este método ha sido adaptado para resolver las ecuaciones de Lyapunov (estándar y generalizada) para tiempo continuo [Benner 97], obteniéndose muy buenos resultados en computadores monoprocesador. Por otra parte, en [Benner 97] se proponen algoritmos para obtener directamente el factor de Cholesky de la solución de la ecuación de Lyapunov, de aquí que nuestro segundo objetivo sea desarrollar algoritmos paralelos que resuelvan las ecuaciones utilizando este método, estudiando su comportamiento y analizando sus prestaciones.

Una parte importante en el desarrollo de aplicaciones es la portabilidad de éstas y, por lo tanto, la utilización más amplia posible de los estándares que van apareciendo en el mercado. Por tanto, en el desarrollo de los algoritmos se han utilizado en lo posible las librerías computacionales y de comunicaciones más difundidas en la actualidad.

Así, se han utilizado en las implementaciones paralelas de nuestros algoritmos los núcleos computacionales BLAS y LAPACK [Dongarra 87a, Dongarra 88, Lawson 79, Anderson 92] y las librerías de comunicaciones PVM (*Parallel Virtual Machine*) y MPI (*Message-Passing Interface*) [Geist 94, Gropp 94]. Por último, en la implementación de los algoritmos basados en la función signo matricial se ha utilizado el ScaLAPACK, una de las librerías paralelas recientemente desarrolladas que más impacto están teniendo en la computación paralela dentro del campo de la computación numérica.

1.1.2 Estructura de la Tesis

El contenido de esta tesis se ha estructurado en 7 capítulos que se detallan a continuación. Tras la presente introducción y la descripción de los objetivos que pretende abordar esta tesis, se recuerdan algunos conceptos básicos de álgebra lineal y del análisis y diseño de sistemas lineales de control (SLC). La última parte de este primer capítulo está dedicada a la descripción del problema de la reducción de modelos.

En el segundo capítulo se realiza una detallada descripción de los métodos de reducción de modelos basados en el truncamiento del espacio de estados. En especial se detallan los métodos basados en realizaciones balanceadas, tanto de sistemas en lazo abierto como en lazo cerrado, y el método de Schur, debido a las importantes características que imprimen a los modelos reducidos que se obtienen con ellos.

En el tercer capítulo se describen tres métodos para la resolución de las ecuaciones matriciales de Lyapunov. El primero es el clásico método de Bartels-Stewart [Bartels 72], también desarrollado para la resolución de la ecuaciones de Sylvester y Stein. El segundo método es el de Hammarling [Hammarling 82, Hammarling 91]. Se trata de una modificación del método de Bartels-Stewart mediante la que es posible obtener el factor de Cholesky de la solución de la ecuación. Por último analizaremos el método de la función signo matricial, muy utilizado para resolver las ecuaciones matriciales de Riccati y que particularizaremos para la resolución de la ecuaciones de Lyapunov [Roberts 80].

En el cuarto capítulo se realiza una descripción de las ideas básicas en que se fundamentan las arquitecturas y algoritmos paralelos. Analizaremos tanto la clasificación de las arquitecturas más utilizadas en los computadores actuales como los métodos más difundidos para su programación.

También analizaremos algunas de las máquinas paralelas más significativas utilizadas hoy en día y en las que hemos obtenido los resultados experimentales de nuestro trabajo.

El capítulo quinto está dedicado a la descripción de los algoritmos paralelos para resolver las ecuaciones de Lyapunov. Nos hemos centrado en la paralelización de los algoritmos de Hammarling y de la función signo matricial. Se han realizado implementaciones en computadores con memoria compartida y distribuida, y se han utilizado para su programación el paralelismo de control, el paso de mensajes y recientes librerías para el trabajo con computadores de altas prestaciones en las aplicaciones de computación numérica.

En el sexto capítulo se presentan los resultados experimentales que se han obtenido a lo largo de todo el trabajo de esta tesis. Los resultados incluyen el análisis de las arquitecturas utilizadas, los diferentes algoritmos implementados y el comportamiento de los diferentes algoritmos sobre las máquinas sobre las que se han desarrollado. En este sentido se han analizado tanto los incrementos de velocidad de los códigos paralelos respecto de los códigos secuenciales como la escalabilidad de los mismos.

Por último, en el séptimo capítulo se detallan las conclusiones a las que se han llegado en la tesis y las futuras líneas de investigación.

1.2 Conceptos básicos del álgebra matricial

1.2.1 Notación

En primer lugar se introducen la notación, definiciones y resultados básicos de la teoría de matrices utilizados a lo largo de la memoria.

Definición 1 *Las notaciones empleadas para referirnos a elementos, filas, columnas o bloques de la matriz* $A \in \mathsf{R}^{mxn}$ $(A \in \mathsf{C}^{mxn})$ *son las siguientes.*

El elemento en la fila i, columna j de A es a_{ij}. *La matriz A se puede dividir por columnas de la forma* $A = \left[a_1, a_2, \ldots, a_n\right]$, *donde* $a_j \in \mathsf{R}^{m\times 1}\left(a_j \in \mathsf{C}^{m\times 1}\right), j = 1,2,\ldots,n$, *es la columna j-ésima de A. Asimismo, la matriz A se puede dividir por filas de la forma* $A = \left[\overline{a}_1^T, \overline{a}_2^T, \ldots, \overline{a}_m^T\right]$, *donde* $\overline{a}_i \in \mathsf{R}^{1\times n}\left(\overline{a}_i \in \mathsf{C}^{1\times n}\right), i = 1,2,\ldots,m$, *es la fila i-ésima de A. Finalmente, la matriz A se puede dividir por bloques de forma que* $A\left(i:j, p:q\right)$ *denota la subrmatriz de A formada por las filas* $i, i+1, \ldots, j$ *y las columnas* $p, p+1, \ldots, q$.

Definición 2 *Una matriz diagonal es aquella cuyos elementos no diagonales son nulos, es decir,* $a_{ij} = 0, \forall i \neq j$. *Esta matriz queda unívocamente definida por sus elementos diagonales* $A = \mathrm{diag}(a_{11}, a_{22}, \ldots, a_{pp})$, $p = \min(m, n)$.

La matriz identidad de orden n se define como la matriz $I_n = diag(1,1,\ldots,1)$ *de dimensión* $n \times n$.

Una matriz es triangular superior (inferior) si $a_{ij} = 0, \forall i > j$ $(a_{ij} = 0 \; \forall i < j)$.

Una matriz es de Hessenberg superior (inferior) si $a_{ij} = 0, \forall i > j+1 \, (a_{ij} = 0 \, \forall i+1 < j)$.

La matriz transpuesta de $A \in \mathsf{R}^{m\times n}$ *se denota como* $A^T \in \mathsf{R}^{n\times m}$ *y se obtiene al intercambiar las filas de A por sus columnas.*

La matriz conjugada transpuesta de $A \in \mathsf{C}^{m\times n}$ *se denota como* $A^H \in \mathsf{C}^{n\times m}$ *y se obtiene al intercambiar las filas de A por sus columnas y conjugar cada uno de sus elementos.*

Una matriz cuadrada es definida positiva si $x^H Ax > 0, \ \forall x \neq 0$. *Una matriz cuadrada es semidefinida positiva si* $x^H Ax \geq 0, \ \forall x \neq 0$.

Una matriz real es simétrica si $A^T = A$.

Una matriz compleja es Hermitiana si $A^H = A$.

Una matriz es nilpotente si $\exists k : A^k = 0$.

Una matriz cuadrada es invertible (no es singular) si $\exists A^{-1} : A^{-1}A = AA^{-1} = I$.

Una matriz cuadrada real es ortogonal si $A^T A = AA^T = I$.

Definición 3 *El producto de Kronecker de dos matrices A y B de dimensión* $m \times n$ *y* $p \times q$, *respectivamente, es la matriz* $A \otimes B$ *de dimensión* $m\,p \times n\,q$, *definida como*

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}.$$

1.2.2 Valores y vectores propios

El siguiente conjunto de definiciones y resultados introduce diversos conceptos y propiedades relacionados con valores y vectores propios. Todos estos conceptos pueden definirse tanto para matrices cuadradas reales ($A \in \mathsf{R}^{m\times n}$) como para matrices cuadradas complejas ($A \in \mathsf{C}^{m\times n}$).

Definición 4 *Los valores propios de una matriz A de dimensión* $n \times n$ *son las raíces de su polinomio característico*

$$p(z) = \det\left(zI_n - A\right).$$

En esta expresión det *denota el determinante de la matriz.*

El conjunto de valores propios de A, $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$, *también denominado espectro de A, se denotará como* $\Lambda(A)$, *o simplemente* Λ *cuando no haya posibilidad de confusión. Además, si* $\lambda = \alpha + i\beta \in \mathsf{C}$ *entonces* $\mathrm{Re}(\lambda) = \alpha$ *e* $\mathrm{Im}(\lambda) = \beta$.

Proposición 1 *Los valores propios de A satisfacen las siguiente propiedades:*

1. *Los valores propios son invariantes bajo transformaciones de semejanza; esto es, si P es una matriz $n \times n$ no singular (existe su inversa) entonces $\Lambda(A) = \Lambda\left(PAP^{-1}\right)$.*

2. *El determinante de A es igual al producto de sus valores propios.*

3. *La traza de A se define como la suma de los elementos diagonales de A y es igual a la suma de sus valores propios.*

4- *Si $\lambda \in \Lambda(A)$ entonces existe un vector $x \neq 0$ de dimensión n tal que*

$$Ax = \lambda x .$$

Este vector se conoce como el vector propio asociado al valor propio λ.

5. *Si $\lambda = \alpha + i\beta$ es un valor propio complejo de $A \in \mathsf{R}^{n\times n}$ entonces su complejo conjugado, $\overline{\lambda} = \alpha - i\beta$, también es un valor propio de A.*

Definición 5 *La función de separación entre dos matrices A y B de dimensiones $m \times m$ y $n \times n$, respectivamente, se define como*

$$\text{sep}(A,B) = \min_{X\neq 0} \frac{\|AX - XB\|_F}{\|X\|_F} = \min_{x\neq 0} \frac{\|Tx\|_2}{\|x\|_2} = \sigma_{min}(T)$$

donde

$$T = I_m \otimes A - B^H \otimes I_n ,$$

$$\|x\|_2 = \left(\sum_i (x_i^2)\right)^{1/2} \text{ es la 2-norma vectorial,}$$

$$\|X\|_F = \left(\sum_{i,j} (x_{ij}^2)\right)^{1/2} \text{ es la norma de Frobenius,}$$

x es el vector columna, de tamaño m n, que se obtiene colocando consecutivamente las columnas de la matriz X de dimensión $m \times n$, esto es, $x = \text{vec}(X)$, y $\sigma_{min}(T)$ es el menor valor singular de T (el concepto de valor singular de una matriz se introduce en el siguiente apartado).

Esta función mide la separación existente entre los valores propios de *A* y *B*.

Teorema 1 *(Forma real de Schur* [Golub 89]*) Si $A \in \mathsf{R}^{n\times n}$ entonces existe una matriz ortogonal $Q \in \mathsf{R}^{n\times n}$ (esto* es, $Q^T Q = I_n$) tal que

$$Q^T AQ = \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1p} \\ 0 & R_{22} & \cdots & R_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{pp} \end{bmatrix}$$

donde $R_{ii} \in \mathbb{R}^{1x1}$, siendo un valor propio real de *A*, o $R_{ii} \in \mathbb{R}^{2x2}$ y su espectro contiene un par de valores propios complejos conjugados de *A*. En esta descomposición la matriz *A* se ha reducido a la forma casi-triangular superior o forma real de Schur.

Los conceptos anteriores pueden generalizarse para "haces de matrices" mediante el siguiente conjunto de definiciones, proposiciones y teoremas.

Definición 6 *Consideremos las matrices A y B de dimensión* $n \times n$. *El conjunto de todas las matrices de la forma* $A - \lambda B, \lambda \in \mathbb{C}$, *es un haz de matrices.*

Definición 7 *Los valores propios del haz* $A - \lambda B$ *se definen como las raíces del polinomio característico*

$$p(z) = \det(zB - A).$$

Proposición 2 *Los valores propios del haz* $A - \lambda B$ coinciden con *los valores propios de la* matriz $B^{-1}A$, *si B es invertible.*

Teorema 2 *(Forma real de Schur generalizada* [Stewart 73]*) Si* $A, B \in \mathbb{R}^{nxn}$ *entonces existen un par de matrices ortogonales Q y Z tal que* $Q^T AZ$ *es casi-triangular superior y* $Q^T BZ$ *es triangular superior.*

1.2.3 Subespacios invariantes y deflactantes

El siguiente conjunto de definiciones introduce diversos conceptos relacionados con subespacios invariantes y deflactantes.

Definición 8 *Un subespacio* $\mathbb{X}$ *de* $\mathbb{R}^n$ $(\mathbb{C}^n)$ *es un subconjunto de* $\mathbb{R}^n$ $(\mathbb{C}^n)$ *que es asimismo un espacio vectorial. La dimensión del subespacio* $\mathbb{X}$ *se denota por* $\dim(\mathbb{X})$.

Definición 9 *El subespacio formado por todas las combinaciones lineales de los vectores* $\{a_1, a_2, \ldots, a_n\}$ *(todos ellos de la misma dimensión) se denota por* $\text{lin}\{a_1, a_2, \ldots, a_n\}$ *(es decir, la envoltura lineal de dicho conjunto de vectores).*

Si $A = [a_1, a_2, \ldots, a_n]$ *es una partición por columnas de la matriz A entonces* $\text{lin}(A) = \text{lin}\{a_1, a_2, \ldots, a_n\}$.

Definición 10 *Un subespacio invariante* $\mathbb{X}$ *de una matriz A es un subespacio* $\mathbb{X}$ *tal que si* $x \in \mathbb{X}$ *entonces* $Ax \in \mathbb{X}$.

Definición 11 *Un haz regular de matrices* $A - \lambda B$ *es un haz cuadrado (A y B son matrices cuadradas) tal que el polinomio* $\det(zB - A)$ *no es idénticamente nulo (al menos uno de los coeficientes del polinomio es no nulo).*

Definición 12 *Un subespacio deflactante* $\mathbb{X}$ *de un haz regular de matrices* $A - \lambda B$ *es un subespacio tal que*

$$\dim(B\mathsf{X} + A\mathsf{X}) \le \dim(\mathsf{X}) \ .$$

1.2.4 Valores y vectores singulares

A continuación se introducen diversas definiciones relacionadas con valores y vectores singulares y un importante concepto asociado: el rango numérico de una matriz.

Teorema 3 *(Descomposición en valores singulares o SVD* [Golub 89]*) Si* $A \in \mathsf{R}^{m\times n}$*, existe un par de matrices ortogonales* $U \in \mathsf{R}^{m\times m}$ *y* $V \in \mathsf{R}^{n\times n}$ *tal que*

$$U^T AV = \Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p), p = \min(m,n) \ ,$$

donde $\sigma_1 \ge \sigma_2 \ge \ldots \ge \sigma_p \ge 0$.

En esta descomposición σ_i *es el valor singular i-ésimo de A y las columnas i-ésimas de U y V son los vectores singulares asociados a* σ_i *por la izquierda y por la derecha, respectivamente.*

Proposición 3 *La 2-norma matricial (o norma espectral)* $\|A\|_2 = \sup_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2}$ *satisface*

1. $\sigma_{max}(A) = \|A\|_2$

2- $\sigma_{min}(A) = 1 \Big/ \left\|A^{-1}\right\|_2$

Definición 13 *El rango de una matriz* $A \in \mathsf{R}^{m\times n}$ *se define como*

$$\mathrm{rango}(A) = \dim(\mathrm{lin}(A)) \ .$$

El rango de una matriz es el número de columnas o filas linealmente independientes.

Proposición 4 *La matriz* $A \in \mathsf{R}^{m\times n}$ *tiene rango r si y solo si (sii) se cumple que*

$$\sigma_1 \ge \sigma_2 \ge \ldots \ge \sigma_r > 0 = \sigma_{r+1} = \ldots = \sigma_p \ .$$

Es decir, el rango de A coincide con el número de valores singulares no nulos.

La proposición 4 propone una versión discretizada del rango matricial. La discretización de un concepto supone la pérdida de precisión y/o la aparición de problemas numéricos. Una versión continua del rango matricial nos lleva a la siguiente definición del rango numérico.

Definición 14 *Una matriz A tiene rango numérico* (δ, ε, r) *respecto a la norma* $\|\cdot\|$ *si* δ, ε *y* r *satisfacen*

$$r = \inf\left\{\mathrm{rango}(B) : \|A - B\| \le \varepsilon\right\} \ ,$$

y

$$\varepsilon < \delta \leq \sup\left\{ \eta . \|A - B\| \leq \eta \Rightarrow \text{rango}(B) \geq r \right\}$$

Una herramienta básica para el cálculo del rango numérico matricial es la descomposición QR reveladora de rango, que se presenta a continuación.

Definición 15 *La descomposición QR reveladora de rango de la matriz* $A \in \mathsf{R}^{m \times n}$ *consiste en encontrar una matriz de permutación* Π *tal que si* $A\Pi = QR$ *es la descomposición QR de la matriz A permutada por* Π*, entonces* $Q \in \mathsf{R}^{m \times n}$ *tiene columnas ortonormales entre si,* $R \in \mathsf{R}^{n \times n}$ *es triangular superior y se puede dividir por bloques de la forma*

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$$

donde $R_{11} \in \mathsf{R}^{k \times k}$, $R_{22} \in \mathsf{R}^{(n-k) \times (n-k)}$ *verifican que*

$$\sigma_{min}(R_{11}) \approx \sigma_k(A)$$

$$\sigma_{max}(R_{22}) \approx \sigma_{k+1}(A)$$

Si en la matriz R, $\sigma_{min}(R_{11}) = \delta$ *y* $\sigma_{max}(R_{22}) = \varepsilon$ *entonces A tiene rango numérico* (δ, ε, k).

1.2.5 Condicionamiento y estabilidad numérica

Consideremos un problema representado por *f* que actúa sobre unos datos *x* del espacio *D* para producir una solución *y* = *f*(*x*) en el espacio de soluciones *S*. Consideremos la aproximación *x** de *x*. *Si f*(*x**) está cercana a *f*(*x*) entonces se dice que el problema *f* está bien condicionado. Por el contrario, si *f*(*x**) puede diferir considerablemente de *f*(*x*), aun en el caso en que *x* y *x** estén muy cercanas, entonces el problema se dice que está mal condicionado. Este concepto de condicionamiento es estudiado para problemas del álgebra matricial en numerosos textos, entre los que cabe destacar [Golub 89, Stewart 73, Stewart 90, Van Dooren 91, Wilkinson 65].

El condicionamiento es una característica propia del problema. Si un problema está mal condicionado entonces, independientemente del método que utilicemos para resolverlo, la solución calculada puede potencialmente diferir en gran medida de la solución real. El condicionamiento o sensibilidad se expresa mediante el número de condición del problema que se define como

$$k(f, x) = \lim_{\delta \to 0} \sup_{d_2(x, x^*) = \delta} \left(\frac{d_1(f(x^*), f(x))}{\delta} \right), \tag{1.1}$$

donde d_1 y d_2 son funciones de distancia (generalmente inducidas por normas matriciales). En este sentido, si el número de condición de un problema es *k*(*f*,*x*) esto indica que un cambio de orden ε en los datos puede producir una perturbación de orden $\varepsilon k(f,x)$ en la solución.

Cuando se utiliza un algoritmo sobre un computador donde la precisión de la aritmética es necesariamente finita, se introducen pequeños errores durante los cálculos. En consecuencia, denotamos como $\bar{f}(\cdot)$ al algoritmo desarrollado para resolver $f(\cdot)$. Independientemente del

condicionamiento del problema estudiado es deseable que el algoritmo desarrollado no introduzca grandes errores en la solución. Esto es, $\bar{f}(x)$ debe estar próxima a $f(x)$ (la proximidad de $f(x)$ y $f(x^*)$ dependerá del condicionamiento del problema). Los algoritmos que satisfacen esta condición se conocen como algoritmos numéricamente estables.

A menudo se utiliza para evaluar la estabilidad de un algoritmo un análisis regresivo del error. En este tipo de análisis se intenta probar que la solución calculada es la solución del mismo problema con unos datos ligeramente perturbados, esto es, la distancia $d_1(\bar{f}(x), f(x^*))/d_2(x)$ debe ser pequeña. Aquellos algoritmos que satisfacen una cota del error de este tipo se conocen como estables regresivamente *(backward stable)*. La expresión anterior revela que los algoritmos numéricamente estables no introducen en la solución errores mucho más grandes que los ya intrínsecos en los datos del problema.

1.3 Conceptos básicos del análisis y diseño de SLC

1.3.1 El modelo del espacio de estados

El siguiente conjunto de definiciones y teoremas introduce varios conceptos básicos de la teoría de control correspondientes al análisis y diseño de SLC.

Después de una definición formal de los SLC en el modelo del espacio de estados se introducen varias propiedades importantes de estos sistemas tales como estabilidad, controlabilidad, estabilizabilidad, observabilidad y detectabilidad.

Definición 16 *Un SLC generalizado o descriptor, continuo e invariante en el tiempo, está definido por una ecuación diferencial y una ecuación algebraica*

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t),\ x(0) = x_{0,} \\ y(t) &= Cx(t), \end{aligned} \tag{1.2}$$

donde $x(t)$ es el vector de estados de dimensión n, $u(t)$ es el vector de controles (entradas) de dimensión m e $y(t)$ es el vector de salidas de dimensión p. El par (A,E), $A, E \in \mathbb{R}^{n\times n}$, define el estado del sistema, $B \in \mathbb{R}^{n\times m}$ es la matriz de controles (entradas) y $C \in \mathbb{R}^{p\times n}$ es la matriz de salidas [Kailath 80, Petkov 91].

Definición 17 *Un SLC continuo e invariante en el tiempo puede verse como un caso particular de un SLC generalizado donde $E = I_n$*

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t),\ x(0) = x_{0,} \\ y(t) &= Cx(t), \end{aligned} \tag{1.3}$$

En ocasiones nos referiremos al SLC como la tripleta (A, B, C).

En el caso de que E sea invertible el SLC generalizado (1.2) *se puede tratar como un SLC, reescribiéndolo del siguiente modo*

$$\begin{aligned} \dot{x}(t) &= E^{-1}Ax(t) + E^{-1}Bu(t), x(0) = x_{0,} \\ y(t) &= Cx(t), \end{aligned}$$

quedando definido el nuevo SLC por la tripleta $(E^{-1}A, E^{-1}B, C)$.

Definición 18 *La matriz función de transferencia del SLC* (1.3), $G(s) \in \mathbb{R}^{p\times m}$, *viene determinada por la siguiente expresión*

$$G(s) = C(sI_n - A)^{-1}B\,.$$

La tripleta (*A, B, C*) *es una realización de G*(*s*) *de orden n.*

La representación o realización (1.3) asociada a una matriz función de transferencia *G*(*s*) no es única. Por ejemplo, si realizamos el cambio de variables

$$x(t) = P\hat{x}(t)$$

donde *P* es una matriz no singular entonces obtenemos el SLC equivalente

$$\begin{aligned} \dot{\hat{x}}(t) &= \hat{A}\hat{x}(t) + \hat{B}u(t), \hat{x}(0) = \hat{x}_{0,} \\ y(t) &= \hat{C}\hat{x}(t), \end{aligned} \tag{1.4}$$

donde $\hat{A} = P^{-1}AP$, $\hat{B} = P^{-1}B$, $\hat{C} = CP$ y $\hat{x}_0 = P^{-1}x_0$. La nueva expresión para *G*(*s*) será

$$G(s) = \hat{C}(sI_n - \hat{A})^{-1}\hat{B} = C(sI_n - A)^{-1}B\,.$$

Prácticamente todas las definiciones y teoremas que expondremos para SLC generalizados en tiempo continuo tienen una versión análoga para el caso de SLC generalizados en tiempo discreto

$$\begin{aligned} Ex_{k+1} &= Ax_k + Bu_k\,, \\ y_k &= Cx_k\,. \end{aligned} \tag{1.5}$$

Por simplicidad, durante el resto del trabajo, restringiremos el estudio al caso de SLC continuos, salvo en las ocasiones en las que realicemos referencia explícita a los SLC generalizados y/o en tiempo discreto.

1.3.2 Estabilidad

En la siguiente definición y teorema se presenta una caracterización de los SLC asintóticamente estables, estables e inestables.

Definición 19 *Un SLC continuo e invariante en el tiempo definido por*

$$\dot{x}(t) = Ax(t), x(0) = x_{0,} \tag{1.6}$$

es asintóticamente estable si $lim_{t\to\infty}x(t) = 0, \forall x_0$. *Si* $\forall x_0$, $\exists c < \infty$ *tal que* $\lim_{t\to\infty}\|x(t)\| < c$ *entonces el sistema es estable. Si existe un estado inicial,* $\widetilde{x}_0$, *tal que* $lim_{t\to\infty}x(t) = \infty, x(0) = \widetilde{x}_0$, *entonces el sistema es inestable.*

Teorema 4 [Gantmacher 66] *El SLC* (1.6) *es asintóticamente estable sii todos los valores propios de A tienen parte real negativa, esto es,* $\mathrm{Re}(\lambda) < 0, \forall \lambda \in \Lambda(A)$. *En tal caso A es una matriz de estabilidad.*

El SLC (1.6) *es estable sii todos los valores propios de A tienen parte real no positiva y aquellos valores propios con parte real nula aparecen, en la forma canónica de Jordan de A, en bloques de Jordan de dimensión* 1×1.

El hecho de que el SLC (1.6) sea asintóticamente estable tiene la siguiente consecuencia importante.

Proposición 5 *Si el SLC* (1.6) *es asintóticamente estable entonces el SLC* (1.3) *es tal que entradas u(t) acotadas producen salidas y(t) acotadas.*

1.3.3 Controlabilidad y Estabilizabilidad

Dos importantes propiedades de los SLC son la controlabilidad y la estabilizabilidad. Estas propiedades se consideran asociadas a la parte del SLC (1.3) que relaciona el par entrada-salida, es decir

$$\dot{x}(t) = Ax(t) + Bu(t), x(0) = x_0, \tag{1.7}$$

Definición 20 *El SLC* (1.7) *(el par (A, B)) es controlable si* $\forall x_0$ *y para cualquier estado final* x_f *existe un tiempo finito* t_f *y un control* $u(t), 0 \le t \le t_f$, *tal que* $x(t_f) = x_f$.

El SLC (1.7) *(el par (A, B)) es estabilizable si existe un control en lazo cerrado,* $u(t) = -Kx(t) + \bar{u}(t)$, $K \in \mathsf{R}^{m\times n}$, *que transforma el sistema en*

$$\dot{x}(t) = (A - BK)x(t) + B\bar{u}(t), x(0) = x_{0,}$$

tal que A - BK es una matriz de estabilidad.

Teorema 5 [Kailath 80] *Si el SLC* (1.7) *es controlable entonces es estabilizable.*

Si el SLC (1.7) *no es controlable entonces existe una transformación no singular en el espacio de estados,* $x(t) = P\bar{x}(t)$, *que reduce el sistema a la forma*

$$\begin{bmatrix} \dot{\bar{x}}_1(t) \\ \dot{\bar{x}}_2(t) \end{bmatrix} = P^{-1}AP\bar{x}(t) + P^{-1}Bu(t) = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}\begin{bmatrix} \bar{x}_1(t) \\ \bar{x}_2(t) \end{bmatrix} + \begin{bmatrix} B_1 \\ 0 \end{bmatrix}u(t)$$

donde el par (A_{11}, B_1) *es controlable y la dimensión del vector* $\bar{x}_1(t)$ *es igual a*

$$rango\left(\left[B, AB, \ldots, A^{n-1}B\right]\right).$$

Si A_{22} *es una matriz de estabilidad entonces el par (A, B) es estabilizable.*

Puede comprobarse que como el par (A_{11}, B_1) *es controlable, existe una matriz de realimentación* K_1*, tal que* $A_{11} - B_1K_1$*, es estable. Si* A_{22} *es también estable entonces la matriz de realimentación* $K = [K_1, 0]P^{-1}$ *estabiliza el par* (*A, B*).

1.3.4 Observabilidad y Detectabilidad

Las dos últimas propiedades que van a ser introducidas son la observabilidad y detectabilidad de los SLC. Estas propiedades se consideran relacionadas con la parte del SLC (1.3) del par estado-salida en ausencia de controles, $u(t) \equiv 0$, es decir

$$\begin{aligned} \dot{x}(t) &= Ax(t), x(0) = x_{0,} \\ y(t) &= Cx(t). \end{aligned} \tag{1.8}$$

Definición 21 *El SLC* (1.8) *(el par* (*A, C*)*) es observable si* $\forall x_0$ *existe un tiempo finito* t_f *tal que, a partir del conocimiento de las salidas,* $y(t),\ 0 \le t \le t_f$*, es posible determinar* x_0.

El SLC (1.8) *(el par* (*A, C*)*) es detectable si existe una matriz* $K \in \mathsf{R}^{nxp}$ *tal que* $A - KC$ *es una matriz de estabilidad.*

Teorema 6 [Kailath 80] *Si el SLC* (1.8) *es observable entonces es detectable.*

Si el SLC (1.8) *no es observable entonces existe una transformación no singular en el espacio de estados,* $x(t) = S\bar{x}(t)$*, que reduce el sistema a la forma*

$$\begin{bmatrix} \dot{\bar{x}}_1(t) \\ \dot{\bar{x}}_2(t) \end{bmatrix} = S^{-1}AS\,\bar{x}(t) = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} \bar{x}_1(t) \\ \bar{x}_2(t) \end{bmatrix}$$

$$y(t) = CS\bar{x}(t) = \begin{bmatrix} C_1 & 0 \end{bmatrix} \begin{bmatrix} \bar{x}_1(t) \\ \bar{x}_2(t) \end{bmatrix}$$

donde el par (A_{11}, C_1) *es observable y la dimensión del vector* $\bar{x}_1(t)$ *es igual a*

$$\text{rango}\left(\left[C^T, A^TC^T, \ldots, \left(A^T\right)^{n-1} C^T \right] \right).$$

Sí A_{22} *es* una matriz *de estabilidad entonces el par* (*A, C*) *es detectable.*

Puede comprobarse que como el par (A_{11}, C_1) *es observable, existe una matriz de realimentación* K_1*, tal que* $A_{11} - C_1K_1$*, es estable. Si* A_{22} *es también estable entonces la matriz de realimentación* $K = [K_1, 0]P^{-1}$ *estabiliza el par* (*A, C*).

La controlabilidad y la observabilidad (estabilizabilidad y detectabilidad) son propiedades duales en el sentido de que el par (*A*, *B*) es controlable (estabilizable) si y sólo si el par (A^T, B^T) es observable (detectable).

1.3.5 Realizaciones minimales

Definición 22 *Una realización (A,B,C) de un SLC (1.9) se dice que es minimal si (A,B) es controlable y (A,C) es observable* [Van Dooren 95].

La identificación de la parte minimal, esto es, de la parte controlable y observable de un SLC, tiene un papel muy importante en la reducción de modelos, debido a que algunas técnicas utilizadas en el espacio de estados [Laub 80, Laub 87, Safonov 89], requieren que el SLC sea minimal.

En consecuencia, la primera etapa en la resolución del problema de reducción de modelos será, en muchos casos, la obtención de la parte o subsistema del SLC que sea controlable y observable. Para ello se han desarrollado algoritmos que permiten dividir las matrices del sistema (reducción a la forma escalera [Van Dooren 79]) de tal forma que sea factible la identificación de este subsistema.

1.3.6 Las matrices Gramian y los valores singulares de Hankel de un SLC

Definición 23 *Si el SLC* (1.3) *es asintóticamente estable entonces las matrices Gramian o Gramianos de controlabilidad (alcanzabilidad) y observabilidad,* W_C y W_O, *se definen, respectivamente, como*

$$W_C = \int_0^{+\infty} e^{tA} B B^T e^{tA^T} dt,$$

$$W_O = \int_0^{+\infty} e^{tA^T} C^T C e^{tA} dt.$$

Si además el par (A,B) es controlable y el par (C,A) es observable las matrices Gramian son solución de las ecuaciones de Lyapunov

$$A W_C + W_C A^T + B B^T = 0,$$
$$A^T W_O + W_O A + C^T C = 0.$$

La definición de las matrices Gramian puede extenderse para los SLC generalizados como el descrito en (1.2)*. En ese caso los Gramianos de controlabilidad y observabilidad se obtienen a partir de las soluciones X e Y de las ecuaciones de Lyapunov generalizadas*

$$A X E^T + E X A^T + B B^T = 0,$$
$$A^T Y E + E^T Y A + C^T C = 0,$$

y de las relaciones

$$W_C = X,$$
$$W_O = E^T Y E.$$

Proposición 6 [Fortuna 92] *Las matrices Gramian de dos SLC equivalentes* (1.3) *y* (1.4), *con vectores de estado* $x(t)$ *y* $\hat{x}(t)$, *están relacionadas por*

$$\hat{W}_C = T W_C T^T \ \text{y} \ \hat{W}_O = T^{-T} W_O T^{-1},$$

donde T es una matriz no singular tal que $x(t) = T\hat{x}(t)$.

Proposición 7 [Fortuna 92] *Los valores propios del producto* $W_C W_O$ *son invariantes bajo cualquier transformación no singular de coordenadas.*

Definición 24 *Si el SLC* (1.3) *es asintóticamente estable, entonces los valores singulares de Hankel de la matriz de transferencia G*(s) *se definen como*

$$\sigma_v(G(s)) = (\lambda_i(W_C W_O))^{1/2} \quad (i = 1,2,\ldots,n).$$

Generalmente se considera que los valores singulares están ordenados en forma decreciente.

1.4 El problema de reducción de modelos

1.4.1 Planteamiento del problema

La reducción de modelos puede abordarse tanto en el dominio del tiempo como en el de la frecuencia. A continuación estudiamos el planteamiento del problema desde ambos puntos de vista.

Supongamos un SLC invariante en el tiempo de orden *n* grande:

$$\begin{aligned} &\dot{x}(t) = Ax(t) + Bu(t), x(0) = x_0, \\ &y(t) = Cx(t), \end{aligned} \tag{1.9}$$

donde $x(t) \in \mathsf{R}^n$, $y(t) \in \mathsf{R}^p$, $u(t) \in \mathsf{R}^m$, y A, B y C son matrices constantes de dimensiones apropiadas. Podemos asumir que el SLC (1.9) se ha obtenido mediante un procedimiento de linealización de un sistema no lineal o de un sistema distribuido mediante procedimientos de aproximación como los de Galerkin o Padé [Barnet 75, Fortuna 97]. Por lo tanto la aproximación del SLC estará gobernada por ecuaciones diferenciales ordinarias de primer orden. Asumiremos también que el SLC está en forma minimal, es decir, es controlable y observable.

Un modelo de orden reducido del anterior tendrá la forma

$$\begin{aligned} &\dot{x}_r(t) = A_r x_r(t) + B_r u(t), x(0) = x_0, \\ &y_r(t) = C_r x_r(t), \end{aligned}$$

siendo $x(t) \in \mathsf{R}^r, y(t) \in \mathsf{R}^p, u(t) \in \mathsf{R}^m$, $r < n$, y A_r, B_r y C_r matrices constantes de dimensiones apropiadas.

En el dominio de la frecuencia [Fortuna 92] el sistema (1.9) está representado por la matriz de transferencia $G(s)$ de dimensión $p \times m$

$$G(s) = \frac{\sum_{i=0}^{n-1} D_i s^i}{\sum_{i=0}^{n} a_i s^i} \quad , \tag{1.10}$$

donde D_i, $i = 0,1,\ldots,n-1$, son matrices constantes $p \times m$ y a_i, $i = 0,1,\ldots,n$, son los coeficientes del polinomio característico del sistema (de la matriz A, si (A,B,C) es una realización de $G(s)$), siendo $a_n = 1$.

La matriz de transferencia del modelo reducido de (1.10) tendrá la forma

$$G_r(s) = \frac{\sum_{i=0}^{r-1} E_i s^i}{\sum_{i=0}^{r} c_i s^i} \quad ,$$

donde E_i, i=0,1,...,r-1, son matrices constantes $p \times m$ y c_i, i=0,1,...,r, los coeficientes del polinomio característico del modelo reducido (de la matriz A_r, si (A_r,B_r,C_r) es una realización de $G_r(s)$), siendo c_r=1 y $r<n$.

1.4.2 Importancia de la reducción de modelos

Las principales razones para obtener modelos de orden reducido [Fortuna 92] son las siguientes:

1. Simplifican la comprensión del sistema.

2. Se reduce el coste computacional en los problemas de simulación.

3. Se requiere menor esfuerzo computacional en el diseño de controladores numéricamente más eficientes.

4. Se obtienen leyes de control más simples.

Especialmente importante es el caso del control de sistemas complejos en ingeniería, dado que la reducción de modelos es primordial para reducir los requerimientos de *hardware*, facilitar el diseño de controladores, en los que aparece la resolución de problemas numéricos particularmente costosos, y en algunos casos, obtener un modelo reducido adecuado para aplicaciones en tiempo real.

Cuando se obtiene un modelo reducido, éste es más simple, pero a la vez más inexacto. El diseñador deberá conocer el impacto que esta reducción produce sobre el comportamiento del sistema, para poder evaluar el tipo y cantidad de reducción posible en cada caso atendiendo a los límites de error permitidos.

1.4.3 Métodos utilizados para la reducción de modelos

Fortuna [Fortuna 92] propone una clasificación de los métodos de reducción de modelos basada en el ámbito de aplicación o dominio donde éstos se intervienen. Así tendremos procedimientos en el dominio de la frecuencia y en el dominio del tiempo. Una clasificación que parece mucho más operativa es la sugerida por Skelton [Skelton 88], que presenta tres categorías de reducción de modelos:

1. Métodos basados en aproximaciones polinomiales (dominio de la frecuencia).

2. Procedimientos de truncamiento del espacio de estados.

3. Técnicas de optimizaciones paramétricas.

Aunque estos métodos aparezcan separados, esta división no excluye la posible combinación de varios procedimientos en ciertos problemas, para la obtención de un modelo reducido. En la Tabla 1 se muestra un esquema de clasificación donde aparecen los procedimientos más usuales agrupados en la forma descrita anteriormente.

Técnicas de Reducción de Modelos

Dominio de la Frecuencia ⟷		**Dominio del Tiempo**
Aproximaciones polinomiales	*Truncamiento de estados*	*Optimizaciones paramétricas*
• Aproximación de Padé. • Fracción Continua. • Aproximación de Routh. • Método basado en la ecuación de estabilidad. • Métodos mixtos. • Aproximación polinomial basada en la consideración de la energía involucrada.	• Método de agregación. • Espacio de estados balanceado (equilibrado). • Perturbación. • Reducción de Modelos por desacoplamiento de coste. • Separación de la escala de tiempo. • Truncamiento de estados involucrando componentes de energía. • Método de Schur.	• Minimización de error. • Aproximación al error basado en la norma de Hankel. • Minimización del error de técnicas de aproximación obtenidas en un primer paso utilizando diferentes métodos.

Tabla 1. *Clasificación de los métodos de reducción de modelos.*

Podemos ver como las aproximaciones polinomiales trabajan en el ámbito de la matriz de transferencia, mientras el truncamiento de estados opera en la representación del espacio de estados. Las optimizaciones paramétricas pueden por otra parte utilizarse tanto en el dominio del tiempo como en el de la frecuencia.

Los métodos de reducción polinomial se aplican en el dominio de la frecuencia y normalmente no necesitan un *cálculo computacional* intensivo. Son utilizados para obtener funciones de

transferencia de orden reducido y el modelo de coeficientes utilizado se escoge de acuerdo con diversos criterios (comparación de momentos y parámetros de Markov entre el modelo original y el reducido). Estos métodos producen SLC reducidos poco precisos.

El segundo tipo de reducción de modelos corresponde a los procedimientos de optimización paramétrica. Son procedimientos secuenciales basados en la minimización de algunos índices definidos apropiadamente, que miden el error o diferencia de comportamiento entre el modelo original y el de orden reducido [Wilson 79]. Si el modelo reducido tiene valores propios constantes puede obtenerse analíticamente; en caso contrario, éste puede obtenerse mediante diversos métodos de aproximación numérica. Tales métodos requieren un alto coste computacional que los hace en muchos casos prohibitivos, sobre todo si el sistema original es de gran tamaño. De especial interés son las optimizaciones basadas en la minimización de la norma de Hankel [Glover 84], definida como la norma de la diferencia entre las matrices de transferencia del SLC original y del SLC reducido.

Por último tenemos los métodos de truncamiento en el espacio de estados, que incluyen todos los procedimientos que involucran una transformación del SLC original en el espacio de estados. Estos métodos están basados en una transformación de la representación en coordenadas de estados del modelo del SLC de orden completo, que permite obtener un modelo reducido que mantenga en la medida de lo posible las propiedades de respuesta temporal, controlabilidad, observabilidad, etc. En comparación con los métodos anteriores, los métodos de truncamiento en el espacio de estados nos permiten una mayor precisión en la representación del sistema reducido que los obtenidos con aproximaciones polinómicas y un menor coste computacional respecto de los métodos basados en los procedimientos de optimización paramétrica. Por todo ello, la investigación de estos métodos de reducción de modelos constituye uno de los campos de mayor actividad científica actualmente. En el siguiente capítulo trataremos de dar una visión más detallada de algunos de ellos y, en particular de aquellos que, por sus características, son más interesantes.

Capítulo 2

Métodos de truncamiento en el espacio de estados

2.1 Introducción

En este capítulo se estudia la resolución del problema de la reducción de modelos mediante los métodos de truncamiento en el espacio de estados. Para ello, presentamos algunos de los métodos de truncamiento que se han desarrollado durante los últimos años [Fortuna 92, Varga 94]. De todos ellos hemos elegido, para un estudio más detallado, aquellos que por las características que imprimen al sistema de orden reducido, la actualidad de su estudio y el interés de la paralelización de los problemas algebraicos subyacentes, merecen especial atención. Así, en la sección 2.2 estudiamos el método de truncamiento basado en la técnica de realizaciones balanceadas, tanto en lazo abierto [Laub 87] como en lazo cerrado [Jonkheere 83]. En la sección 2.3 abordaremos el método de Schur [Safonov 89] y sus variantes [Varga 91]. Por último, en la sección 2.4, analizaremos los métodos anteriormente presentados y la importancia que en ellos presenta la resolución las ecuaciones de Lyapunov en éstos.

Consideremos un SLC continuo e invariante en el tiempo

$$\begin{aligned} &\dot{x}(t) = Ax(t) + Bu(t),\ x(0) = x_0, \\ &y(t) = Cx(t), \end{aligned} \tag{2.1}$$

donde $x(t) \in \mathsf{R}^n$, $u(t) \in \mathsf{R}^m$, $y(t) \in \mathsf{R}^p$, $A \in \mathsf{R}^{nxn}$, $B \in \mathsf{R}^{nxm}$, $C \in \mathsf{R}^{pxn}$. Utilizando una transformación de coordenadas en el espacio de estados S, $\bar{x}(t) = S^{-1}x(t)$, podemos obtener una realización del SLC equivalente

$$\begin{aligned} &\begin{bmatrix} \dot{\bar{x}}_1(t) \\ \dot{\bar{x}}_2(t) \end{bmatrix} = \begin{bmatrix} \overline{A}_{11} & \overline{A}_{12} \\ \overline{A}_{21} & \overline{A}_{22} \end{bmatrix} \begin{bmatrix} \bar{x}_1(t) \\ \bar{x}_2(t) \end{bmatrix} + \begin{bmatrix} \overline{B}_1 \\ \overline{B}_2 \end{bmatrix} u(t),\ \bar{x}(0) = \bar{x}_0 \\ &y(t) = \begin{bmatrix} \overline{C}_1 & \overline{C}_2 \end{bmatrix} \bar{x}(t), \end{aligned} \tag{2.2}$$

con $\bar{x}(t) = \left(\bar{x}_1(t) \quad \bar{x}_2(t)\right)^T$ de la misma dimensión que $x(t)$, y donde el subvector $\bar{x}_1(t)$ contiene los r elementos (variables de estado del SLC) *principales* del nuevo vector de estados. La importancia de las variables de estado contenidas en $\bar{x}_1(t)$ viene dada por que son las que tienen una mayor influencia o repercusión en el comportamiento del SLC. El procedimiento por el que se efectúa la elección de las variables de estado principales del SLC determinará el método de truncamiento en el espacio de estados.

Un procedimiento bastante utilizado para establecer esta división es el método de aproximación basado en la perturbación singular, propuesto por Kokotovic y Sannuti [Kokotovic 86]. Éste es posible cuando el sistema puede ser dividido a priori de forma heurística en dos subsistemas, uno rápido y otro lento. El sistema puede ser representado entonces del siguiente modo,

$$\begin{aligned} \begin{bmatrix} \dot{x}_1(t) \\ \varepsilon\dot{x}_2(t) \end{bmatrix} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u(t), \\ y(t) &= \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}, \end{aligned} \tag{2.3}$$

donde ε es un entero positivo pequeño. En una primera aproximación, si suponemos que ε=0 y que existe el $\lim_{t\to\infty} x_2(t) = x_2$ y es constante, la parte lenta del sistema puede considerarse dominante y el sistema puede expresarse mediante el siguiente modelo de orden reducido

$$\begin{aligned} \dot{x}_1(t) &= \left[A_{11} - A_{12}A_{22}^{-1}A_{21}\right] x_1(t) + \left[B_1 - A_{12}A_{22}^{-1}B_2\right] u(t), \\ y(t) &= \left[C_2 - C_2A_{22}^{-1}A_{21}\right] x_1(t) - \left[C_2A_{22}^{-1}B_2\right] u(t). \end{aligned} \tag{2.4}$$

Otro método es el de agregación [Aoki 78], consistente en obtener el vector de estados del modelo de orden reducido $x_r(t)$ como $x_r(t) = Kx(t)$, donde $K \in \mathbb{R}^{r\times n}$ es la matriz de agregación elegida para proporcionar el sistema en lazo cerrado, que contiene el modelo agregado de bajo orden con las propiedades deseadas. La aplicación de este método, en cuanto al esfuerzo computacional necesario, estará en función del orden del sistema y de las características de sus valores propios.

La teoría de realizaciones balanceadas ha supuesto una contribución significativa en el campo de la aproximación de modelos. Moore introdujo un conjunto de invariantes bajo transformaciones de semejanza [Moore 81], los *modos de segundo orden* del sistema [Mullis 76], que representan el peso de cada una de las variables de estado respecto a la controlabilidad y observabilidad del modelo en lazo abierto, y que corresponden a los cuadrados de los valores singulares de Hankel de la matriz de transferencia del sistema $\sigma_1^2 \geq \sigma_2^2 \geq \ldots \geq \sigma_n^2 > 0$ (ver la definición 25).

La transformación de estados buscada es tal que, en la nueva representación, las matrices Gramian de controlabilidad y observabilidad (Gramianos de controlabilidad y observabilidad, respectivamente) son iguales y diagonales [Laub 80, Pernebo 82],

$$W_O = W_C = \Sigma = diag(\sigma_1, \sigma_2, \ldots, \sigma_n). \tag{2.5}$$

El modelo de orden reducido se obtiene al escoger las r componentes que contribuyen más significativamente al comportamiento del sistema y eliminar el resto

$$\sigma_1^2 \geq \sigma_2^2 \geq \ldots \sigma_r^2 >> \sigma_{r+1}^2 \geq \ldots \geq \sigma_n^2 \geq 0.$$

Posteriormente, Kabamba mostró que los modos de segundo orden no representan completamente la contribución de cada variable de estado, en términos de la magnitud L^2 de la respuesta al impulso [Kabamba 85]. Así, se introduce un nuevo tipo de invariantes denominados *ganancias balanceadas* que permiten evaluar la respuesta al impulso relacionada con la energía asociada a cada variable de estado.

Otro método de truncación del espacio de estados es el basado en la aproximación de la q-covarianza equivalente. Consiste en comparar el comportamiento transitorio y en estado estable del sistema utilizando los parámetros de Markov y los datos de covarianza de salida, respectivamente [Skelton 88].

El método de Schur, propuesto recientemente por Safonov y Chiang, utiliza las bases ortonormales de los subespacios invariantes a la derecha y a la izquierda asociados a los valores propios grandes de la matriz $W_C W_O$, para obtener un modelo reducido que comparte muchas de las propiedades de los modelos internamente balanceados del mismo orden [Safonov 89, Varga 91].

En 1983 Jonckheere y Silverman introducen un nuevo conjunto de invariantes [Jonckheere 83] que dan una medida del grado de contribución de cada variable de estado en el sistema en lazo cerrado, en función del filtrado de Kalman y del control gaussiano-cuadrático-lineal (LQG). Se trata de obtener una realización balanceada en lazo cerrado, frente al lazo abierto de las aproximaciones anteriores. Esta idea ya fue mencionada por Laub en un artículo anterior [Laub 87]. Sea la tripleta (*A,B,C*) una realización minimal de la matriz de transferencia *G*(*s*). La ecuación matricial de Riccati para el filtrado de Kalman y la ecuación matricial de Riccati para el control óptimo lineal-cuadrático son, respectivamente, las siguientes

$$\begin{aligned} AP_K + P_K A^T + BB^T - P_K C^T C P_K = 0, \\ A^T P_C + P_C A + C^T C - P_C B B^T P_C = 0. \end{aligned} \tag{2.6}$$

Para cada una de estas ecuaciones matriciales existe solución, ésta es única y semidefinida positiva. Sean P_{K+} y P_{C+} las soluciones de estas ecuaciones. Los valores propios de la matriz producto $P = P_{K+} P_{C+}$, son invariantes bajo cualquier transformación de coordenadas no singular $\hat{x}(t) = Tx(t)$, es decir, son invariantes bajo transformaciones de semejanza. Los valores propios de P son también reales, no negativos, y se denominan valores característicos del sistema: $\mu_1^{\,2} \geq \mu_2^{\,2} \geq \ldots \geq \mu_n^{\,2} > 0$. En este caso la transformación buscada es tal que las soluciones de las ecuaciones matriciales de Riccati sean matrices diagonales con sus elementos iguales, $P_{K+} = P_{C+} = diag(\mu_1, \mu_2, \ldots, \mu_n)$, dando lugar a una *realización balanceada en lazo cerrado*.

En los últimos años existe un especial interés en el balanceado de sistemas simétricos [Fortuna 88]. La teoría para el estudio de estos sistemas está basada en la utilización de dos nuevos tipos de ecuaciones matriciales, las llamadas ecuaciones de *Gramian* y de *Riccati cruzadas* [Fernando 85].

A continuación abordamos de forma más detallada el estudio de los métodos de truncación basados en realizaciones balanceadas y el método de Schur, sobre los que se centra nuestro trabajo. Éstos se han elegido en función de las características que imprimen al sistema de orden reducido, la actualidad de su estudio y el interés de la paralelización de los problemas algebraicos que en ellos aparecen.

2.2 Realizaciones balanceadas

2.2.1 Sistemas en lazo abierto

Supongamos un SLC continuo invariante en el tiempo

$$\dot{x}(t) = Ax(t) + Bu(t),\ x(0) = x_0\,,$$

$$y(t) = Cx(t),$$

siendo $x(t) \in \mathsf{R}^n, u(t) \in \mathsf{R}^m, y(t) \in \mathsf{R}^p$, $A \in \mathsf{R}^{nxn}$, $B \in \mathsf{R}^{nxm}$, $C \in \mathsf{R}^{pxn}$. Asumimos que el SLC es *asintóticamente estable*, que el par (*A,B*) es *controlable* y que el par (*A,C*) es *observable*.

Como definimos en el apartado 1.3.6, las matrices Gramian (o Gramianos) de controlabilidad y de observabilidad del sistema, W_C y W_O, son simétricas y definidas positivas, como consecuencia del carácter controlable y observable del sistema y satisfacen las siguientes ecuaciones matriciales de Lyapunov en tiempo continuo

$$AW_C + W_C A^T + BB^T = 0,$$

$$A^T W_O + W_O A + C^T C = 0.$$

Si se aplica una transformación de coordenadas en el espacio de estados

$$x(t) = S\hat{x}(t)\,,$$

donde $S \in \mathsf{R}^{n\times n}$ es una matriz invertible, se obtiene la nueva realización del SLC representado por las ecuaciones

$$\dot{\hat{x}} = \hat{A}\hat{x}(t) + \hat{B}u(t), x(0) = x_0\,,$$

$$y(t) = \hat{C}\hat{x}(t)\,,$$

donde $\hat{A} = S^{-1}AS, \hat{B} = S^{-1}B$ y $\hat{C} = CS$.

Es fácil comprobar que las matrices Gramian del nuevo sistema y las del sistema inicial están relacionadas por

$$\hat{W}_C = S^{-1}W_C S^{-T}, \hat{W}_O = S^T W_O S\,.$$

De estas expresiones se puede deducir que:

1) Los valores propios de *A* (*modos del sistema* [Laub 87]) son invariantes bajo la transformación de coordenadas, ya que $\hat{A}$ y *A* son matrices semejantes,

$$\Lambda(A) = \Lambda(\hat{A})\,.$$

2) Los valores propios de las matrices Gramian no son invariantes bajo la transformación de coordenadas, ya que las matrices $\hat{W}_C$ y W_C ($\hat{W}_O$ y W_O) no están relacionados mediante una transformación de semejanza.

3) Los productos de las matrices Gramian satisfacen una relación de semejanza y por tanto sus valores propios (modos de segundo orden del sistema [Mullis 76]) permanecen invariantes frente a la transformación de coordenadas. En efecto

$$\hat{W}_C\hat{W}_O = S^{-1}W_CS^{-T}S^TW_OS = S^{-1}(W_CW_O)S,$$

por lo tanto $$\Lambda(\hat{W}_C\hat{W}_O) = \Lambda(W_CW_O).$$

Nos interesa en particular un tipo de cambio de coordenadas *S* que denominaremos *transformación contragradiente* (TCG) y que definimos seguidamente.

Definición 26 *Una transformación de coordenadas S por la que las nuevas matrices Gramian* $\hat{W}_C$ *y* $\hat{W}_O$ *son diagonales será una transformación contragradiente*.

Veamos ahora qué tipo de transformaciones contragradiente son de interés para nuestro problema.

Puesto que $W_C^T = W_C > 0$ (simétrica definida positiva), existe una matriz ortogonal $V_C \in \mathsf{R}^{n\times n}$ tal que

$$V_C^TW_CV_C = \Lambda_C^2,$$

donde Λ_C es una matriz diagonal y definida positiva. Por otra parte, puesto que $W_O^T = W_O > 0$, existe una matriz ortogonal $V \in \mathsf{R}^{n\times n}$ y una matriz diagonal definida positiva Λ, tales que

$$V^T[(V_C\Lambda_C)^TW_O(V_C\Lambda_C)]V = \Lambda^2.$$

Consideremos a continuación la familia de transformaciones que podemos deducir de la anterior expresión

$$S_k = V_C\Lambda_CV\Lambda^{-k},\ -\infty < k < +\infty.$$

Es fácil comprobar que al aplicar estas transformaciones a las matrices Gramian se obtienen como resultado las siguientes matrices diagonales

$$S_k^{-1}W_CS_k^{-T} = \Lambda^{2k} \quad \text{y} \quad S_k^TW_OS_k = \Lambda^{2-2k},$$

y en consecuencia se trata de transformaciones contragradiente.

Son de especial interés las TCG para los valores de *k* iguales a 0, 1/2 y 1, denominadas por Moore [Moore 81] como :

k=0 Entrada-normal $(\hat{W}_C = I_n,\ \hat{W}_O = \Lambda^2)$,

k=1/2	Internamente-balanceada	$(\hat{W}_C = \hat{W}_O = \Lambda)$,
k=1	Salida-normal	$(\hat{W}_C = \Lambda^2, \hat{W}_O = I_n)$.

Por supuesto también existen unas matrices ortogonales V_O y U, y una matriz diagonal definida positiva $\Gamma(=\Lambda)$ tal que:

$$U^T[(V_O\Lambda_O)^T W_C (V_O\Lambda_O)]U = \Gamma^2$$

donde,

$$V_O^T W_O V_O = \Lambda_O^2.$$

En este caso la familia de *TCG* vendrá dada por

$$S_k = V_O\Lambda_O U\Gamma^{-k}, \quad -\infty < k < +\infty.$$

Podemos observar que en el caso de *entrada-normal* se puede relajar la condición de observabilidad por la de detectabilidad para la construcción de la transformación contragradiente. Es decir W_O puede ser sólo semidefinida positiva. De igual forma ocurre para el caso de *salida-normal* donde se puede relajar la condición de controlabilidad, sustituyéndola por la de estabilizabilidad. Es decir, W_C es semidefinida positiva.

Es fácil comprobar que los valores propios del producto $W_C W_O$ son los elementos diagonales de Λ^2

$$S_k^{-1}(W_C W_O)S_k = \Lambda^2, \quad -\infty < k < +\infty.$$

También es fácil ver que los elementos de Λ son los valores singulares de la matriz de Moore [Moore 81] definida por

$$M = \Lambda_O V_O^T V_C \Lambda_C,$$

es decir, son precisamente los "modos de segundo orden" definidos por Mullis y Roberts [Mullis 76]. Nos interesa el caso *internamente balanceado*, correspondiente a k=1/2. La transformación balanceada, que denominaremos a partir de ahora S, tendrá la siguiente expresión

$$S = V_C\Lambda_C V\Lambda^{-1/2}.$$

En la práctica no es necesario calcular la descomposición en valores propios de W_C y W_O Basta con calcular los factores de Cholesky L_C y L_O de W_C y W_O, respectivamente, y los valores singulares del producto $L_O^T L_C$.

Reconstruyendo las ecuaciones anteriores e incluyendo en ellas los factores de Cholesky de las martrices Gramian, a partir de la descomposición de Cholesky de W_C,

$$W_C = L_C L_C^T,$$

de su descomposición en valores propios

$$W_C = V_C \Lambda_C^2 V_C^T = V_C \Lambda_C \Lambda_C^T V_C^T = (V_C \Lambda_C)(V_C \Lambda_C)^T ,$$

y utilizando la simetría existente entre ambas expresiones, podemos sustituir $V_C \Lambda_C$ por L_C a la hora de construir la transformación contragradiente. Tendremos entonces que existe una matriz ortogonal $V \in \mathsf{R}^{n \times n}$ tal que:

$$V^T [L_C^T W_O L_C] V = \Lambda^2 .$$

A partir de la descomposición de Cholesky de W_O, $W_O = L_O L_O^T$, se cumple que

$$V^T [(L_O^T L_C)^T (L_O^T L_C)] V = \Lambda^2 .$$

De esta expresión podemos ver que existe una matriz ortogonal $U \in \mathsf{R}^{n \times n}$ tal que

$$V^T [(L_O^T L_C)^T U U^T (L_O^T L_C)] V = \Lambda^2$$

y que cumple

$$L_O^T L_C = U \Lambda V^T . \tag{2.7}$$

En consecuencia, podemos definir la transformación contragradiente como

$$S = L_C V \Lambda^{-1/2} ,$$

$$S^{-1} = \Lambda^{-1/2} U^T L_O^T .$$

Las principales etapas del algoritmo para obtener una transformación balanceada son las siguientes [Laub 87]:

Etapa 1: Calcular los factores de Cholesky de las matrices Gramian (W_C y W_O) definidas como las soluciones de las ecuaciones de Lyapunov

$$A W_C + W_C A^T + B B^T = 0,$$

$$A^T W_O + W_O A + C^T C = 0.$$

Es decir,

$$W_C = L_C L_C^T , \qquad W_O = L_O L_O^T ,$$

siendo L_C y L_O matrices triangulares inferiores con elementos diagonales positivos. Nos interesan los factores de Cholesky de los Gramianos ya que se puede obtener la matriz de valores propios de $W_C W_O$, Λ^2, sin necesidad de calcular el producto de los Gramianos.

Etapa 2: Calcular la descomposición en valores singulares (SVD) del producto de los factores de Cholesky [Fernando 88, Heath 86]

$$L_O^T L_C = U\Lambda V^T.$$

Etapa 3: Obtener la transformación balanceada

$$S = L_C V \Lambda^{-1/2},$$

y su inversa

$$S^{-1} = \Lambda^{-1/2} U^T L_O^T.$$

Etapa 4: Obtener las matrices que definen la nueva realización internamente balanceada

$$\hat{A} = S^{-1}AS = \Lambda^{-½} U^T L_O^T A L_C V \Lambda^{-½},$$

$$\hat{B} = S^{-1}B = \Lambda^{-½} U^T L_O^T B,$$

$$\hat{C} = CS = CL_C V \Lambda^{-1/2}.$$

Utilizando las expresiones de la etapa 3 es fácil comprobar que S es una transformación contragradiente para W_C y W_O. Los valores singulares de $L_O^T L_C$ (que son los valores singulares de la matriz de Moore *M)*, coinciden con las raíces cuadradas positivas de los valores propios de $W_C W_O$.

Una vez obtenida la nueva realización internamente balanceada

$$\dot{\hat{x}}(t) = \hat{A}\hat{x}(t) + \hat{B}u(t),$$

$$y(t) = \hat{C}\hat{x}(t),$$

para obtener el modelo reducido de orden r, con $r<n$, se realiza la siguiente partición del sistema [Moore 81]

$$\dot{\hat{x}}(t) = \begin{bmatrix} \hat{A}_{11} & \hat{A}_{12} \\ \hat{A}_{21} & \hat{A}_{22} \end{bmatrix} \hat{x}(t) + \begin{bmatrix} \hat{B}_1 \\ \hat{B}_2 \end{bmatrix} u(t),$$

$$y(t) = \begin{bmatrix} \hat{C}_1 & \hat{C}_2 \end{bmatrix} \hat{x}(t),$$

donde $\hat{A}_{11} \in \mathbf{R}^{rxr}$, $\hat{A}_{22} \in \mathbf{R}^{(n-r)x(n-r)}$, $\hat{B}_1 \in \mathbf{R}^{rxm}$ y $\hat{C}_1 \in \mathbf{R}^{pxr}$. Truncando los *n-r* estados menos controlables y observables obtenemos el modelo reducido cuya matriz de transferencia viene dada por

$$\hat{G}(s) = \hat{C}_1 (I_s - \hat{A}_{11}) \hat{B}_1.$$

Este modelo reducido es estable, minimal y balanceado, con los Gramianos de controlabilidad y observabilidad diagonales e iguales a

$$\Lambda_r = diag(\sigma_1, \sigma_2, \ldots, \sigma_r).$$

El modelo reducido $\hat{G}(s)$, cumple el siguiente límite de la norma-L^∞ del error, en el dominio de la frecuencia [Glover 84], respecto del modelo original $G(s)$,

$$\overline{\sigma}(G(jw) - \hat{G}(jw)) \leq 2 \sum_{i=r+1}^{n} \sigma_i \quad \forall w.$$

El método desarrollado para realizaciones balanceadas de SLC en tiempo continuo, en el modelo de espacio de estados, puede ser aplicado, con pequeñas modificaciones, al caso de tiempo discreto [Laub 87]. A continuación describimos de forma concisa algunas de estas diferencias.

Las ecuaciones del modelo en tiempo discreto serán ahora

$$x_{k+1} = Ax_k + Bu_k, \quad x_0 = x_0,$$

$$y_k = Cx_k,$$

de modo que el par de matrices (*A,B*) es *alcanzable*, el par (*A,C*) es *observable (detectable)* y la matriz *A* es *convergente* (todos sus valores propios tienen módulo menor que uno).

En este caso las matrices Gramian de alcanzabilidad (W_r) y de observabilidad (W_o) [Pernebo 82] satisfacen las siguientes ecuaciones de Lyapunov en tiempo discreto

$$AW_r A^T - W_r + BB^T = 0,$$

$$A^T W_o A - W_o + C^T C = 0.$$

El algoritmo para calcular la transformación contragradiente que nos permite obtener la realización internamente balanceada es básicamente el mismo que hemos estudiado anteriormente y que consta de las etapas *1-4*. La única diferencia vendrá dada por la resolución de la ecuación discreta de Lyapunov que abordaremos en la etapa *1* y de la que obtendremos directamente L_O y L_r [Varga 90, Hammarling 82, Hammarling 91].

2.2.2 Sistemas en lazo cerrado

Sea un SLC continuo invariante en el tiempo definido por las ecuaciones

$$\dot{x}(t) = Ax(t) + Bu(t), \; x(0) = x_0,$$

$$y(t) = Cx(t),$$

siendo $x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m, y(t) \in \mathbb{R}^p, \quad A \in \mathbb{R}^{nxn}, B \in \mathbb{R}^{nxm}, C \in \mathbb{R}^{pxn}$. Asumimos que éste se encuentra en forma minimal (controlable y observable). Entonces, la ecuación de Riccati para el filtrado de Kalman (ERFK) y la ecuación de Riccati para el control óptimo (ERCO) están definidas, respectivamente, del siguiente modo

$$AP_K + P_K A^T + BB^T - P_K C^T C P_K = 0,$$
$$A^T P_C + P_C A + C^T C - P_C BB^T P_C = 0.$$

Que el sistema (A,B,C) sea minimal implica que para cada una de estas ecuaciones [Jonckheere 83], dual una de la otra, existe una única solución definida positiva, estabilizante (definida negativa, antiestabilizante) que denotaremos por P_{K+} (P_{K-}) y P_{C+} (P_{C-}) respectivamente.

Consideremos una transformación de coordenadas en el espacio de estados

$$\hat{x} = Tx\,,$$

donde $T \in \mathsf{R}^{nxn}$ es una matriz invertible. Sean $\hat{P}_{K+}$ y $\hat{P}_{C+}$ las soluciones de las ecuaciones ERFK y ERCO del nuevo sistema. Es fácil comprobar, de forma similar al caso de las realizaciones balanceadas en lazo abierto, que

$$\hat{P}_{K+} = TP_{K+}T^T\,,$$

$$\hat{P}_{C+} = T^{-T}P_{C+}T^{-1},$$

Además, los valores propios del producto $P=P_{K+}P_{C+}$ son invariantes bajo esta transformación [Laub-80], puesto que

$$\hat{P}_{K+}\hat{P}_{C+} = TP_{K+}P_{C+}T^{-1}.$$

También puede comprobarse, mediante manipulaciones algebraicas elementales [Jonkheere 83], que

$$P_{K+} = -P_{C-}^{-1} \text{ y } P_{C+} = -P_{K-}^{-1}\ .$$

Las matrices optimas de filtrado y de control en lazo cerrado definidas, respectivamente, como

$$A^O = A - P_{K+}C^TC,$$
$$A^C = A - BB^TP_{C+},$$

son semejantes, ya que:

$$A^O = (I + P_{K+}P_{C+})A^C(I + P_{K+}P_{C+})^{-1}.$$

La representación en el espacio de estados del compensador óptimo lineal COL ($K(s)$), resultado del acoplamiento de un observador óptimo y de un regulador lineal óptimo, viene dada por

$$\dot{\tilde{x}}(t) = A^K\tilde{x}(t) + B^Ky(t), \qquad \text{(observador óptimo)}$$

$$u(t) = -C^K\tilde{x}(t)\,, \qquad \text{(control de ganancia óptimo)}$$

donde

$$A^K = A - BB^TP_{C+} - P_{K+}C^TC,$$
$$B^K = P_{K+}C^T\,,$$
$$C^K = B^TP_{C+}.$$

Como se ha comentado antes, los valores propios del producto $P=P_{K+}P_{C+}$ son invariantes bajo una transformación de semejanza y además son reales y estrictamente positivos. Si $\mu_1^2 \geq \mu_2^2 \ldots \geq \mu_n^2 > 0$ $(\mu_k > 0)$, son los valores propios, ordenados en sentido decreciente, existe una transformación de semejanza $\hat{T}$

$$(A, B, C) \xrightarrow{\hat{T}} (\hat{A}, \hat{B}, \hat{C}),$$

tal que

$$\hat{P}_{K+} = \hat{P}_{C+} = M = \text{diag}(\mu_1, \mu_2, \ldots, \mu_n).$$

La transformación de semejanza $\hat{T}$ distribuye el peso de las componentes de estado entre dos problemas, uno de control y otro de filtrado, de igual peso ($\hat{P}_{K+} = \hat{P}_{C+}$). Este peso o importancia del estado $\hat{x}_k$ en el problema del COL es μ_k, pues μ_k es al mismo tiempo el error de covarianza del filtrado en la dirección de $\hat{x}_k$ y el coste inducido por una condición inicial alineada con $\hat{x}_k$. Esto nos lleva al problema de compensación, si pensamos en un compensador constituido por la cascada de un observador (o filtro) y un control de ganancia.

Así, μ_k especifica en qué forma, la variable de estado $\hat{x}_k$, participa en el comportamiento en lazo cerrado del sistema. Si μ_k es *grande*, entonces $\hat{x}_k$ es *difícil* de filtrar y de controlar, y por lo tanto debe de tenerse en cuenta en el diseño del compensador. Si μ_k es *pequeño*, $\hat{x}_k$ es *fácil* de filtrar y controlar, por lo que $\hat{x}_k$ no es una variable de estado esencial y puede descartarse en el diseño del compensador reducido [Jonckheere 83].

De acuerdo con las consideraciones anteriores, se propone el siguiente método de actuación: La matriz *M* se divide en dos bloques diagonales

$$M = \begin{pmatrix} M_{11} & 0 \\ 0 & M_{22} \end{pmatrix} \begin{matrix} r \\ n-r \end{matrix},$$
$$\begin{matrix} r & n-r \end{matrix}$$

donde M_{11} es mucho mayor que M_{22}, es decir, $\mu_1 \geq \cdots \geq \mu_r >> \mu_{r+1} \geq \cdots \geq \mu_n > 0$.

Dividimos las ecuaciones del sistema y el compensador óptimo COL de acuerdo con la partición realizada en *M*

$$\dot{\hat{x}}(t) = \begin{pmatrix} \hat{A}_{11} & \hat{A}_{12} \\ \hat{A}_{21} & \hat{A}_{22} \end{pmatrix} \hat{x}(t) + \begin{pmatrix} \hat{B}_1 \\ \hat{B}_2 \end{pmatrix} u(t),$$

$$y(t) = \begin{pmatrix} \hat{C}_1 & \hat{C}_2 \end{pmatrix} \hat{x}(t),$$

$$\dot{\tilde{\hat{x}}}(t) = \begin{pmatrix} \hat{A}_{11}^K & \hat{A}_{12}^K \\ \hat{A}_{21}^K & \hat{A}_{22}^K \end{pmatrix} \tilde{\hat{x}}(t) + \begin{pmatrix} \hat{B}_1^K \\ \hat{B}_2^K \end{pmatrix} y(t),$$

$$u(t) = \begin{pmatrix} \hat{C}_1^K & \hat{C}_2^K \end{pmatrix} \tilde{\hat{x}}(t).$$

El modelo reducido es el representado por ($\hat{A}_{11}, \hat{B}_1, \hat{C}_1$) y su compensador COL óptimo de orden reducido es evidentemente ($\hat{A}_{11}^K, \hat{B}_1^K, \hat{C}_1^K$).

De forma intuitiva se deduce que el compensador de orden reducido garantiza la estabilidad del sistema si M_{22} es lo suficientemente pequeño.

En el caso discreto el procedimiento a seguir es muy similar al descrito para tiempo continuo. Las ecuaciones matriciales ERFK y ERCO serán, respectivamentes

$$AP_1A^T - P_1 + BB^T - P_1C^TCP_1 = 0,$$
$$A^TP_2A - P_2 + C^TC - P_2BB^TP_2 = 0.$$

Existen diversos métodos, todos ellos iterativos, para resolver la ecuación de Riccati [Petkov 91]. Entre ellos el método de Newton (o iteración de Kleinmann) introducido en [Kleinmann 68] es el único método conocido cuya estabilidad numérica ha sido demostrada y en la que en cada iteración hay que resolver una ecuación de Lyapunov. Este método requiere una aproximación inicial a la solución del problema y la convergencia puede ser lenta si esta aproximación está alejada de la misma. En consecuencia, y debido a su alto coste computacional por iteración y lenta convergencia, este método se considera como un algoritmo de refinamiento óptimo para construir algoritmos globales numéricamente estables más rápidos aplicándolo en la última etapa de otros métodos más rápidos (como los basados en la función signo matricial [Roberts 80] o en la descomposición de Schur [Laub 79]) pero cuya estabilidad numérica no está demostrada.

En el método de Newton se obtiene una secuencia de matrices que, bajo ciertas condiciones, converge a la única solución simétrica semidefinida positiva de las ecuaciones de Riccati anteriores [Kleinmann 68]

$$AX + XA^T + BB^T - XC^TCX = 0,$$
$$A^TX + XA + C^TC - XBB^TX = 0.$$

Aunque es posible desarrollar diversos algoritmos, matemáticamente equivalentes, para la resolución de las ecuaciones de Riccati, basados en [Kleinmann 68], el siguiente es uno de los que mejor comportamiento presenta desde el punto de vista de robustez en presencia de errores de redondeo [Benner 94]. Para el caso de la matriz de Riccati

$$A^TX + XA + C^TC - XBB^TX = 0. \tag{2.8}$$

los pasos a seguir serán los siguientes:

1. *Calcular una matriz inicial simétrica* X_0 *tal que* $A - SX_0$, $S = BB^T$, *es estable.*

2. *Para k= 0,1,2,... hasta la convergencia o k > maxit*

   a) $R_k = -(A^TX_k + X_kA + C^TC - X_kSX_k)$.

   b) *Calcular la solución* $\hat{X}_k$ *de la ecuación de Lyapunov en tiempo continuo*

$$(A^T - SX_k)\hat{X}_k + \hat{X}_k(A - SX_k) = R_k.$$

   c) $X_{k+1} = X_k + \hat{X}_k$

Las matrices *A, S* y *Q* no se modifican, mientras que la matriz solución *X* se actualiza en cada iteración mediante una corrección $\hat{X}_k$ decreciente.

El método de Newton puede utilizarse para resolver la ecuación (2.8) obteniendo directamente el factor de Cholesky, *L*, de la matriz solución *X* mediante algunas modificaciones del anterior algoritmo [Hammarling 82b].

2.3 Método de Schur

Este método, propuesto por Safonov y Chiang [Safonov 89] para sistemas en lazo abierto, pretende evitar el cálculo de la transformación balanceada (lo que Varga denomina métodos de raíz cuadrada o **SR** (*square-root methods*)) que puede introducir errores considerables cuando el sistema está próximo a la inobservabilidad y/o la incontrolabilidad. Esta circunstancia fue tratada por Tombs y Postlethwaite [Tombs 87], quienes mostraron que el método de Laub *et al.* [Laub 87] puede utilizarse para calcular directamente las primeras *r* columnas de la transformación balanceada que son usadas para calcular las *r* primeras filas y columnas de la realización balanceada. Sin embargo este método puede estar mal condicionado cuando algunos estados son más controlables que observables y viceversa.

Consideremos inicialmente el siguiente sistema lineal continuo invariante en el tiempo

$$\dot{x}(t) = Ax(t) + Bu(t),\ x(0) = x_0\,,$$

$$y(t) = Cx(t)\,,$$

donde $x(t) \in \mathsf{R}^n, u(t) \in \mathsf{R}^m, y(t) \in \mathsf{R}^p, A \in \mathsf{R}^{nxn}, B \in \mathsf{R}^{nxm}, C \in \mathsf{R}^{pxn}$. Asumimos que el par (*A,B*) es *controlable* y que el par (*A,C*) es *observable* .

Sean W_C y W_O las matrices Gramian de controlabilidad y observabilidad asociadas a este sistema. Es posible reducir el producto de los Gramianos, $W_C W_O$, a la forma real de Schur y obtener los valores propios ordenados en orden ascendente y descendente, respectivamente,

$$Q_a^T W_C W_O Q_a = \begin{bmatrix} \lambda_{a_n} & x & x & \cdots & x \\ 0 & \lambda_{a_{n-1}} & x & \cdots & x \\ \vdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & \lambda_{a_1} \end{bmatrix},$$

$$Q_d^T W_C W_O Q_d = \begin{bmatrix} \lambda_{d_1} & x & x & \cdots & x \\ 0 & \lambda_{d_2} & x & \cdots & x \\ \vdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & \lambda_{d_n} \end{bmatrix},$$

donde,

$$\lambda_{a_i} = \lambda_{d_i} = \sigma_i^2,\ i = 1, \cdots, n\,,$$

siendo Q_a y Q_d matrices ortogonales fruto de la acumulación de las transformaciones ortogonales necesarias para calcular la forma real de Schur correspondiente. A continuación partimos las matrices Q_a y Q_d como sigue

$$Q_a = \underbrace{[Q_{a_1}}_{n-r}, \underbrace{Q_{a_2}]}_{r} \text{ y } Q_d = \underbrace{[Q_{d_1}}_{r}, \underbrace{Q_{d_2}]}_{n-r},$$

donde Q_{d_1} y Q_{a_1} forman, respectivamente, una base ortonormal del espacio de vectores propios a la derecha de $W_C W_O$ asociado a los valores propios grandes $(\sigma_1{}^2, \cdots, \sigma_r{}^2)$ y pequeños $(\sigma_{r+1}{}^2, \cdots, \sigma_n{}^2)$. Las columnas de Q_{a_2} y Q_{d_2} nos dan una descomposición similar a la anterior pero para el espacio de vectores propios a la izquierda.

Si ahora calculamos la descomposición en valores singulares del producto $Q_{a_2}^T Q_{d_1}$

$$Q_{a_2}^T Q_{d_1} = U \Lambda V^T,$$

tenemos que las matrices reales invertibles de dimensiones *n*x*r* que conforman la transformación vienen dadas por las siguientes expresiones:

$$S_i = Q_{a_2} U \Lambda^{-1/2},$$
$$S_d = Q_{d_1} V \Lambda^{-1/2}.$$

El principal inconveniente de este método es que requiere la formación inicial del producto $W_C W_O$ que nos llevará, en algunos casos, a una pérdida de precisión [Varga 90, Varga 91].

Una posible alternativa, sugerida en [Safonov 89], consiste en obtener las matrices que forman la base ortonormal para el espacio de valores propios a la derecha y a la izquierda de $W_C W_O$ a partir de la descomposición en valores singulares del producto de los factores de Cholesky de las matrices Gramian (L_O y L_C) del siguiente modo.

$$L_C L_O = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix}^T,$$

donde $\Lambda_1 \in \mathbb{R}^{rxr}$ y $U_1, V_1 \in \mathbb{R}^{nxr}$. Λ_1 contiene los valores singulares mayores del producto $L_C L_O$ y Λ_2 los valores singulares más pequeños. Como se desprende de [Safonov 89]

$$V_d = L_C U_1,$$

$$V_i = L_O V_1,$$

forman, respectivamente, una base ortonormal para el espacio de vectores propios a la de derecha y a la izquierda de $W_C W_O$ asociados a los valores propios mayores $(\sigma_1{}^2, \cdots, \sigma_r{}^2)$. Si obtenemos ahora la descomposición QR de V_d y V_i,

$$V_d = \begin{bmatrix} Q_{d_1} & Q_{d_2} \end{bmatrix} \begin{bmatrix} R_d \\ 0 \end{bmatrix},$$
$$V_i = \begin{bmatrix} Q_{i_1} & Q_{i_2} \end{bmatrix} \begin{bmatrix} R_i \\ 0 \end{bmatrix},$$

donde $Q_{d_1}, Q_{i_1} \in \mathsf{R}^{nxr}$ y calculamos la descomposición en valores singulares del producto

$$Q_{i_1}^T Q_{d_1} = U_E \Lambda V_E{}^T,$$

tenemos que las matrices reales invertibles de dimensiones $n \times r$ que conforman la nueva transformación, vienen dadas por las expresiones:

$$S_i = Q_{i_1} U_E \Lambda^{-1/2},$$
$$S_d = Q_{d_1} V_E \Lambda^{-1/2}.$$

Las matrices de la nueva realización reducida de orden *r* son

$$\hat{A} = S_i^T A S_d, \quad \hat{B} = S_i^T B \quad \text{y} \quad \hat{C} = C S_d,$$

con los siguientes Gramianos de controlabilidad y observabilidad

$$\hat{W}_C = S_i^T W_C S_i \quad \text{y} \quad \hat{W}_O = S_d^T W_O S_d.$$

Aunque el modelo obtenido no está balanceado (lo que Varga denomina métodos sin balanceado o **BF** (*balancing-free methods*)) posee muchas de las propiedades propias del modelo balanceado del mismo orden (por ejemplo, ambos poseen los mismos polos).

2.4 Conclusiones

La reducción de modelos supone un compromiso entre el orden del modelo y la adecuación de éste a las características de nuestro sistema. Como apunta Moore [Moore 81] no existe ningún método de reducción que sirva para todos los sistemas, dado que muchas de las características del sistema dependen en gran medida de la aplicación.

En este capítulo hemos descrito algunas de las técnicas utilizadas en la reducción de modelos mediante truncamiento en el espacio de estados y nos hemos centrado en dos de las más importantes y representativas para su estudio detallado. Esto no quiere decir que éstas técnicas se utilicen de forma exclusiva, sino que a menudo se combinan dos o más técnicas de reducción de modelos para conseguir un modelo reducido.

Los métodos de reducción basados en realizaciones balanceadas y la forma real de Schur son de especial interés por estar basados en transformaciones ortogonales, lo que les otorga una importante estabilidad numérica. Por otra parte el modelo reducido que generan estos métodos mantiene casi inalteradas muchas de las propiedades del sistema original.

Como hemos podido ver en éste capítulo, la resolución de las ecuaciones de Lyapunov juega un papel clave en los métodos de reducción que se han tratado con mayor detalle. Estas ecuaciones deben resolverse para obtener las matrices Gramian o su factor de Cholesky. Además, las ecuaciones

que hay que resolver están acopladas, es decir, poseen una matriz, la de coeficientes, que las relaciona directamente. Por todo ello será importante obtener algoritmos que resuelvan estas ecuaciones con la mayor exactitud y en el menor tiempo posible. El resto de este trabajo estará dedicado a obtener estos objetivos mediante la utilización de los métodos numéricos más adecuados en cada caso y el uso de las técnicas de programación paralela y distribuida más actuales.

Capítulo 3

Métodos para la resolución de las ecuaciones de Lyapunov

3.1 Introducción

En este capítulo estudiaremos los métodos más utilizados para resolver la ecuación de Lyapunov en tiempo continuo

$$A^T X + XA + Q = 0\,, \tag{3.1}$$

y en tiempo discreto (también denominada ecuación de Stein simétrica)

$$A^T XA - X + Q = 0\,, \tag{3.2}$$

donde $A \in \mathsf{R}^{n\times n}$ es la matriz de coeficientes, $Q = Q^T \in \mathsf{R}^{n\times n}$ es la matriz de términos independientes y $X = X^T \in \mathsf{R}^{n\times n}$ es la matriz de incógnitas.

En esta sección planteamos el problema de la resolución de las ecuaciones de Lyapunov y su resolución mediante su transformación en un sistema de ecuaciones lineales; en la sección 3.2 presentamos un estudio del condicionamiento de estas ecuaciones ligadas a la resolución del problema de la reducción de modelos. A continuación se estudian los métodos más eficientes para resolver estas ecuaciones. Así, en la sección 3.3 estudiamos el método de Bartels-Stewart, uno de los primeros métodos desarrollados. En la sección 3.4 se analiza el método de Hammarling, desarrollado a partir del anterior, pero que permite obtener directamente el factor de Cholesky de la solución de las ecuaciones de Lyapunov. La sección 3.5 está dedicada al método de la función signo matricial, diseñada para resolver ecuaciones matriciales de Riccati, y que adaptaremos para las ecuaciones de Lyapunov.

La ecuación de Lyapunov en tiempo continuo es un caso particular de la ecuación de Sylvester

$$AX + XB + C = 0, \tag{3.3}$$

donde $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times m}$ son las matrices de coeficientes, $C \in \mathbb{R}^{n\times m}$ es la matriz de términos independientes y $X \in \mathbb{R}^{n\times m}$ es la matriz de incógnitas.

La ecuación de Sylvester (3.3) tiene solución y además es única cuando las matrices A y $-B$ no tienen valores propios en común, es decir,

$$\lambda_i + \mu_j \neq 0 ,$$

donde λ_i, $i = 1,2,\ldots,n$, son los valores propios de A y $\mu_i, i = 1,2,\ldots,m$, son los valores propios de B. En el caso de la ecuación de Lyapunov esta condición se reduce a

$$\lambda_i - \lambda_j \neq 0,\ i \neq j .$$

La ecuación de Sylvester (3.3) puede representarse como un sistema de mn ecuaciones para las mn incógnitas de X de la forma

$$Tx = c , \tag{3.4}$$

donde c = vec(C), x = vec(X) y $T = I_m \otimes A + B^T \otimes I_n$ es una matriz de dimensión $mn \times mn$. El sistema de ecuaciones (3.4) tiene solución si la matriz T es no singular.

Para la ecuación de Lyapunov (3.1) el sistema de ecuaciones correspondiente será

$$Kx = q , \tag{3.5}$$

donde $K = I_n \otimes A^T + A^T \otimes I_n$ es una matriz $n^2 \times n^2$, q = vec(Q), x = vec(X).

El coste computacional de la resolución de la ecuación (3.5) es de orden n^6 y la necesidad de almacenamiento es de orden n^4.

Si consideramos la ecuación de Lyapunov para tiempo discreto (3.2), ésta puede ser transformada en un sistema de ecuaciones lineales (3.5), donde ahora

$$K = A^T \otimes A^T - I_{2n} ,$$

siendo $K \in \mathbb{R}^{n^2\times n^2}$ no singular si y solo si los valores propios de A cumplen que $\lambda_i \lambda_j \neq 1$.

No es aceptable utilizar este método de resolución para tamaños de n elevados. En las secciones 3.3 a 3.5 de este capítulo presentamos métodos alternativos eficientes para resolver las ecuaciones (3.1) sin necesidad de formar matrices tan grandes, y con un coste aritmético mucho menor.

3.2 Condicionamiento del problema

Cuando resolvemos la ecuación de Sylvester (3.3) en un computador con una precisión ε, los errores de redondeo $\varepsilon\|A\|_F, \varepsilon\|B\|_F, \varepsilon\|C\|_F$ estarán presentes en *A*, *B* y *C*. En consecuencia, en el mejor de los casos, y antes de tener en cuenta el algoritmo utilizado para resolver la ecuación, podemos esperar que la solución calculada $\overline{X}$ satisfaga

$$(A+E)\overline{X} + \overline{X}(B+F) + (C+G) = 0\,, \tag{3.6}$$

donde $\|E\|_F \le \varepsilon\|A\|_F$, $\|F\|_F \le \varepsilon\|B\|_F$, y $\|G\|_F \le \varepsilon\|C\|_F$.

La exactitud de la solución de la ecuación de Sylvester (3.3) depende de la sensibilidad de *X* a las perturbaciones en *A*, *B* y *C*. La influencia de las perturbaciones *E*, *F* y *G* en la solución $\overline{X}$ pueden ser analizadas del siguiente modo. La ecuación de Sylvester (3.6) puede reescribirse como el sistema lineal perturbado

$$(T+Y)\overline{x} = -(c+g)\,,$$

donde

$$T = I_m \otimes A + B^T \otimes I_n\,,$$

$$Y = I_m \otimes E + F^T \otimes I_n\,,$$

$$\overline{x} = \mathrm{vec}(\overline{X})\,,$$

$$g = \mathrm{vec}(G)\,.$$

Utilizando la desigualdad fácilmente comprobable $\|P \otimes Q\|_2 \le \|P\|_2\|Q\|_2$, podemos deducir que

$$\|T\|_2 \le \|A\|_F + \|B\|_F\,,$$

$$\|Y\|_2 \le \|E\|_F + \|F\|_F\,,$$

$$\|c\|_2 = \|C\|_F \le (\|E\|_F + \|F\|_F)\|X\|_F\,,$$

$$\|g\|_2 = \|G\|_F\,.$$

A partir de la definición 5

$$\sigma_{min}(T) = sep(A,-B)\,,$$

tenemos que

$$\left\|T^{-1}\right\|_2 = \frac{1}{\sigma_{min}(T)} = \frac{1}{sep(A,-B)}\,,$$

donde sep(A,-B)>0 si y solo si A y -B no tiene valores propios en común. De aquí se desprende el siguiente teorema.

Teorema 7 [Byers 83] *Si las matrices A y -B no tienen valores propios en común,* $C \neq 0$ *y* $\frac{\|E\|_F + \|F\|_F}{sep(A,-B)} \equiv \delta < 1$, *entonces las perturbaciones relativas en la solución de la ecuación de Sylvester* (3.3) *debidas a E, F y G en las matrices A, B y C satisfacen*

$$\frac{\|\overline{X} - X\|}{\|X\|} \leq \frac{1}{1-\delta} \frac{\|E\|_F + \|F\|_F + \frac{\|G\|_F}{\|X\|_F}}{sep(A,-B)}.$$

En el caso de que ε sea la perturbación relativa en las matrices A, B y C, obtenemos que

$$\delta \equiv \varepsilon \frac{\|A\|_F + \|B\|_F}{sep(A,-B)} < 1; \tag{3.7}$$

entonces

$$\frac{\|\overline{X} - X\|}{\|X\|} \leq \frac{2\varepsilon}{1-\delta} \frac{\|A\|_F + \|B\|_F}{sep(A,-B)} \tag{3.8}$$

y el valor

$$\mathrm{cond}_2(AX + XB + Q) = \frac{\|A\|_F + \|B\|_F}{sep(A,-B)}$$

se considera el *número de condición de la ecuación de Sylvester*.

De las condiciones expuestas en las ecuaciones (3.7) y (3.8) podemos deducir que cuanto más pequeña es la separación entre los valores propios de A y -B, mayor es el cambio de $\overline{X}$ producido por una perturbación en las matrices de datos. Un algoritmo para el cálculo de sep(A,-B) sin necesidad de formar la matriz T de (3.4) fue propuesto en [Byers 83].

El caso de la ecuación de Lyapunov lo podemos tratar como un corolario del teorema 7. Si A^T y -A no tienen valores propios en común, $Q \neq 0$ y $\frac{2\varepsilon\|A\|_F}{sep(A^T,-A)} \equiv \delta < 1$

entonces

$$\frac{\|\overline{X} - X\|}{\|X\|} \leq \frac{4\varepsilon}{1-\delta} \frac{\|A\|_F}{sep(A^T,-A)} = \frac{4\varepsilon}{1-\delta} \mathrm{cond}_2(A^T X + XA + Q).$$

En el caso de la ecuación de Lyapunov en tiempo discreto la sensibilidad depende del valor

$$sep_d(A^T,-A) = \sigma_{min}(A^T \otimes A^T - I_{2n}).$$

Si A es una matriz convergente se tiene que

$$0 < sep_d(A^T, -A) < 1 .$$

Siguiendo el mismo procedimiento que en el caso de tiempo continuo, tenemos que si A no tiene valores propios repetidos, $Q \neq 0$ y $\dfrac{(2\varepsilon + \varepsilon^2)\|F\|_F}{sep_d(A^T, A)} \equiv \delta < 1$, entonces

$$\frac{\|\overline{X} - X\|}{\|X\|} \leq \frac{\varepsilon(3+\varepsilon)}{1-\delta} \frac{\|A\|_F + 1}{sep_d(A^T, -A)} = \frac{\varepsilon(3+\varepsilon)}{1-\delta} \operatorname{cond}_2(A^T XA + X + Q) .$$

Para la ecuación de Lyapunov generalizada

$$A^T XE + E^T XA + Q = 0 , \tag{3.9}$$

donde $A, E \in \mathbb{R}^{n\times n}$, $Q = Q^T \in \mathbb{R}^{n\times n}$, $X = X^T$ y $\Lambda(A) \subset \mathbb{C}^-$, el sistema de ecuaciones correspondiente será

$$Wx = q , \tag{3.10}$$

donde $W = E^T \otimes A^T + A^T \otimes E^T$ es una matriz $n^2 \times n^2$. Del análisis de este sistema lineal se deduce que el número de condición de la ecuación de Lyapunov generalizada viene dado por la expresión

$$\operatorname{cond}_2(A^T XE + E^T XA + Q) = \operatorname{cond}_2(W) = \|W\|_2 \|W^{-1}\|_2 .$$

La sensibilidad o separación de la ecuación de Lyapunov generalizada viene dada por

$$sep((E^{-1}A)^T, -(E^{-1}A)) = \sigma_{min}(W) ,$$

y puede comprobarse que $sep((E^{-1}A)^T, -(E^{-1}A)) = 1 \Big/ \|W^{-1}\|_2$ (ver por ejemplo [Penzl 96]). Por lo tanto, el número de condición para esta ecuación vendrá dado por

$$\operatorname{cond}_2(A^T XE + E^T XA + Q) = \frac{\|W\|_2}{sep((E^{-1}A)^T, -(E^{-1}A))} \approx \frac{2\|A\|_2 \|E\|_2}{sep((E^{-1}A)^T, -(E^{-1}A))} .$$

Los números de condición y separación se utilizarán en el capítulo 6 para juzgar la calidad de las soluciones aproximadas obtenidas por los métodos numéricos testados.

3.3 Método de Bartels-Stewart

El método de Bartels-Stewart [Bartels 72] es el primero método basado en transformaciones ortogonales, que fue desarrollado para la resolución de las ecuaciones de Sylvester y Lyapunov.

A continuación estudiamos los pasos de que consta este método para resolver las ecuaciones de Lyapunov. El primer paso consiste en la reducción ortogonal de las matrices de coeficientes a una forma más simple, la forma real de Schur, mediante el algoritmo iterativo QR [Francis 61].

Consideremos la ecuación de Lyapunov en tiempo continuo

$$A^T X + XA + Q = 0\,, \tag{3.11}$$

donde $A \in \mathbb{R}^{n\times n}$, $Q = Q^T \in \mathbb{R}^{n\times n}$ y $X = X^T$. Reducimos la matriz A a la forma real de Schur mediante la siguiente transformación de semejanza

$$A = U\overline{A}\,U^T\,,$$

donde $U \in \mathbb{R}^{n\times n}$ es una matriz ortogonal y $\overline{A}$ es una matriz triangular superior por bloques. Los bloques diagonales son de orden 1×1 (escalares que se corresponden con valores propios reales de A) ó 2×2 (asociados a pares de valores propios conjugados de A).

De este modo la ecuación de Lyapunov quedará representada del siguiente modo

$$\overline{A}^T \overline{X} + \overline{X}\,\overline{A} + \overline{Q} = 0\,, \tag{3.12}$$

donde $\overline{Q} = U^T Q U$ y $\overline{X} = U^T X U$.

A continuación, dividimos la ecuación (3.12) del siguiente modo

$$\overline{A} = \begin{bmatrix} \overline{A}_{11} & \overline{A}_{12} \\ 0 & \overline{A}_{22} \end{bmatrix},$$

$$\overline{Q} = \begin{bmatrix} \overline{Q}_{11} & \overline{Q}_{12} \\ \overline{Q}_{21} & \overline{Q}_{22} \end{bmatrix}, \tag{3.13}$$

$$\overline{X} = \begin{bmatrix} \overline{X}_{11} & \overline{X}_{12} \\ \overline{X}_{21} & \overline{X}_{22} \end{bmatrix},$$

donde $\overline{A}_{11}$, $\overline{C}_{11}$ y $\overline{X}_{11}$ son bloques diagonales de tamaño 1×1 ó 2×2. De esta partición, y trabajando por bloques, se obtienen las siguientes ecuaciones:

$$\overline{A}_{11}^T \overline{X}_{11} + \overline{X}_{11}\overline{A}_{11} + \overline{Q}_{11} = 0\,, \tag{3.14}$$

$$\overline{A}_{22}^T \overline{X}_{21} + \overline{X}_{21}\overline{A}_{11} + (\overline{A}_{12}^T \overline{X}_{11} + \overline{Q}_{21}) = 0\,, \tag{3.15}$$

$$\overline{A}_{22}^T \overline{X}_{22} + \overline{X}_{22}\overline{A}_{22} + (\overline{A}_{12}^T \overline{X}_{12} + \overline{X}_{21}\overline{A}_{12} + \overline{Q}_{22}) = 0\,. \tag{3.16}$$

El bloque $\overline{X}_{11}$ de la ecuación (3.14) se resuelve directamente si se trata de un escalar. Si es un bloque 2×2, mediante las propiedades del producto de Kronecker, tal y como de describió en la sección 3.1, se obtiene $\overline{X}_{11}$ de un sistema lineal de tres ecuaciones linealmente independientes.

Una vez resuelto el bloque $\overline{X}_{11}$, la ecuación (3.15) puede resolverse como un sistema lineal del que se obtendrá $\overline{X}_{21}$ mediante el algoritmo de Gauss con pivotamiento parcial y eliminación progresiva.

Por último, la ecuación (3.16) es una ecuación de Lyapunov de orden menor *n*-1 ó *n*-2. El tamaño del problema por lo tanto se ha reducido. El mismo procedimiento se aplicará sucesivamente hasta la resolución completa del problema.

Algunas de las operaciones realizadas en el cálculo de $\overline{Q} = U^T C U$ y $\overline{X} = U^T X U$ pueden simplificarse si se tiene en cuenta la estructura simétrica de estas matrices. Sea $Q = Z + Z^T$, donde Z es una matriz triangular superior; entonces se puede comprobar que

$$\overline{Q} = U^T Q U = U^T Z U + (U^T Z U)^T .$$

Por lo tanto sólo es necesario calcular $U^T Z U$ y, puesto que Z es triangular superior, el producto ZU puede realizarse con la mitad de operaciones de las requeridas para el producto QU.

A continuación presentamos un algoritmo que recoge todas los pasos hasta ahora comentados y que resuelve la ecuación de Lyapunov en tiempo continuo

$$A^T X + X A + Q = 0 .$$

Algoritmo 1 *[BSTC]*

1. Calcular la forma real de Schur $\overline{A} = U^T A U$
2. $W = U^T Z U$ siendo $Q = Z + Z^T$
3. Calcular $\overline{Q} = W + W^T$

   /* *p*= nº de bloques diagonales en los que se divide $\overline{A}$ */
4. para *l* = 1, *p*
   - 4.1. para *k* = *l, p*
     - 4.1.1. $\overline{Q}_{kl} = \overline{Q}_{kl} + \sum_{i=1}^{k-1} \overline{A}_{ik}^T \overline{X}_{il}$
     - 4.1.2. Resolver $\overline{A}_{kk}^T \overline{X}_{kl} + \overline{X}_{kl} \overline{A}_{ll} + \overline{Q}_{kl} = 0$ y obtener $\overline{X}_{kl}$

     fin para
   - 4.2. para *j* = *l*+1, *p*
     - 4.2.1. $\overline{X}_{lj} = \overline{X}_{jl}^T$
     - 4.2.2. para *i* = *j*, *p*
       - 4.2.2.1. $\overline{Q}_{ij} = \overline{Q}_{ij} + \overline{X}_{il} \overline{A}_{lj} + \overline{A}_{li}^T \overline{X}_{lj}$

       fin para

     fin para

   fin para
5. $W = U \overline{Y} U^T$ siendo $\overline{X} = \overline{Y} + \overline{Y}^T$

6. Calcular $X = W + W^T$

fin

La ecuación de Lyapunov para tiempo discreto

$$A^T XA - X + Q = 0\,, \tag{3.17}$$

puede resolverse de forma similar a la de tiempo continuo. Consideremos la ecuación de Lyapunov en tiempo discreto

$$\overline{A}^T \overline{X}\, \overline{A} - \overline{X} + \overline{Q} = 0\,, \tag{3.18}$$

donde la matriz $\overline{A} = U^T AU$ está en forma real de Schur, $\overline{Q} = U^T QU$, $\overline{X} = U^T XU$, y una división por bloques de estas matrices como la considerada en (3.13). La ecuación (3.16) puede reescribirse como el siguiente conjunto de ecuaciones

$$\sum_{i=1}^{k} \overline{A}_{ik}^T \left(\sum_{j=1}^{l} \overline{X}_{il} \overline{A}_{jl} \right) - \overline{X}_{kl} + \overline{Q}_{kl} = 0, \;\; l = 1,2,\ldots,p, \;\; k = l, l+1,\ldots,p\,,$$

donde *p* es el número de bloques diagonales 1×1 ó 2×2 en los que se divide $\overline{A}$.

Estas ecuaciones se resuelven sucesivamente para $\overline{X}_{11}, \overline{X}_{12}, \ldots, \overline{X}_{p1}, \overline{X}_{22}, \ldots, \overline{X}_{p2}, \ldots, \overline{X}_{pp}$. Cada bloque $\overline{X}_{kl}$ se obtiene resolviendo una ecuación de Lyapunov de tiempo discreto (*k*=*l*) o una ecuación de Stein (*k*>*l*) de orden uno o dos.

A continuación mostramos el algoritmo que resuelve la ecuación de Lyapunov para tiempo discreto

$$A^T XA - X + Q = 0\,.$$

Algoritmo 2 *[BSTD]*

1. Calcular la forma real de Schur $\overline{A} = U^T AU$
2. $W = U^T ZU$, siendo $Q = Z + Z^T$
3. Calcular $\overline{Q} = W + W^T$
4. para l = 1, p
   4.1. para $k = l, p$
      4.1.1. $\overline{Q}_{kl} = \overline{Q}_{kl} + \overline{A}_{kk}^T \sum_{j=1}^{l-1} \overline{X}_{kj} \overline{A}_{jl} + \sum_{i=1}^{k-1} \overline{A}_{ik}^T \sum_{j=1}^{l} \overline{X}_{ij} \overline{A}_{jl}$
      4.1.2. Resolver $\overline{A}_{kk}^T \overline{X}_{kl} \overline{A}_{ll} + \overline{X}_{kl} + \overline{Q}_{kl} = 0$ y obtener $\overline{X}_{kl}$
      fin para
   fin para

5. $W = U\overline{Y}U^T$ siendo $\overline{X} = \overline{Y} + \overline{Y}^T$

6. Calcular $X = W + W^T$

fin

3.4 Método de Hammarling

El método de Hammarling [Hammarling 82, Hammarling 91, Varga 90] está basado en el método de Bartels-Stewart y permite resolver las ecuaciones de Lyapunov en tiempo continuo

$$A^T X + XA + C^T C = 0 ,$$

y en tiempo discreto

$$A^T XA - X + C^T C = 0 ,$$

obteniendo directamente el factor de Cholesky de la solución X sin la necesidad de realizar explícitamente el producto $C^T C$. Por esta razón, y al estar basado en transformaciones ortogonales, el método de Hammarling consigue una mayor exactitud que el método de Bartels-Stewart [Bartels 72], que como vimos en la sección anterior calcula la solución X, teniendo posteriormente que realizar una descomposición de Cholesky de ésta para obtener su factor de Cholesky.

3.4.1 Resolución de la ecuación de Lyapunov en tiempo continuo

En esta sección estudiamos el problema de la resolución de la ecuación de Lyapunov en tiempo continuo

$$A^T X + XA + C^T C = 0 , \tag{3.19}$$

donde $A \in \mathbb{R}^{n\times n}$ es una matriz estable (todos sus valores propios tienen parte real negativa) y $C \in \mathbb{R}^{m\times n}$, con $m \geq n$, es una matriz de rango completo (n). En el caso de que $m < n$ el método de Hammarling es el mismo con ligeras modificaciones [Hammarling 82]. Si el producto $C^T C$ es una matriz (semi)definida positiva la solución de la ecuación X es (semi)definida positiva.

Estamos interesados en calcular el factor de Cholesky U de la solución X tal que

$$X = U^T U ,$$

donde $U \in \mathbb{R}^{n\times n}$ es una matriz triangular superior. A continuación plantemos un método general para este problema y en los siguientes apartados consideramos los casos en que: (1) todos los valores propios de la matriz A son reales y (2) existen valores propios de A complejos.

Para resolver la ecuación de Lyapunov (3.19), al igual que en el método de Bartels-Stewart, en primer lugar se simplifica la estructura de la matriz coeficiente. Así, es posible calcular una matriz Q ortogonal, tal que

$$A = QSQ^T ,$$

donde S será una matriz triangular superior por bloques, con bloques diagonales 1x1 ó 2x2, según se trate respectivamente de los valores propios reales de A o bloques asociados a los pares de valores propios complejos conjugados. Así la ecuación quedará de la siguiente forma

$$S^T \overline{X} + \overline{X} S = -\overline{C}^T \overline{C} ,$$

donde $\overline{X} = Q^T X Q = \overline{U}^T \overline{U}$, $\overline{U} = UQ$ y $\overline{C} = CQ$.

Utilizando la factorización de Cholesky de $\overline{X}$, $\overline{X} = \overline{U}^T \overline{U}$, la ecuación anterior se convierte en

$$S^T (\overline{U}^T \overline{U}) + (\overline{U}^T \overline{U}) S = -\overline{C}^T \overline{C} . \tag{3.20}$$

Podemos ahora simplificar aún más la ecuación (3.20) si hacemos que la matriz $\overline{C}$ sea triangular. En efecto podemos calcular una matriz ortogonal $P \in \mathbb{R}^{m \times m}$ tal que

$$\overline{C} = P \begin{pmatrix} R \\ 0 \end{pmatrix},$$

donde $R \in \mathbb{R}^{n \times n}$ es triangular superior. De este modo la ecuación (3.20) se convierte en

$$S^T (\overline{U}^T \overline{U}) + (\overline{U}^T \overline{U}) S + R^T R = 0 .$$

A esta ecuación la llamaremos de ahora en adelante, *ecuación reducida de Lyapunov en tiempo continuo*.

A continuación dividimos las matrices S, $\overline{U}$ y R en bloques de la siguiente forma

$$S = \begin{pmatrix} \lambda_{11} & \mathbf{s}^T \\ 0 & S_1 \end{pmatrix}, \quad \overline{U} = \begin{pmatrix} \upsilon_{11} & \mathbf{u}^T \\ 0 & \overline{U}_1 \end{pmatrix}, \quad R = \begin{pmatrix} \gamma_{11} & \mathbf{r}^T \\ 0 & R_1 \end{pmatrix}$$

donde λ_{11} será un escalar (valor propio real de A, $\lambda_{11} = s_{11}$) o un bloque 2x2 (asociado a dos valores propios complejos conjugados) de la forma

$$\lambda_{11} = \begin{pmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{pmatrix}.$$

En el primer caso, también serán escalares $\upsilon_{11} = \overline{u}_{11}$ y $\gamma_{11} = r_{11}$, siendo **s**, **u** y **r** vectores de n-1 elementos. En el segundo caso, serán bloques 2x2 tanto υ_{11} como γ_{11},

$$\upsilon_{11} = \begin{pmatrix} \overline{u}_{11} & \overline{u}_{12} \\ 0 & \overline{u}_{22} \end{pmatrix} \text{ y } \gamma_{11} = \begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{pmatrix},$$

siendo **s**, **u** y **r** bloques de tamaño (n-2)x2.

A partir de la anterior división de las matrices y trabajando por bloques, obtenemos las siguientes ecuaciones:

$$\lambda_{11}^T(\upsilon_{11}^T\upsilon_{11}) + (\upsilon_{11}^T\upsilon_{11})\lambda_{11} = -\gamma_{11}^T\gamma_{11}\,, \tag{3.21}$$

$$S_1^T\mathbf{u} + \mathbf{u}(\upsilon_{11}\lambda_{11}\upsilon_{11}^{-1}) = -\mathbf{r}\alpha - \mathbf{s}\upsilon_{11}^T\,, \tag{3.22}$$

$$S_1^T(\overline{U}_1^T\overline{U}_1) + (\overline{U}_1^T\overline{U}_1)S_1 + R^TR = 0\,, \tag{3.23}$$

donde

$$R^TR = R_1^TR_1 + \mathbf{y}\mathbf{y}^T\,,\ \mathbf{y} = \mathbf{r} - \mathbf{u}\alpha^T \text{ y } \alpha = \gamma_{11}\upsilon_{11}^{-1}.$$

Podemos resolver directamente las ecuaciones (3.21) y (3.22). La ecuación (3.23) se puede tratar como una ecuación de Lyapunov de dimensión menor que la inicial si realizamos algunas operaciones sobre la matriz *R*. El resultado del producto R^TR puede reescribirse como

$$R^TR = \begin{pmatrix} R_1^T & \mathbf{y} \end{pmatrix}\begin{pmatrix} R_1 \\ \mathbf{y}^T \end{pmatrix} = R_1^TR_1 + \mathbf{y}\mathbf{y}^T\,.$$

Si ahora calculamos la descomposición QR de *R* tal que

$$R = \hat{Q}\begin{pmatrix} \hat{R} \\ 0 \end{pmatrix},$$

donde *Q* es una matriz ortogonal y $\hat{R}$ es triangular superior, podemos comprobar que

$$R^TR = \begin{pmatrix} \hat{R}^T & 0 \end{pmatrix}\hat{Q}^T\hat{Q}\begin{pmatrix} \hat{R} \\ 0 \end{pmatrix} = \hat{R}^T\hat{R}\,.$$

De este modo la ecuación (3.23) quedará como sigue

$$S_1^T(\overline{U}_1^T\overline{U}_1) + (\overline{U}_1^T\overline{U}_1)S_1 + \hat{R}^T\hat{R} = 0\,.$$

Podemos observar que se trata de una ecuación de Lyapunov con la misma estructura que la ecuación (3.20) pero de dimensión menor. La solución de esta nueva ecuación la abordaremos igual que la (3.19) y así sucesivamente.

3.4.1.1 Algoritmo para valores propios reales

En el caso de que todos los valores propios de *A* sean reales, la ecuaciones (3.21), (3.22) y (3.23), obtenidas anteriormente, vendrán dadas respectivamente por las siguientes expresiones:

$$2s_{11}\overline{u}_{11}^2 = -r_{11}^2 \tag{3.24}$$

$$(S_1^T + s_{11}I_{n-1})\mathbf{u} = -\mathbf{r}\alpha - \mathbf{s}u_{11}\,, \tag{3.25}$$

$$S_1^T(\overline{U}_1^T\overline{U}_1) + (\overline{U}_1^T\overline{U}_1)S_1 + R^TR = 0\,, \tag{3.26}$$

donde

$$R = \begin{pmatrix} R_1 \\ \mathbf{y}^T \end{pmatrix},\ \mathbf{y} = \mathbf{r} - \mathbf{u}\alpha^T \text{, y } \alpha = r_{11}\overline{u}_{11}^{-1}.$$

Por ejemplo, para n= 6 la estructura de R es

$$R = \begin{pmatrix} x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \\ & & & & x \\ x & x & x & x & x \end{pmatrix}.$$

La ecuación (3.24) se resuelve ahora directamente obteniéndose u_{11}. La ecuación (3.25) es un sistema lineal triangular inferior que resolvemos por eliminación progresiva, obteniendo **u**.

A continuación realizamos la factorización QR de R mediante rotaciones de Givens. Dada la estructura de R, solo son necesarias (n-1) rotaciones. Ello nos permite obtener una ecuación de Lyapunov de dimensión n-1

$$S_1^T (U_1^T U_1) + (U_1^T U_1) S_1 + \hat{R}^T \hat{R} = 0$$

con

$$R = \hat{Q} \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix}.$$

Continuaremos entonces con esta nueva ecuación aplicando las mismas etapas descritas anteriormente hasta obtener el factor de Cholesky de la solución de la ecuación (3.19).

Antes de presentar en lenguaje algorítmico la solución de la ecuación para el caso de valores propios reales, recordaremos la notación que seguiremos en todos ellos para una mayor claridad en la exposición [Golub 89].

A continuación presentamos el algoritmo para valores propios reales. Partimos de la ecuación

$$S_1^T (U_1^T U_1) + (U_1^T U_1) S_1 + R^T R = 0$$

donde $S(1{:}n,1{:}n)$ y $R(1{:}n,1{:}n)$ son matrices triangulares superiores y $U(1{:}n,1{:}n)$ es el factor de Cholesky que queremos calcular. Los elementos no definidos en la ecuación serán variables internas del programa.

Algoritmo 3 [*HTCVPR]*:

para i=1, n

1.Calculo del elemento diagonal de la fila i.

$\alpha = \sqrt{-2S(i,i)}$, $U(i,i) = -R(i,i)/\alpha$

2.Calculo del resto de elementos superdiagonales de la fila *i* (**u**= *U*(*i*,*i*+1:*n*)).

b(i+1:*n) = -*α *R(i,i*+1:*n) - S(i,i*+1:*n)U(i,i)*

para *j*=*i*+*1*, *n*

U(i,j) = b(j)/(S(j,j) + U(i,i))

b(j+1:*n) = b(j*+1:*n) - S(j,j*+1:*n)U(i,j)*

fin para

3. Actualización de la matriz independiente *R* para resolver la siguiente columna.

3.1. Calculo el vector fila **y**.

y(i+1:*n) = R(i,i*+1:*n) -* α*U(i,i*+1:*n)*

3.2. Calculo del factor de Cholesky de la nueva matriz *R*(*i*+1:*n*,*i*+1:*n*).

para *j*=*i*+1, *n*

3.2.1.Calculo de la rotación de Givens tal que:

$$\begin{bmatrix} R(j,j) \\ y(j) \end{bmatrix} \begin{bmatrix} \cos\theta_j & \text{sen}\,\theta_j \\ -\text{sen}\,\theta_j & \cos\theta_j \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$$

3.2.2. Aplicación de las rotaciones de Given al resto de la columna de *R*.

para *k*= *j, n*

$$\begin{bmatrix} R(j,k) \\ y(k) \end{bmatrix} = \begin{bmatrix} R(j,k) \\ y(k) \end{bmatrix} \begin{bmatrix} \cos\theta_j & \text{sen}\,\theta_j \\ -\text{sen}_j & \cos\theta_j \end{bmatrix}$$

fin para

fin para

fin para

fin

3.4.1.2 Algoritmo para valores propios complejos

En el caso más general, la matriz *A* puede tener valores propios complejos que darán lugar a la existencia de bloques diagonales 2×2 en su forma real de Schur. Para resolver este caso consideramos de nuevo a las ecuaciones (3.21)-(3.23).

Existen dos posibilidades para abordar la resolución de esta ecuación (3.21). La primera es utilizar aritmética compleja para triangularizar λ_{11} y después resolver la ecuación por eliminación progresiva. La segunda posibilidad es transformar esta ecuación de Lyapunov de dimensión 2×2, utilizando las propiedades del producto de Kronecker [Barnett 75].

En nuestro caso particular, la ecuación (3.21) se transforma en el siguiente sistema lineal

$$(I_2 \otimes \lambda_{11}^T + \lambda_{11}^T \otimes I_2)\hat{\mathbf{x}} = -\hat{\mathbf{r}}$$

donde

$$(I_2 \otimes \lambda_{11}^T + \lambda_{11}^T \otimes I_2) = \begin{pmatrix} 2s_{11} & s_{21} & s_{21} & 0 \\ s_{12} & s_{11}+s_{22} & 0 & s_{21} \\ s_{12} & 0 & s_{11}+s_{22} & s_{21} \\ 0 & s_{12} & s_{12} & 2s_{22} \end{pmatrix},$$

$$\hat{\mathbf{x}} = \mathrm{vec}(\upsilon_{11}{}^T \upsilon_{11}) = \begin{pmatrix} \overline{u}_{11}{}^2 \\ \overline{u}_{12}\overline{u}_{11} \\ \overline{u}_{12}\overline{u}_{11} \\ \overline{u}_{12}{}^2 + \overline{u}_{22}{}^2 \end{pmatrix} \quad \text{y} \quad \hat{\mathbf{r}} = \mathrm{vec}(\gamma_{11}{}^T \gamma_{11}) = \begin{pmatrix} r_{11}{}^2 \\ r_{12}r_{11} \\ r_{12}r_{11} \\ r_{12}{}^2 + r_{22}{}^2 \end{pmatrix}.$$

Obtenemos entonces un sistema lineal de cuatro ecuaciones con cuatro incógnitas. Como podemos ver, existe redundancia en la segunda y tercera ecuación, por lo que el sistema se puede reducir a tres ecuaciones con tres incógnitas (suprimiendo, por ejemplo, la tercera ecuación y la tercera incógnita). Para obtener la solución de esta ecuación resolvemos este sistema utilizando el método de Gauss con pivotamiento parcial y aplicamos después eliminación progresiva y sustitución regresiva.

En cuanto a la ecuación (3.22), puede observarse que se trata de una ecuación de Sylvester. Las matrices de coeficientes de ésta, S_1 (en la forma real de Schur) y β son cuadradas de orden (n-2) y 2, respectivamente. Es posible resolver una ecuación de este tipo mediante el método de Bartels y Stewart [Bartels 72] o por el de Hessenberg-Schur [Golub-79].

Por otra parte, dado que β es una matriz 2×2, podemos abordar la resolución de esta ecuación transformándola en un sistema lineal, utilizando las propiedades del producto de Kronecker [Barnett 75], cuya resolución no supone un elevado coste.

Así, si llamamos

$$\mathbf{u} = \begin{pmatrix} \overline{u}_{13} & \overline{u}_{23} \\ \overline{u}_{14} & \overline{u}_{24} \\ \bullet & \bullet \\ \bullet & \bullet \\ \overline{u}_{1n} & \overline{u}_{2n} \end{pmatrix} \text{ y } \mathbf{c} = -\mathbf{r}\alpha - \mathbf{s}u_{11}^T = \begin{pmatrix} \overline{c}_{13} & \overline{c}_{23} \\ \overline{c}_{14} & \overline{c}_{24} \\ \bullet & \bullet \\ \bullet & \bullet \\ \overline{c}_{1n} & \overline{c}_{2n} \end{pmatrix},$$

tenemos que la ecuación (3.22) se transforma en el siguiente sistema

$$(I_2 \otimes S_1^T + \beta^T \otimes I_{2(n-2)})\hat{\mathbf{u}} = \hat{\mathbf{c}},$$

donde

$$\hat{\mathbf{u}} = \mathrm{vec}(\mathbf{u}) = \begin{pmatrix} \overline{u}_{13} \\ \overline{u}_{14} \\ \vdots \\ \overline{u}_{1n} \\ \overline{u}_{23} \\ \overline{u}_{24} \\ \vdots \\ \overline{u}_{2n} \end{pmatrix} \qquad \text{y} \qquad \hat{\mathbf{c}} = \mathrm{vec}(\mathbf{c}) = \begin{pmatrix} \overline{c}_{13} \\ \overline{c}_{14} \\ \vdots \\ \overline{c}_{1n} \\ \overline{c}_{23} \\ \overline{c}_{24} \\ \vdots \\ \overline{c}_{2n} \end{pmatrix}.$$

Resolvemos este sistema por el método de Gauss con pivotamiento parcial y eliminación progresiva y sustitución regresiva. Debido a que existe una gran cantidad de elementos nulos en este sistema de ecuaciones, será interesante aprovechar esta circunstancia para obtener una menor complejidad de cálculo.

Por último nos queda la ecuación (3.23). Actuamos ahora como en el caso real pero con la salvedad de que **y** es una matriz $(n-2)\times 2$ por lo que la matriz que debemos triangularizar para obtener la nueva ecuación de Lyapunov es de dimensión $n\times(n-2)$. Por ejemplo, en el caso de que n=7, ésta será de la forma

$$R = \begin{pmatrix} x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \\ & & & & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}.$$

Para ello utilizaremos rotaciones de Givens. En este caso el número de rotaciones necesarias será $2n$-4.

3.4.2 Resolución de la ecuación de Lyapunov en tiempo discreto

Consideremos ahora el problema de la resolución de la ecuación de Lyapunov en tiempo discreto

$$A^T XA - X + C^T C = 0$$

donde $A \in \mathsf{R}^{n\times n}$ es una matriz convergente (todos sus valores propios tienen módulo menor que uno) y $C \in \mathsf{R}^{m\times n}$, con $m \geq n$, es una matriz de rango completo (n).

Queremos calcular el factor de Cholesky U de la solución X, $X = U^T U$, donde $U \in \mathsf{R}^{n\times n}$ es una matriz triangular superior. Veamos en primer lugar como es posible resolver la ecuación de forma general y a continuación consideraremos los casos en que: (1) todos los valores propios de la matriz A son reales y (2) existen valores propios complejos de A.

Como vimos en el caso en tiempo continuo, comenzamos reduciendo en lo posible las matrices coeficiente de la ecuación. Así, primero reducimos a forma real de Schur la matriz A

$$A = QSQ^T ,$$

donde Q una matriz ortogonal y S una matriz triangular superior por bloques. Los bloques diagonales de S son escalares (bloques 1x1) o bloques 2x2, según estén asociados a valores propios reales o a pares de valores propios complejos conjugados de A, respectivamente. Así la ecuación queda de la siguiente forma

$$S^T \overline{X} S - \overline{X} = -\overline{C}^T \overline{C} ,$$

donde

$$\overline{X} = Q^T X Q \text{ y } \overline{C} = CQ .$$

Teniendo en cuenta la factorización de Cholesky de X y su relación con la de $\overline{X}$ tenemos que

$$\overline{U} = UQ ,$$

quedando la ecuación de la siguiente forma

$$S^T (\overline{U}^T \overline{U}) S - (\overline{U}^T \overline{U}) = -\overline{C}^T \overline{C} . \tag{3.27}$$

Podemos ahora simplificar aún más la ecuación (3.27) si hacemos que la matriz $\overline{C}$, al igual que en el caso continuo, sea triangular. Necesitamos calcular por tanto una matriz ortogonal $P \in \mathbb{R}^{m \times m}$ tal que,

$$\overline{C} = P \begin{pmatrix} R \\ 0 \end{pmatrix},$$

donde $R \in \mathbb{R}^{nxn}$ es triangular superior. De este de modo la ecuación (3.27) se convierte en la *ecuación reducida de Lyapunov en tiempo discreto*

$$S^T (\overline{U}^T \overline{U}) S - (\overline{U}^T \overline{U}) + R^T R = 0 . \tag{3.28}$$

A continuación dividimos las matrices $S, \overline{U}$ y R en los siguientes bloques

$$S = \begin{pmatrix} \lambda_{11} & \mathbf{s}^T \\ 0 & S_1 \end{pmatrix}, \ \overline{U} = \begin{pmatrix} \upsilon_{11} & \mathbf{u}^T \\ 0 & \overline{U}_1 \end{pmatrix}, \ R = \begin{pmatrix} \gamma_{11} & \mathbf{r}^T \\ 0 & R_1 \end{pmatrix},$$

donde λ_{11} será un escalar (valor propio real de A, $\lambda_{11} = s_{11}$) o un bloque 2×2 (asociado a dos valores propios complejos conjugados), de la forma

$$\lambda_{11} = \begin{pmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{pmatrix}.$$

En el primer caso, también serán escalares $\upsilon_{11} = \overline{u}_{11}$ y $\gamma_{11} = r_{11}$, siendo **s**, **u** y **r** vectores de n-1 elementos. En el segundo caso, serán bloques 2×2 tanto υ_{11} como γ_{11},

$$\upsilon_{11} = \begin{pmatrix} \overline{u}_{11} & \overline{u}_{12} \\ 0 & \overline{u}_{22} \end{pmatrix} \text{ y } \gamma_{11} = \begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{pmatrix},$$

siendo **s**, **u** y **r** bloques de tamaño $(n-2)\times 2$.

A partir de la anterior partición de las matrices y trabajando por bloques obtenemos las siguientes ecuaciones

$$\lambda_{11}^T(\upsilon_{11}^T\upsilon_{11})\lambda_{11} - (\upsilon_{11}^T\upsilon_{11}) = -\gamma_{11}^T\gamma_{11}, \tag{3.29}$$

$$S_1^T\mathbf{u}\beta + \mathbf{u} = -\mathbf{r}\alpha - \mathbf{s}\upsilon_{11}^T\beta, \tag{3.30}$$

donde $\alpha = \gamma_{11}\upsilon_{11}^{-1}$ y $\beta = \upsilon_{11}\lambda_{11}\upsilon_{11}^{-1}$, y

$$S_1^T(\overline{U}_1^T\overline{U}_1)S_1 - (\overline{U}_1^T\overline{U}_1) + R^TR = 0, \tag{3.31}$$

donde

$$R^TR = R_1^TR_1 + \mathbf{y}\mathbf{y}^T,\ \mathbf{y}\mathbf{y}^T = \mathbf{r}\mathbf{r}^T - \mathbf{u}\mathbf{u}^T + \mathbf{v}\mathbf{v}^T \text{ y } \mathbf{v} = S_1^T\mathbf{u} + \mathbf{s}\upsilon_{11}^T. \tag{3.32}$$

De estas ecuaciones podemos resolver la (3.29) y la (3.30). La ecuación (3.31) se puede tratar como una ecuación reducida de Lyapunov de dimensión menor que la inicial si triangularizamos la matriz *R*. Por lo tanto, si calculamos una factorización QR de *R* tal que

$$R = \hat{Q}\begin{pmatrix} \hat{R} \\ 0 \end{pmatrix},$$

donde $\hat{Q}$ es una matriz ortogonal y $\hat{R}$ es triangular superior, la ecuación (3.31) quedará como sigue

$$S_1^T(\overline{U}_1^T\overline{U}_1)S_1 - (\overline{U}_1^T\overline{U}_1) + \hat{R}^T\hat{R} = 0.$$

Podemos observar que se trata de una ecuación de Lyapunov con la misma estructura que la ecuación (3.28) pero de dimensión menor. Abordaremos su solución al igual que para (3.28) y así sucesivamente.

3.4.2.1 Algoritmo para valores propios reales

En el caso en que todos los valores propios de la matriz *A* sean reales, el término λ_{11} será un escalar y la ecuación (3.29) quedará como sigue:

$$(s_{11}^2 - 1)\overline{u}_{11}^2 = -r_{11}^2 \tag{3.33}$$

por lo que calculamos directamente el elemento $\overline{u}_{11}$.

La ecuación (3.30) se puede reducir al sistema lineal de ecuaciones

$$(s_{11}S_1^T - I_{n-1})\mathbf{u} = -\mathbf{r}\alpha - \mathbf{s}u_{11}s_{11}, \tag{3.34}$$

donde $\alpha = r_{11}u_{11}^{-1}$. Se trata de un sistema de ecuaciones triangular inferior que resolvemos utilizando eliminación progresiva.

Nos queda por último abordar la ecuación

$$S_1^T(\overline{U}_1^T\overline{U}_1)S_1 - (\overline{U}_1^T\overline{U}_1) + R^TR = 0, \tag{3.35}$$

donde

$$R^TR = R_1^TR_1 + \mathbf{y}\mathbf{y}^T,\ \mathbf{y} = \alpha\mathbf{v} - s_{11}\mathbf{r}\text{, y }\ \mathbf{v} = S_1^T\mathbf{u} + \mathbf{s}u_{11}.$$

En este caso **y** es un vector de *n*-1 elementos y como propusimos en el caso general obtendremos el factor de Cholesky $\hat{R}$ del producto R^TR mediante una descomposición QR

$$R = \begin{pmatrix} R_1 \\ \mathbf{y}^T \end{pmatrix} = \hat{Q}\begin{pmatrix} \hat{R} \\ 0 \end{pmatrix},$$

siendo $\hat{Q}$ una matriz ortogonal y $\hat{R}$ triangular superior. La ecuación (3.35) quedará entonces como sigue

$$S_1^T(\overline{U}_1^T\overline{U}_1)S_1 - (\overline{U}_1^T\overline{U}_1) + \hat{R}^T\hat{R} = 0.$$

Tenemos entonces una ecuación reducida de Lyapunov de dimensión *n*-1 que volveremos a tratar del mismo modo. A continuación presentamos el algoritmo para valores propios reales. Partimos de la ecuación

$$S^T(U^TU)S - (U^TU) + R^TR = 0,$$

donde $S(1{:}n,1{:}n)$ y $R(1{:}n,1{:}n)$ son matrices triangulares superiores y $U(1{:}n,1{:}n)$ el factor de Cholesky que vamos a calcular. Los elementos no definidos en la ecuación serán variables internas del programa.

Algoritmo 4 *[HTDVPR]*

para i= 1, n

1. Cálculo del elemento diagonal de la fila *i.*

   $\alpha = \sqrt{1 - S(i,i)^2}$, $U(i,i) = \alpha^{-1}R(i,i)$

2. Cálculo de los elementos superdiagonales de la fila *i.*

   $b(i{+}1{:}n) = -\alpha R(i,i{+}1{:}n) - S(i,i{+}1{:}n)\ U(i,i)\ S(i,i)$

   para j= i+1, n

   $U(i,j) = b(j)/(S(j,j)S(i,i) - 1)$

   $b(j{+}1{:}n) = b(j{+}1{:}n) - S(j,j{+}1{:}n)U(i,j)S(i,i)$

   fin para

3. Actualización de la matriz *R* para el siguiente paso.

   3.1. Cálculo del vector fila **y**.

para j= i+1, n

$v(j) = S(i+1:j,j)^{\mathrm{T}} U(i,i+1:j)^{\mathrm{T}} + U(i,i)S(i,j)$

fin para

$y(i+1:n) = v(i+1:n)\alpha$ - $S(i,i)R(i,i+1:n)$

3.2. Cálculo y aplicación de las rotaciones de Givens que anulen **y**.

para j= i+1, n

3.2.1. Cálculo de la rotación de Givens tal que:

$$\begin{bmatrix} R(j,j) \\ y(j) \end{bmatrix} \begin{bmatrix} \cos\theta_j & \text{sen}\,\theta_j \\ -\text{sen}\,\theta_j & \cos\theta_j \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$$

3.2.2. Aplicación de las rotaciones de Given al resto de la columna de R.

para k= j, n

$$\begin{bmatrix} R(j,k) \\ y(k) \end{bmatrix} = \begin{bmatrix} R(j,k) \\ y(k) \end{bmatrix} \begin{bmatrix} \cos\theta_j & \text{sen}\,\theta_j \\ -\text{sen}_j & \cos\theta_j \end{bmatrix}$$

fin para

fin para

fin para

fin

3.4.2.2 Algoritmo para valores propios complejos

Estudiaremos en este apartado el caso en que la matriz A pueda tener valores propios complejos que den lugar a la existencia de bloques diagonales 2×2 en su forma real de Schur. Si esto ocurre, la ecuación (3.29) ya no la podemos resolver directamente como en el caso de valores propios reales. Se trata de una ecuación de Lyapunov en tiempo discreto de orden 2 que resolveremos utilizando las propiedades del producto de Kronecker [Barnet 75]. El sistema resultante obtenido a partir de ésta ecuación al aplicar esta transformación será

$$(\lambda_{11}^T \otimes \lambda_{11}^T - I_4)\hat{\mathbf{x}} = -\hat{\mathbf{r}},$$

donde

$$(s_{11}^T \otimes s_{11}^T - I_4) = \begin{pmatrix} s_{11}^2 - 1 & s_{11}s_{21} & s_{21}s_{11} & s_{21}^2 \\ s_{11}s_{12} & s_{11}s_{22} - 1 & s_{21}s_{12} & s_{21}s_{22} \\ s_{12}s_{11} & s_{12}s_{21} & s_{22}s_{11} - 1 & s_{22}s_{21} \\ s_{12}^2 & s_{12}s_{22} & s_{22}s_{12} & s_{22}^2 - 1 \end{pmatrix},$$

$$\hat{\mathbf{x}} = \text{vec}(\upsilon_{11}{}^T \upsilon_{11}) = \begin{pmatrix} \overline{u}_{11}{}^2 \\ \overline{u}_{12}\overline{u}_{11} \\ \overline{u}_{12}\overline{u}_{11} \\ \overline{u}_{12}{}^2 + \overline{u}_{22}{}^2 \end{pmatrix} \qquad \text{y} \qquad \hat{\mathbf{r}} = \text{vec}(\gamma_{11}{}^T \gamma_{11}) = \begin{pmatrix} r_{11}{}^2 \\ r_{12}r_{11} \\ r_{12}r_{11} \\ r_{12}{}^2 + r_{22}{}^2 \end{pmatrix}.$$

Como en el caso continuo, obtenemos un sistema de ecuaciones lineales con redundancia en las ecuaciones segunda y tercera debido a la simetría de la solución. Por tanto lo reducimos a tres ecuaciones con tres incógnitas y procedemos a resolverlo por el método de Gauss con pivotamiento parcial y eliminación progresiva y sustitución regresiva.

En cuanto a la ecuación (3.30) puede observarse que se trata de una ecuación de Stein. Las matrices de coeficientes de ésta, S_1 (en la forma real de Schur) y β son cuadradas de orden (n-2) y 2, respectivamente. Es posible resolver una ecuación de este tipo mediante el método de Bartels y Stewart [Bartels 72] o por el de Hessenberg-Schur [Golub 79].

Por otra parte, dado que β es una matriz 2×2, podemos abordar la resolución de esta ecuación transformándola en un sistema lineal, utilizando las propiedades del producto de Kronecker [Barnett 75], cuya resolución no supone un elevado coste.

Así, si denotamos

$$\mathbf{u} = \begin{pmatrix} \overline{u}_{13} & \overline{u}_{23} \\ \overline{u}_{14} & \overline{u}_{24} \\ \bullet & \bullet \\ \bullet & \bullet \\ \overline{u}_{1n} & \overline{u}_{2n} \end{pmatrix} \quad \text{y} \quad \mathbf{c} = -\mathbf{r}\alpha - \mathbf{s}u_{11}^T\beta = \begin{pmatrix} \overline{c}_{13} & \overline{c}_{23} \\ \overline{c}_{14} & \overline{c}_{24} \\ \bullet & \bullet \\ \bullet & \bullet \\ \overline{c}_{1n} & \overline{c}_{2n} \end{pmatrix},$$

tenemos que la ecuación (3.30) se transforma en el sistema lineal

$$(\beta^T \otimes S_1^T - I_{2(n-2)})\hat{\mathbf{u}} = -\hat{\mathbf{c}},$$

donde $\hat{\mathbf{u}} = vec(\mathbf{u})$ y $\hat{\mathbf{c}} = vec(\mathbf{c})$. Este sistema de 2$n$-4 ecuaciones se resolverá mediante el método de Gauss con pivotamiento parcial y eliminación progresiva y sustitución regresiva.

En la ecuación

$$S_1^T(\overline{U}_1^T\overline{U}_1)S_1 - (\overline{U}_1^T\overline{U}_1) + R^TR = 0,$$

tenemos que calcular el factor de Cholesky $\hat{R}$ del producto R^TR, para poder continuar resolviendo la siguiente ecuación reducida de Lyapunov de orden n-2

$$S_1^T(\overline{U}_1^T\overline{U}_1)S_1 - (\overline{U}_1^T\overline{U}_1) + \hat{R}^T\hat{R} = 0.$$

Veamos ahora como podemos calcular la matriz ***y*** equivalente en el caso complejo. En este caso ***y*** tiene dimensiones $(n-2)\times 2$ y, como podemos observar, es complicado calcularla directamente de las expresiones vistas en (3.32). Un método para obtener la matriz ***y*** ha sido propuesto recientemente por Hammarling [Hammarling 91]. En él, primero se reescribe la ecuación (3.28) de la forma:

$$\overline{U}^T\overline{U} = \begin{pmatrix} R \\ \overline{U}S \end{pmatrix}^T \begin{pmatrix} R \\ \overline{U}S \end{pmatrix}. \tag{3.36}$$

Teniendo en cuenta la partición en bloques que hicimos de las matrices R, $\overline{U}$ y S, inicialmente esta ecuación la reescribiremos como

$$\begin{pmatrix} R \\ \overline{U}S \end{pmatrix} = \begin{pmatrix} \gamma_{11} & \mathbf{r}^T \\ 0 & R_1 \\ \upsilon_{11}\lambda_{11} & \mathbf{v}^T \\ 0 & S_1 U_1 \end{pmatrix}.$$

Si ahora realizamos la factorización QR dada por

$$\begin{pmatrix} \gamma_{11} & \mathbf{r}^T \\ \upsilon_{11}\lambda_{11} & \mathbf{v}^T \end{pmatrix} = \widetilde{Q}\begin{pmatrix} t_{11} & \mathbf{p}^T \\ 0 & \mathbf{z}^T \end{pmatrix} \tag{3.37}$$

y realizamos el producto indicado en la ecuación (3.36), tenemos la siguiente igualdad

$$\begin{pmatrix} \upsilon_{11}^T\upsilon_{11} & \upsilon_{11}^T\mathbf{u}^T \\ \mathbf{u}\upsilon_{11} & \mathbf{u}\mathbf{u}^T + U_1^T U_1 \end{pmatrix} = \begin{pmatrix} t_{11}^T t_{11} & t_{11}^T\mathbf{p}^T \\ \mathbf{p}t_{11} & \mathbf{p}\mathbf{p}^T + R_1^T R_1 + \mathbf{z}\mathbf{z}^T + S_1^T U_1^T U_1 S_1 \end{pmatrix}.$$

Desarrollando la ecuación anterior obtenemos las tres ecuaciones siguientes

$$\upsilon_{11}^T\upsilon_{11} = t_{11}^T t_{11}, \tag{3.38}$$

$$\mathbf{u}u_{11} = \mathbf{p}t_{11}, \tag{3.39}$$

$$\mathbf{u}\mathbf{u}^T + U_1^T U_1 = \mathbf{p}\mathbf{p}^T + R_1^T R_1 + \mathbf{z}\mathbf{z}^T + S_1^T U_1^T U_1 S_1. \tag{3.40}$$

De la ecuación (3.38) y (3.39) se deduce directamente que

$$\mathbf{u}\mathbf{u}^T = \mathbf{p}\mathbf{p}^T.$$

Sustituyendo la igualdad anterior en la ecuación (3.40), ésta quedará como sigue

$$S_1^T(U_1^T U_1)S_1 - (U_1^T U_1) + (R_1^T R_1 + \mathbf{z}\mathbf{z}^T) = 0.$$

Observamos que ***z*** equivale a la ***y*** buscada y tiene dimensiones $(n-2)\times 2$. Ahora ya se puede calcular el factor de Cholesky $\hat{R}$ del término independiente de la ecuación (3.31) y continuar resolviendo la ecuación reducida de Lyapunov de menor dimensión. El cálculo de $\hat{R}$ lo haremos como en los casos anteriores utilizando rotaciones de Givens.

3.5 Método de la función signo matricial

3.5.1 La función signo matricial y la ecuación de Lyapunov

La función signo matricial fue introducida en [Roberts 80] como un método fiable para resolver numéricamente la ecuación algebraica de Riccati (EAR). Actualmente, debido a sus propiedades, existe una amplia corriente de investigación sobre los aspectos numéricos de la función signo matricial y sus aplicaciones.

La función signo matricial puede introducirse como una generalización de la función signo escalar tal como se muestra en la siguiente definición.

Definición 26 *Consideremos la matriz* $A \in \mathbb{R}^{n\times n}$ *y sea*

$$Y^{-1}AY = D + N$$

su descomposición canónica de Jordan, donde Y es la matriz de vectores propios y vectores principales de A, $D = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ *es la matriz diagonal formada por los valores propios de A y N es una matriz nilpotente que conmuta con D.*

La función signo matricial de A se define entonces como

$$\operatorname{signo}(A) = Y \operatorname{diag}(\alpha_1, \alpha_2, \ldots, \alpha_n) Y^{-1} \tag{3.41}$$

donde

$$\alpha_i = \begin{cases} +1 \ \ si \ \ \operatorname{Re}(\lambda_i) > 0, \\ -1 \ \ si \ \ \operatorname{Re}(\lambda_i) < 0. \end{cases}$$

También podemos expresar esta función del siguiente modo

$$\operatorname{signo}(A) = Y \begin{bmatrix} J^- & 0 \\ 0 & J^+ \end{bmatrix} Y^{-1}$$

donde los bloques de Jordan correspondientes a los k valores propios de A con parte real negativa se encuentran en J^- y los bloques de Jordan correspondientes a los (n-k) valores propios de A con parte real positiva se encuentran en J^+.

Si A tiene algún valor propio con parte real nula entonces la función signo matricial no está definida para A [Roberts 80].

A continuación se presentan algunas de las propiedades de la función signo matricial.

Proposición 7 *La función signo matricial de* $A \in \mathbb{R}^{n\times n}$ *satisface*:

1. $\operatorname{signo}(A)^2 = I_n$.
2. $\operatorname{signo}(\alpha A) = \operatorname{signo}(\alpha)\operatorname{signo}(A)$, $\alpha \in \mathbb{R}$.
3. $\operatorname{signo}(TAT^{-1}) = T\operatorname{signo}(A)T^{-1}$, *T no singular*.
4. $A \operatorname{signo}(A) = \operatorname{signo}(A) A$.
5. *Los valores propios de* $\operatorname{signo}(A)$ *son iguales a* ± 1.

La función signo matricial puede aplicarse en la resolución de la EAR,

$$A^T X + XA + Q - XBR^{-1}B^T X = 0\,,$$

como se describe en [Beavers 74, Byers 87, Petkov 91]. A continuación particularizaremos el estudio realizado en las anteriores referencias para el caso de la ecuación de Lyapunov en tiempo continuo

$$A^T X + XA + Q = 0\,, \tag{3.42}$$

donde $A \in \mathsf{R}^{n\times n}, Q = Q^T \in \mathsf{R}^{n\times n}$ y $X = X^T$.

La matriz Hamiltoniana relacionada con la ecuación de Lyapunov (3.42) y definida por

$$\mathcal{H} = \begin{bmatrix} A & 0 \\ -Q & -A^T \end{bmatrix} \in \mathsf{R}^{2n\times 2n},$$

puede descomponerse en

$$\mathcal{H} = \begin{bmatrix} I_n & 0 \\ X & I_n \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & -A^T \end{bmatrix} \begin{bmatrix} I_n & 0 \\ -X & I_n \end{bmatrix},$$

siendo X la solución de la ecuación de Lyapunov. Si los valores propios de A tienen parte real negativa (la matriz A es estable) la función signo matricial de $\mathcal{H}$ está definida. La solución de la ecuación de Lyapunov puede obtenerse mediante el siguiente teorema.

Teorema 8 [Roberts 80] *Consideremos*

$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} = \text{signo}(\mathcal{H})\,,$$

donde $W_{11}, W_{12}, W_{21}, W_{22} \in \mathsf{R}^{n\times n}$.

Se verifica entonces que:

1. *La solución de la ecuación de Lyapunov* (3.42) *satisface que*

$$X = -\tfrac{1}{2} W_{21}\,.$$

2. *La solución X de la ecuación de Lyapunov* (3.42) *satisface el sistema lineal múltiple*

$$\begin{bmatrix} W_{12} \\ W_{22} + I_n \end{bmatrix} X = -\begin{bmatrix} W_{11} + I_n \\ W_{21} \end{bmatrix}. \tag{3.43}$$

A continuación se presenta una breve demostración de los 2 resultados anteriores.

1. La función signo de $\mathcal{H}$ puede descomponerse como

$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} = \text{signo}(\mathcal{H}) = \begin{bmatrix} I_n & 0 \\ X & I_n \end{bmatrix} \begin{bmatrix} -I_n & 0 \\ 0 & I_n \end{bmatrix} \begin{bmatrix} I_n & 0 \\ -X & I_n \end{bmatrix}. \tag{3.44}$$

Multiplicando los términos a la derecha en la expresión (3.44), tenemos que

$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} = \begin{bmatrix} I_n & 0 \\ -2X & I_n \end{bmatrix}$$

de donde se deduce directamente que

$$X = -\tfrac{1}{2} W_{21}.$$

2. Si a la expresión (3.44) le sumamos a ambos lados la matriz identidad I_{2n} tendremos que

$$W + I_{2n} = \begin{bmatrix} W_{11} + I_n & W_{12} \\ W_{21} & W_{22} + I_n \end{bmatrix} = \begin{bmatrix} I_n & 0 \\ X & I_n \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 2I_n \end{bmatrix} \begin{bmatrix} I_n & 0 \\ -X & I_n \end{bmatrix} \tag{3.45}$$

operando adecuadamente tendremos que

$$\begin{bmatrix} W_{11} + I_n + W_{12}X & W_{12} \\ W_{21} + (W_{22} + I_n)X & W_{22} + I_n \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 2I_n \end{bmatrix},$$

y a partir de esta expresión se obtiene directamente

$$\begin{bmatrix} W_{12} \\ W_{22} + I_n \end{bmatrix} X = -\begin{bmatrix} W_{11} + I_n \\ W_{21} \end{bmatrix}.$$

En consecuencia, si $Q>0$ (es decir, definida positiva) la solución definida positiva de la ecuación de Lyapunov se obtiene resolviendo el sistema lineal sobredeterminado, de rango completo descrito en (3.43).

Si $Q \geq 0$ (matriz semidefinida positiva), la solución de la ecuación de Lyapunov es semidefinida positiva y puede obtenerse resolviendo el sistema lineal sobredeterminado, de rango completo, de la ecuación (3.43).

El teorema anterior nos muestra como obtener la solución de la ecuación de Lyapunov a partir de la función signo de $\mathcal{H}$. El siguiente teorema describe un método para el cálculo de la función signo.

Teorema 9 [Roberts 80] *Aplicando el método de Newton a la ecuación matricial* $\text{signo}(\mathcal{H})^2 = I_{2n}$*, se obtiene la siguiente iteración*

$$W_{k+1} = \tfrac{1}{2}(W_k + W_k^{-1}),\ W_0 = \mathcal{H}, \tag{3.46}$$

que converge cuadráticamente a $\text{signo}(\mathcal{H})$.

A partir de la iteración descrita en el teorema 9 se obtiene el siguiente algoritmo para resolver la ecuación de Lyapunov en tiempo continuo.

Algoritmo 5 *[GSH]*

1. Sea $W_0 = \mathcal{H}$ *y* τ *un parámetro de tolerancia que actúa como criterio de convergencia.*

2. para k= 1, 2, ... *hasta la convergencia o* $k >$ *maxit*

2.1. $W_{k+1} = \frac{1}{2}(W_k + W_k^{-1})$.

2.2. si $\|W_{k+1} - W_k\| / \|W_k\| < \tau$ entonces

$p = k+1$, fin bucle.

fin para.

3. Consideremos $$W = W_p = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix},$$

La solución X puede obtenerse de

$$\begin{bmatrix} W_{12} \\ W_{22} + I_n \end{bmatrix} X = -\begin{bmatrix} W_{11} + I_n \\ W_{21} \end{bmatrix},$$

o bien de

$$X = -\frac{1}{2} W_{21}.$$

fin

3.5.2 Análisis de perturbaciones de la función signo matricial

Los estudios realizados hasta la fecha sobre la estabilidad de la función signo matricial generalmente sólo incluyen el análisis de perturbaciones de los subespacios invariantes calculados [Bai 92, Byers 86, Higham 84,Kenney 91b]. De entre estos trabajos, uno de los más recientes y completos es el presentado en [Byers 97].

Si aplicamos los resultados de [Byers 97], basados en el análisis del error progresivo (*forward error*), al cálculo del subespacio invariante de la matriz Hamiltoniana, obtenemos el siguiente teorema.

Teorema 10 [Byers 97] *Consideremos que* $\text{signo}(\mathcal{H}) + F$ *es el resultado obtenido al calcular, con precisión finita, la matriz* $\text{signo}(\mathcal{H})$ *y sea*

$$Q^T \mathcal{H} Q = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix}, \; Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix},$$

la forma real de Schur de $\mathcal{H}$ *donde* Q_1 *y* Q_2 *forman una base del subespacio invariante estable e inestable de* $\mathcal{H}$*, respectivamente. Entonces* $lin(Q_1 + Q_2 W_S)$ *es la versión perturbada del subespacio invariante estable* $lin(Q_1)$*, donde*

$$\|W_S\| \le 4 \frac{\|T_{21}\|}{\|\text{signo}(\mathcal{H})\|_2} \frac{\|\mathcal{H}\|_F}{\text{sep}(T_{11}, T_{22})} + O\left(\frac{\|F\|_2^2}{\|\text{signo}(\mathcal{H})\|_2} \right).$$

Este teorema muestra que la perturbación, W_S, que aparece en el cálculo de un subespacio invariante mediante la función signo matricial, depende de la propia matriz $\mathcal{H}$, la función signo

matricial de ésta, signo($\mathcal{H}$), la perturbación F que se introduce al calcular la función signo matricial y la separación entre los valores propios estables e inestables de $\mathcal{H}$.

Muy recientemente se han obtenido nuevos resultados experimentales [Benner 97] en cuanto a la estabilidad de la función signo matricial y el análisis de perturbaciones de la misma. Estos resultados indican que el método de la función signo matricial para resolver la EAR puede, en la práctica, funcionar como un método numéricamente estable en la mayoría de los casos, mediante el uso de técnicas de escalado y desplazamiento [Gardiner 96].

3.5.3 Aceleración de la convergencia: escalado e iteración de Halley

Aunque la convergencia del esquema iterativo (3.46) es asintóticamente cuadrática, en las iteraciones iniciales puede ser un proceso lento [Higham 84]. En [Balzer 80] se describen diversas técnicas de aceleración para la función signo matricial. La mayoría de las técnicas consisten en un escalado del tipo

$$W_{k+1} = \tfrac{1}{2}(\gamma_k W_k + (\gamma_k W_k)^{-1}),\ W_0 = \mathcal{H}. \tag{3.47}$$

Recientemente se ha demostrado que únicamente se puede aplicar un escalado óptimo en la función signo matricial si el espectro de la matriz W_k es totalmente conocido [Kenney 89].

Sin embargo, se conocen estrategias de escalado casi-óptimas que además son fáciles de usar. La técnica de aceleración más conocida es el escalado mediante el determinante, que fue introducido en [Balzer 80, Byers 87]. La combinación de esta estrategia de escalado y la iteración (3.46) se conoce como la *iteración clásica para la función signo matricial.*

$$\begin{aligned} &W_{k+1} = \tfrac{1}{2}(\gamma_k W_k + (\gamma_k W_k)^{-1}),\ W_0 = \mathcal{H}, \\ &\gamma_k = \left|\det(W_k)\right|^{-1/2n}. \end{aligned} \tag{3.48}$$

Una técnica de aceleración diferente consiste en un escalado espectral [Kenney 89]. En este caso,

$$\gamma_k = \sqrt{\rho(W_k^{-1})/\rho(W_k)},\ \rho(W_k) = max_i\left\{|\lambda_i| : \lambda_i \in \Lambda(W_k)\right\}.$$

En la práctica, el escalado mediante el determinante suele utilizarse hasta que $\left|\det(W_k)\right|^{-1/2n}$ está cercano a la unidad y, a partir de entonces, se recomienda utilizar el escalado espectral [Kenney 89].

El algoritmo correspondiente para resolver la ecuación de Lyapunov mediante el esquema iterativo (3.48), y los esquemas iterativos que se presentarán a continuación, no difiere significativamente del algoritmo *GSH*. Tan sólo es necesario cambiar el esquema iterativo de la etapa *2.1*, y en algún caso modificar el criterio de convergencia en *2.2*.

Una aproximación diferente para el cálculo de la función signo matricial es propuesta en [PKL 90]. Este trabajo está basado en una amplia teoría, desarrollada en [Kenney 91a], sobre expresiones racionales para la función signo matricial de la forma

$$W_{k+1} = W_k P_r(W_k^2) Q_m^{-1}(W_k^2)$$

donde P_r y Q_m son polinomios de grado *r* y *m*, respectivamente. En [Kenney 91a] se demuestra que utilizando los aproximantes de Padé diagonales (*r*=*m*), se obtiene un esquema iterativo que converge globalmente a la función signo matricial e incluye, como caso particular, el esquema iterativo (3.46).

En [Pandey 90] se propone evitar el cálculo secuencial de los términos de los polinomios P_r y Q_m mediante la expansión en fracciones matriciales

$$P_r(W_k^2) Q_m^{-1}(W_k^2) = \sum_{i=1}^{m} w_i (W_k^2 - z_i I_{2n})^{-1} , \tag{3.49}$$

donde los pesos propuestos, w_i y z_i, pueden obtenerse sin error para la función signo matricial a partir de las raíces de los polinomios de Chebyshev [Henrici 74].

A partir de la expansión (3.49), se obtiene la *iteración racional para la función signo matricial*

$$W_{k+1} = W_k \sum_{i=1}^{m} \tfrac{1}{m x_i} (W_k^2 - \alpha_i^2 I_{2n})^{-1} , \;\; W_0 = \mathcal{H} . \tag{3.50}$$

En esta expresión, *m* es el orden del aproximante de Padé utilizado, (*m*-1,*m*),

$$x_i = \tfrac{1}{2}\left(1 + \cos\left(\frac{\pi(2i-1)}{2m}\right)\right) \text{ y } \alpha_i^2 = \tfrac{1}{x_i} - 1 > 0 .$$

Se puede demostrar que $\lim_{k\to\infty} W_k = \text{signo}(\mathcal{H})$ y un criterio de parada para (3.50) es detener el proceso cuando

$$\left\| W_{k+1}^2 - I_{2n} \right\| < \tau .$$

Este criterio de convergencia es conveniente puesto que, en algunos casos, reduce el número de iteraciones del método, y además su evaluación apenas supone un sobrecoste ya que W_{k+1}^2 debe calcularse en cada iteración.

Una particularización de la iteración racional para la función signo matricial es la iteración de Halley definida por

$$W_{k+1} = W_k (3I_{2n} + W_k^2)(I_{2n} + 3W_k^2)^{-1} , \;\; W_0 = \mathcal{H} . \tag{3.51}$$

Como se puede observar, esta iteración tiene un coste computacional mayor que la iteración de Newton. Sin embargo, la convergencia de la iteración de Halley es asintóticamente cúbica.

3.5.4 Iteraciones localmente convergentes

Los esquemas iterativos para la función signo matricial localmente convergentes únicamente aseguran la convergencia cuando se satisfacen algunas propiedades sobre la matriz inicial [Kenney

91a, Kenney 93, Kovarik 70, Leipnik 71]. A cambio, estos esquemas pueden presentar otras ventajas como un menor coste por iteración, una mayor velocidad de convergencia, etc.

Si la matriz Hamiltoniana inicial $W_0 = \mathcal{H}$ (o la aproximación actual W_k a signo($\mathcal{H}$)) está cercana a la función signo matricial y, en concreto, si $\left\| W_{k+1}^2 - I_{2n} \right\| < 1$, la *iteración de Newton-Schulz para la función signo matricial*

$$W_{k+1} = \tfrac{1}{2} W_k (3I_{2n} + W_k^2),\ W_0 = \mathcal{H}, \tag{3.52}$$

converge de forma asintóticamente cuadrática a signo($\mathcal{H}$) [Kenney 91a].

A pesar de su mayor coste por iteración, este esquema iterativo puede ser mucho más eficiente en la mayoría de las arquitecturas actuales de computadores puesto que es un método con abundantes productos matriciales.

La aplicación de este esquema iterativo puede realizarse cuando

$$\left\| W_{k+1} - W_k \right\| < 1.$$

Asimismo, un criterio de convergencia adecuado para la iteración de Newton-Schulz es detener el proceso cuando

$$\left\| W_{k+1} - W_k \right\| / \left\| W_k \right\| < \tau.$$

La combinación del esquema iterativo (3.46) y el esquema iterativo de Newton-Schulz del modo descrito da lugar a la *iteración mixta para la función signo matricial*.

3.5.5 Algoritmo para resolver la ecuación de Lyapunov

En este apartado se describe detalladamente un algoritmo para resolver la ecuación de Lyapunov en tiempo continuo

$$A^T X + XA + Q = 0 \tag{3.53}$$

donde $A \in \mathbf{R}^{n \times n}, Q = Q^T \in \mathbf{R}^{n \times n}$, $X = X^T$ y $\Lambda(A) \subset \mathbf{C}^-$, utilizando la iteración de Newton definida por la ecuación (3.46) para calcular la función signo del Hamiltoniano $\mathcal{H}$.

La iteración de Newton definida como

$$W_{k+1} = \tfrac{1}{2}(W_k + W_k^{-1}),\ W_0 = \mathcal{H},$$

puede reescribirse para el caso de la ecuación de Lyapunov del siguiente modo

$$\begin{bmatrix} A_{k+1} & 0 \\ -Q_{k+1} & -A_{k+1}^T \end{bmatrix} = \frac{1}{2}\left(\begin{bmatrix} A_k & 0 \\ -Q_k & -A_k^T \end{bmatrix} + \begin{bmatrix} A_k^{-1} & 0 \\ -A_k^{-T} Q_k A_k^{-1} & -A_k^{-T} \end{bmatrix} \right),$$

donde $A_0 = A$ y $Q_0 = Q$. Explotando la estructura de la matriz Hamiltoniana en la ecuación anterior, se pueden obtener directamente las siguientes iteraciones [Roberts 80]

$$A_{k+1} = \frac{1}{2}\left(A_k + A_k^{-1}\right),$$

$$Q_{k+1} = \frac{1}{2}\left(Q_k + A_k^{-T} Q_k A_k^{-1}\right). \tag{3.54}$$

A partir del teorema 7 se tiene que

$$X = -\tfrac{1}{2}\left(lim_{k\to\infty} Q_k\right).$$

Además, en la ecuación de Lyapunov en tiempo continuo sabemos que todos los valores propios de *A* tienen parte real negativa y que $A_\infty = \text{signo}(A)$; en consecuencia

$$A_\infty = lim_{k\to\infty} A_k = -I_n,$$

y un posible criterio de parada para las iteraciones puede ser

$$\left\|A_k + I_n\right\| \le tol$$

para una tolerancia *tol* definida por el usuario. Existen otros criterios de parada para la iteración de Newton como $\left\|A_{k+1} - A_k\right\| \le tol \left\|A_{k+1}\right\|$ y $\left\|A_k^2 - I_n\right\| \le tol$ que sin embargo requieren un espacio de trabajo o coste de cálculo adicional.

Aunque para la elección de *tol* se han realizado muchas sugerencias, se ha observado que una decisión habitual es la de tomar $tol = c \cdot n \cdot \varepsilon$, donde ε es la precisión de la máquina, *c* se elige como 10 ó 100 y *n* es el orden de la matriz *A*. Sin embargo, ésta puede llevar a un estancamiento en la iteración debido a que el criterio no se llegue a satisfacer a causa de un mal condicionamiento de la función signo matricial. Por eso la condición de parada anterior puede ser sustituida por $tol = c \cdot n \cdot \sqrt{\varepsilon}$, y realizando una o dos iteraciones adicionales después de que el criterio de parada se haya cumplido. Debido a que la convergencia de la iteración de Newton es asintóticamente cuadrática, normalmente esto es suficiente para alcanzar la precisión deseada.

Para acelerar la convergencia de la iteración de Newton utilizaremos la iteración clásica para la función signo matricial adaptada a la ecuación de Lyapunov y definida por el valor de escalado

$$\gamma = \left|\det(A_k)\right|^{1/n}.$$

A continuación presentamos el algoritmo para resolver la ecuación de Lyapunov en tiempo continuo utilizando la iteración de Newton para el cálculo de la función signo.

Algoritmo 6 *[SILE]*

1. $V = A$, $X = Q$

2. mientras $\left\|V + I_n\right\|_1 > tol$

    2.1. $A = LUP$, mediante la descomposición LU con pivotamiento parcial

2.2. $\gamma = \left|\det(A)\right|^{1/n} = \prod_{k=1}^{n} \left|u_{kk}\right|^{1/n}$

2.3. $W = P^T U^{-1} L^{-1}$, mediante eliminación progresiva y sustitución regresiva

2.4. $A = \frac{1}{2}\left(\frac{1}{\gamma} V + \gamma W\right)$

2.5. $X = \frac{1}{2}\left(\frac{1}{\gamma} X + \gamma W^T X W\right)$

2.6. $V = A$

fin mientras

3. $X = -\frac{1}{2} X$

fin

Al igual que para la iteración de Newton, se pueden obtener iteraciones más simples para las iteraciones de Halley y Newton Schulz.

En el caso de la iteración de Halley, la ecuación (3.51) puede escribirse de la forma

$$A_0 = A,\ A_{k+1} = A_k \left(3I_n - A_k^2\right)\left(I_n + 3A_k^2\right)^{-1},$$

$$Q_0 = Q,\ Q_{k+1} = \left(\left(A_n + 3A_k\right)^T \left(A_k^T Q_k - Q_k A_k\right) + Q_k \left(3I_n + A_k^2\right)\right)\left(I_n + 3A_k^2\right)^{-1}, \qquad (3.55)$$

y para la iteración de Newton-Schulz (3.52), ésta será

$$A_0 = A,\ A_{k+1} = \frac{1}{2} A_k \left(3I_n - A_k^2\right),$$

$$Q_0 = Q,\ Q_{k+1} = \frac{1}{2}\left(Q_k \left(3I_n - A_k^2\right) - A_k^T \left(A_k^T Q_k - Q_k A_k\right)\right). \qquad (3.56)$$

3.5.6 Algoritmo para obtener directamente el factor de Cholesky

La función signo matricial también se puede utilizar para calcular directamente el factor de Cholesky Y de la solución de la ecuación de Lyapunov $X = Y^T Y$ de forma similar a como se realiza en el algoritmo de Hammarling.

Consideremos la ecuación de Lyapunov en tiempo continuo semidefinida positiva

$$A^T X + XA + C^T C = 0,$$

donde $A \in \mathbb{R}^{n\times n}, C \in \mathbb{R}^{m\times n}$, $X = X^T$ y $\Lambda(A) \subset \mathbb{C}^-$. Si aplicamos, como en el apartado anterior, la iteración de Newton para resolver la ecuación, ésta puede reescribirse ahora del siguiente modo

$$\begin{bmatrix} A_{k+1} & 0 \\ -C_{k+1}^T C_{k+1} & -A_{k+1}^T \end{bmatrix} = \frac{1}{2}\left(\begin{bmatrix} A_k & 0 \\ -C_k^T C_k & -A_k^T \end{bmatrix} + \begin{bmatrix} A_k^{-1} & 0 \\ -A_k^{-T} C_k^T C_k A_k^{-1} & -A_k^{-T} \end{bmatrix}\right),$$

donde $A_0 = A$ y $C_0 = C$. De esta ecuación, se pueden obtener directamente las siguientes iteraciones

$$A_{k+1} = \frac{1}{2}\left(A_k + A_k^{-1}\right),$$

$$C_{k+1}^T C_{k+1} = \frac{1}{2}\left(C_k^T C_k + A_k^{-T} C_k^T C_k A_k^{-1}\right) =$$

$$= \tfrac{1}{2}\begin{bmatrix} C_k \\ C_k A_k^{-1} \end{bmatrix}^T \begin{bmatrix} C_k \\ C_k A_k^{-1} \end{bmatrix}.$$

De donde se deduce que en cada iteración la matriz C_k verá incrementado su tamaño por el producto $C_k A_k^{-1}$, por lo que

$$C_{k+1} = \tfrac{1}{\sqrt{2}}\begin{bmatrix} C_k \\ C_k A_k^{-1} \end{bmatrix}.$$

Como se puede constatar, este método requiere que se duplique el espacio de trabajo que se necesita para la iteración sobre C_k. Dado que el rango de la solución X y el de su descomposición de Cholesky Y no puede ser conocido a partir del rango de C, la implementación del algoritmo de Hammarling en [Hammarling 82, Hammarling 91] requiere de una dimensión del espacio de trabajo de al menos $n \times n$ para C, si se quiere que ésta sirva para sobreescribir la matriz Y.

El coste aritmético para la *k*-ésima iteración de C_k es de $2(2^k m)n^2$, donde *m* es el número de filas de C. Comparémoslo con los $3n^3$ para cada iteración de Q_k del apartado anterior. Esto sugiere que compensa incrementar el tamaño de C_k en cada iteración mientras se cumpla que $2^k m$ sea menor que $n/2$, que es el límite para el cual la iteración original (3.54) es menos costosa. Este límite viene dado por la condición

$$k > \left\lfloor \log_2 \frac{n}{m} \right\rfloor, \tag{3.57}$$

donde $\lfloor x \rfloor$ indica la parte entera de x.

Si *k* ha alcanzado el límite anterior, en [Benner 97] se propone construir la matriz

$$C_{k+1} = [C_k^T \quad (C_k A_k^{-1})^T]^T \in \mathbb{R}^{2s_k \times n},$$

donde $C_k \in \mathbb{R}^{s_k \times n}$ y $s_0 = m$, y realizar la descomposición QR tal que

$$C_{k+1} = U_{k+1}\widetilde{R}_{k+1} = U_{k+1}\begin{bmatrix} R_{k+1} \\ 0 \end{bmatrix} \begin{matrix} \} r_{k+1} \\ \} 2s_k - r_{k+1} \end{matrix},$$

donde $r_{k+1} = \text{rango}(\widetilde{C}_{k+1})$. De lo que se deduce que $C_{k+1}^T C_{k+1} = \widetilde{C}_{k+1}^T \widetilde{C}_{k+1}/2 = R_{k+1}^T R_{k+1}/2$ y por lo tanto podemos asignar $C_{k+1} = R_{k+1}/\sqrt{2}$ y $s_{k+1} = r_{k+1}$. Se puede observar que para obtener el factor de

Cholesky de *X*, debe calcularse la factorización QR de C_{k+1} una vez el método ha convergido, aún en el caso de que *k* no alcance el límite expresado en (3.57). Para determinar el rango de $\widetilde{C}_{k+1}$ correctamente, es más adecuado el uso de la factorización QR con pivotamiento [Golub 89] o bien la factorización QR reveladora de rango [Chan 87]. En este caso R_{k+1} se obtiene como parte superior de dimensión $r_{k+1} \times n$ del producto de la matriz triangular superior $\widetilde{R}_{k+1}$ y una matriz de permutación Π_{k+1}, por ejemplo,

$$\widetilde{R}_{k+1}\Pi_{k+1} = \begin{bmatrix} \hat{R}_{k+1} & T_{k+1} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} (\Pi_{k+1})_{11} & (\Pi_{k+1})_{12} \\ (\Pi_{k+1})_{21} & (\Pi_{k+1})_{22} \end{bmatrix} = \begin{bmatrix} R_{k+1} \\ 0 \end{bmatrix}.$$

Otra alternativa es calcular la factorización QR de la matriz aumentada en cada iteración. Como el rango de *X* puede llegar a ser *n*, esta aproximación requiere un espacio de trabajo de tamaño $2n \times n$. Esta aproximación fue apuntada (sin ningún tipo de detalle de implementación) en [Larin 93] y posteriormente desarrollada en [Benner 97].

De esta forma el algoritmo para resolver la ecuación de Lyapunov en tiempo continuo quedará como sigue.

Algoritmo 7 *[SILCE]*

1. $V = A$, $Y = C$, $k = 0$
2. mientras $\|V + I_n\|_1 > tol$
   - 2.1. $k = k + 1$
   - 2.2. $A = LUP$, mediante la descomposición LU con pivotamiento parcial
   - 2.3. $\gamma = \left|\det(A)\right|^{1/n} = \prod_{k=1}^{n} \left|u_{kk}\right|^{1/n}$
   - 2.4. $W = P^T U^{-1} L^{-1}$, mediante eliminación progresiva y sustitución regresiva
   - 2.5. $A = \frac{1}{2}\left(\frac{1}{\gamma} V + \gamma W\right)$
   - 2.6. $\gamma = \sqrt{\gamma}$
   - 2.7. si $k \le \left\lfloor \log_2 \frac{n}{m} \right\rfloor$ entonces

     $$Y = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{\gamma} Y \\ \gamma Y W \end{bmatrix}$$

     sino

     $$\begin{bmatrix} \frac{1}{\gamma} Y \\ \gamma Y W \end{bmatrix} = U \begin{bmatrix} Y \\ 0 \end{bmatrix}$$ mediante una factorización QR

     fin si
   - 2.7. $V = A$

   fin mientras
3. $Y = -\frac{1}{\sqrt{2}} Y$

fin

3.5.7 Algoritmo para resolver la ecuación de Lyapunov generalizada

En este apartado se describe un algoritmo para resolver la ecuación generalizada de Lyapunov en tiempo continuo

$$A^T XE + E^T XA + Q = 0\,, \tag{3.58}$$

donde $A, E \in \mathsf{R}^{n\times n}, Q = Q^T \in \mathsf{R}^{n\times n}$, $X = X^T$ y $\Lambda(A) \subset \mathsf{C}^-$, utilizando una generalización de la iteración de Newton definida por la ecuación (3.46) para la función signo matricial generalizada. La generalización del método de la función signo matricial para un haz de matrices $Z - \lambda Y$ fue propuesta por Gardiner y Laub [Gardiner 86] en el caso de que Z y Y no sean singulares. En concreto, se considera la siguiente iteración

$$\begin{gathered}\gamma_k = \left(\frac{|\det(Z_k)|}{\det(Y)}\right)^{\frac{1}{2}},\\ Z_{k+1} = (Z_k + \gamma_k^2 Y Z_k^{-1} Y)\,,\ Z_0 = Z\,,\end{gathered} \tag{3.59}$$

donde γ_k es un factor de escalado para acelerar la convergencia. Se puede comprobar que esta iteración es equivalente a calcular la función signo de la matriz $Y^{-1}Z$ mediante la iteración de Newton estándar dada en (3.46) (ver [Benner 97] y la bibliografía allí referenciada para más detalles).

Por otra parte, si E es una matriz invertible la ecuación (3.58) puede transformarse en la ecuación de Lyapunov

$$\widetilde{A}^T X + X\widetilde{A} + \widetilde{Q} = 0\,, \tag{3.60}$$

donde $\widetilde{A} = AE^{-1}$ y $\widetilde{Q} = E^{-T} Q E^{-1}$, pudiendo ser tratada igual que en el apartado 3.5.4. Así, se puede calcular la función signo de la matriz Hamiltoniana

$$\widetilde{\mathcal{H}} = \begin{bmatrix} \widetilde{A} & 0 \\ -\widetilde{Q} & -\widetilde{A}^T \end{bmatrix} = \begin{bmatrix} AE^{-1} & 0 \\ -E^{-T} Q E^{-1} & -E^{-T} A^T \end{bmatrix}$$

asociada a la ecuación de Lyapunov (3.60), tal y como se describió en el algoritmo *GSH*.

La iteración de Newton dada por la ecuación

$$W_{k+1} = \tfrac{1}{2}(W_k + W_k^{-1}),\ W_0 = \widetilde{\mathcal{H}}\,,$$

puede reescribirse para el caso de la ecuación generalizada de Lyapunov del siguiente modo

$$\begin{bmatrix} A_{k+1}E^{-1} & 0 \\ -E^{-T}Q_{k+1}E^{-1} & -E^{-T}A_{k+1}^T \end{bmatrix} = \frac{1}{2}\left(\begin{bmatrix} A_k E^{-1} & 0 \\ -E^{-T}Q_k E^{-1} & -A_k^T \end{bmatrix} + \begin{bmatrix} EA_k^{-1} & 0 \\ -A_k^{-T}Q_k A_k^{-1} & -A_k^{-T}E^T \end{bmatrix}\right),$$

donde $A_0 = A$ y $Q_0 = Q$. De esta ecuación, se obtienen directamente las siguientes iteraciones

$$A_{k+1} = \frac{1}{2}\left(A_k + EA_k^{-1}E\right),$$

$$Q_{k+1} = \frac{1}{2}\left(Q_k + E^T A_k^{-T} Q_k A_k^{-1} E\right). \tag{3.61}$$

La solución de la ecuación generalizada de Lyapunov viene dada por

$$X = \tfrac{1}{2}\left(lim_{k\to\infty}\widetilde{Q}_k\right) = \tfrac{1}{2}E^{-T}\left(lim_{k\to\infty}Q_k\right)E^{-1}.$$

Además, la iteración converge para $\widetilde{A}_k$

$$\widetilde{A}_\infty = lim_{k\to\infty}\widetilde{A}_k = (lim_{k\to\infty}A_k)E^{-1} = -I_n,$$

o lo que es lo mismo

$$A_\infty = lim_{k\to\infty}A_k = -E.$$

Esto nos sugiere que el criterio de parada

$$\|A_k + E\| \le tol \cdot \|E\|,$$

donde la tolerancia *tol* está definida por el usuario.

Para acelerar la convergencia de la iteración de Newton utilizaremos una generalización de la iteración clásica para la función signo matricial adaptada a la ecuación generalizada de Lyapunov y definida por un valor de escalado que dependerá del par de matrices (*A,E*)

$$\gamma = \frac{\left|\det(A_k)\right|^{1/n}}{\left|\det(E)\right|^{1/n}}.$$

A continuación presentamos el algoritmo para resolver la ecuación generalizada de Lyapunov en tiempo continuo. En este algoritmo utilizamos una factorización *QL* de *E* para reducir el coste por iteración del método.

Algoritmo 8 *[SILGE]*

1. $E = U_E L_E$ mediante la factorización QR,

   $\gamma_E = \left|\det(E)\right|^{1/n} = \prod_{k=1}^{n}\left|(l_E)_{kk}\right|^{1/n}$,

   $A = U_E A$, $V = A$, $X = Q$

2. mientras $\|V + L_E\|_1 > tol\|L_E\|_1$

   2.1. $A = LUP$, mediante la descomposición LU con pivotamiento parcial

   2.2. $\gamma = \left|\det(A)\right|^{1/n} = \prod_{k=1}^{n}\left|u_{kk}\right|^{1/n}$

   2.3. $W = P^T U^{-1} L^{-1} L_E$, mediante eliminación progresiva y sustitución regresiva

2.4. $\gamma = \gamma_A / \gamma_E$

2.5. $A = \frac{1}{2}\left(\frac{1}{\gamma} V + \gamma W\right)$

2.6. $X = \frac{1}{2}\left(\frac{1}{\gamma} X + \gamma W^T X W\right)$

2.7. $V = A$

fin mientras

3. $X = -\frac{1}{2} L_E^{-T} X L_E^{-1}$ mediante eliminación progresiva

4. $X = U_E X U_E^T$

fin

Siguiendo lo mismos pasos que hemos tomado en este apartado podemos desarrollar un algoritmo para obtener el factor de Cholesky de la solución de la ecuación generalizada de Lyapunov [Benner 97]. A este algoritmo lo denominaremos *SILGCE*.

3.5.8 Algoritmo para resolver ecuaciones de Lyapunov acopladas

En este apartado vamos a proponer un algoritmo para resolver las ecuaciones de Lyapunov en tiempo continuo acopladas

$$A^T X + XA + Q = 0\,, \tag{3.62}$$

$$AY + YA^T + P = 0\,, \tag{3.63}$$

donde $A \in \mathsf{R}^{n\times n}, Q = Q^T \in \mathsf{R}^{p\times n}, P = P^T \in \mathsf{R}^{m\times n}$, $X = X^T$, $Y = Y^T$ y $\Lambda(A) \subset \mathsf{C}^-$.

En el apartado 3.5.5 se propuso resolver la ecuación (3.62) mediante las siguientes iteraciones

$$\begin{aligned} A_{k+1} &= \frac{1}{2}\left(A_k + A_k^{-1}\right), \\ Q_{k+1} &= \frac{1}{2}\left(Q_k + A_k^{-T} Q_k A_k^{-1}\right), \end{aligned} \tag{3.64}$$

donde $A_0 = A$, $Q_0 = Q$, y

$$X = \tfrac{1}{2} Q_\infty = \tfrac{1}{2}\left(lim_{k\to\infty} Q_k\right).$$

Por otra parte, la ecuación (3.63) puede transformarse en una ecuación de Lyapunov de la forma

$$\widetilde{A}^T Y + Y\widetilde{A} + P = 0\,, \tag{3.65}$$

donde $\widetilde{A} = A^T$. Reescribiendo la iteración para esta nueva ecuación, obtendremos

$$\widetilde{A}_{k+1} = \frac{1}{2}\left(\widetilde{A}_k + \widetilde{A}_k^{-1}\right),$$

$$P_{k+1} = \frac{1}{2}\left(P_k + \widetilde{A}_k^{-T} P_k \widetilde{A}_k^{-1}\right), \tag{3.66}$$

donde $\widetilde{A}_0 = \widetilde{A}$, $P_0 = P$ y la solución de la ecuación de Lyapunov viene dada por

$$Y = \tfrac{1}{2} P_\infty = \tfrac{1}{2}\left(lim_{k\to\infty} P_k\right).$$

Es fácil ver que para la iteración de (3.66) tenemos que $\widetilde{A}_j = A_j^T$, si A_j denota la iteración de (3.64). Por lo tanto X e Y pueden calcularse simultáneamente mediante la siguiente iteración

$$\begin{aligned} A_{k+1} &= \frac{1}{2}\left(A_k + A_k^{-1}\right), \\ Q_{k+1} &= \frac{1}{2}\left(Q_k + A_k^{-T} Q_k A_k^{-1}\right), \\ P_{k+1} &= \frac{1}{2}\left(P_k + A_k^{-1} P_k A_k^{-T}\right). \end{aligned} \tag{3.67}$$

A continuación presentamos el algoritmo para resolver la ecuación de Lyapunov en tiempo continuo acopladas.

Algoritmo 9 *[SILAE]*

1. $V = A$, $X = Q$, $Y = P$
2. mientras $\|V + I_n\|_1 > tol$
   - 2.1. $A = LUP$, mediante la descomposición LU con pivotamiento parcial.
   - 2.2. $\gamma = \left|\det(A)\right|^{1/n} = \prod_{k=1}^{n} \left|u_{kk}\right|^{1/n}$
   - 2.3. $W = P^T U^{-1} L^{-1}$, mediante eliminación progresiva y sustitución regresiva
   - 2.4. $A = \frac{1}{2}\left(\frac{1}{\gamma} V + \gamma W\right)$
   - 2.5. $X = \frac{1}{2}\left(\frac{1}{\gamma} X + \gamma W^T X W\right)$
   - 2.6. $Y = \frac{1}{2}\left(\frac{1}{\gamma} Y + \gamma W Y W^T\right)$
   - 2.7. $V = A$

   fin mientras
3. $X = -\frac{1}{2} P X P^T$, $Y = -\frac{1}{2} P Y P^T$

fin

Este algoritmo es fácilmente extensible para las ecuaciones de Lyapunov generalizadas acopladas. A este algoritmo lo denominaremos *SILGAE*. Además si aplicamos las ideas propuestas en el algoritmo *SILCE* a la resolución de las ecuaciones de Lyapunov, estándar y generalizadas, acopladas obtenemos dos algoritmos que calculan el factor de Cholesky de la solución de estas ecuaciones y que denominaremos *SILCAE* y *SILCGAE*, respectivamente.

3.5.9 Algoritmo para resolver la ecuación de Lyapunov en tiempo discreto

En este apartado estudiaremos cómo se puede aplicar el método de la función signo matricial para resolver las ecuaciones de Lyapunov en tiempo discreto. Para ello sólo necesitaremos aplicar ciertas manipulaciones algebraicas.

Consideremos la ecuación de Lyapunov en tiempo discreto semidefinida positiva

$$A^T XA - A + Q = 0\,,$$

donde $A \in \mathsf{R}^{n\times n}, Q = Q^T \in \mathsf{R}^{n\times n}$, $X = X^T$ y $\Lambda(A) \subset \{\lambda_i \in \mathsf{R} : |\lambda_i| < 1\}$. Para esta ecuación se define la matriz simpléctica Z asociada y definida por [Petkov 91]

$$Z = \begin{bmatrix} A & 0 \\ -A^{-T}Q & A^{-T} \end{bmatrix},$$

cuyos valores propios serán $\lambda_i, i = 1,2,\ldots,n$ y $\dfrac{1}{\lambda_i}, i = 1,2,\ldots,n$.

Usando la tranformación bilineal o de Cayley obtenemos

$$\mathcal{H} = (Z+1)(Z-1)^{-1} \tag{3.68}$$

donde $\mathcal{H}$ es la matriz Hamiltoniana

$$\mathcal{H} = \begin{bmatrix} \widetilde{A} & 0 \\ -\widetilde{Q} & -\widetilde{A}^T \end{bmatrix}$$

cuyos valores propios serán $\dfrac{\lambda_i - 1}{\lambda_i + 1}, i = 1,2,\ldots,n$ y $-\dfrac{\lambda_i + 1}{\lambda_i - 1}, i = 1,2,\ldots,n$. Esta matriz Hamiltoniana está asociada a una ecuación de Lyapunov en tiempo continuo definida por

$$\widetilde{A}^T X + X\widetilde{A} + \widetilde{Q} = 0\,. \tag{3.69}$$

La resolución de esta ecuación puede abordarse mediante el algoritmo 6 descrito en el apartado 3.5.5 para el caso estándar a partir de las siguientes iteraciones

$$\widetilde{A}_{k+1} = \frac{1}{2}\left(\widetilde{A}_k + \widetilde{A}_k^{-1}\right),$$

$$\widetilde{Q}_{k+1} = \frac{1}{2}\left(\widetilde{Q}_k + \widetilde{A}_k^{-T}\widetilde{Q}_k\widetilde{A}_k^{-1}\right). \tag{3.70}$$

donde $\widetilde{A}_0 = \widetilde{A}$ y $\widetilde{Q}_0 = \widetilde{Q}$.

Podemos evitar la inversión del factor $(Z - I)$ en la expresión (3.68) si consideramos

$$\mathcal{H} = KL^{-1},$$

y tratamos este problema como una ecuación generalizada de Lyapunov en tiempo continuo.

3.6 Conclusiones

En este capítulo se ha estudiado la resolución de las ecuaciones de Lyapunov desde tres métodos distintos. Los dos primeros, Bartels-Stewart y Hammarling, son métodos con una primera parte iterativa y una segunda parte directa, mientras que el segundo, el basado en la función signo matricial, es un método esencialmente iterativo.

Los métodos de Bartels-Stewart y Hammarling son métodos numéricamente estables. Éstos requieren de una primera etapa iterativa muy costosa en la que la matriz de coeficientes se factoriza en la forma real de Schur mediante el algoritmo iterativo QR. A continuación se obtiene la solución mediante una etapa directa compuesta por la descomposiciones LU y resolución de sistemas triangulares.

El método de Hammarling es especialmente interesante pues proporciona directamente el factor de Cholesky de la solución de las ecuaciones de Lyapunov. En numerosas aplicaciones de control es este factor de Cholesky el resultado que se necesita, por lo que resolver la ecuación de Lyapunov para la solución explícita X y obtener a partir de esta el factor de Cholesky puede producir una pérdida de precisión en los resultados. Además, el algoritmo de Hammarling trabaja directamente sobre el factor C o B (matrices de entradas o de control) por lo que no es necesario formar explícitamente el producto $C^T C$ o $B^T B$, evitando de este modo una nueva posible pérdida en la precisión de los resultados.

El método de la función signo matricial está basado en la iteración de Newton. Este método no es numéricamente estable, pero en la práctica la precisión obtenida es similar a la de aquellos. Una de las principales características de este método es que requiere operaciones como productos de matrices, resolución de sistemas triangulares e inversión de matrices. Estas operaciones están muy optimizadas en gran número de librerías computacionales, lo que permite un alto rendimiento a la hora de la implementación del código.

Por otra parte, el método de la función signo permite una fácil generalización del método para la resolución de ecuaciones de Lyapunov generalizadas o su modificación para la obtención del factor de Cholesky de la solución.

Capítulo 4

Arquitecturas y algoritmos paralelos

4.1 Introducción

El desarrollo de nuevos y más potentes computadores ha permitido en los últimos años resolver problemas que, por su alto coste computacional y/o por posibles restricciones en el tiempo de respuesta, eran prácticamente inabordables. La evolución ha sido tan notable, que en pocos años se ha pasado de "supercomputadores" con unas prestaciones de unos pocos cientos de flops (*floating-point operations per second* u operaciones en coma flotante por segundo) a las máquinas masivamente paralelas que persiguen como objetivo el Teraflop (10^{12} flops).

Un estudio de las arquitecturas de altas prestaciones debe necesariamente comenzar por establecer una clasificación que las agrupe y permita identificar sus características más notables. En la sección 4.2 se describen diversas clasificaciones, atendiendo a diferentes tipos de parámetros. En una clasificación en función de su utilización en la computación científica y técnica, sobresalen por sus prestaciones los multiprocesadores, máquinas con varios procesadores completos capaces de funcionar de manera asíncrona.

Los multiprocesadores con memoria compartida disponen de herramientas que permiten una programación "secuencial" clásica que obtiene prestaciones óptimas. Los programas escritos bajo un modelo de paso de mensajes obtienen las máximas prestaciones en el caso de multicomputadores. A cambio, este modelo presenta una programación mucho más compleja. En la sección 4.3 de este capítulo se presentan los principales parámetros que intervienen en la programación de multicomputadores, a saber, distribución de los datos, subrutinas de paso de mensajes y librerías recientemente desarrolladas para facilitar la generación de código.

Además, en la sección 4.4 se presentan las arquitecturas paralelas utilizadas en la evaluación de los algoritmos secuenciales y paralelos desarrollados en esta memoria. Esta descripción permitirá estudiar las peculiaridades de cada máquina. En concreto las máquinas utilizadas son las siguientes: Alliant FX/80, Silicon Graphics (SGI) PowerChallenge, Cray T3D e IBM SP2.

4.2 Arquitecturas de Altas prestaciones

4.2.1 Clasificación de las arquitecturas

La evolución de los sistemas de altas prestaciones ha supuesto el desarrollo de un gran número de arquitecturas diferentes. Esta variedad ha llevado a diversas apuestas de futuro por parte de los fabricantes.

En el campo de los sistemas informáticos de aplicación general la estructura básica está fundamentada en uno o más procesadores que siguen el paradigma Von Neumann o secuencial con considerables modificaciones y prestaciones adicionales. Algunas de las contribuciones más importantes realizadas en la arquitectura de los computadores monoprocesador han sido:

- La microprogramación, introducida por M. Wilkes en 1951.
- El segmentado en la ejecución de instrucciones (lo que ha dado lugar a los procesadores supersegmentados) introducido en 1964 por S. Cray en el computador CDC 6600.
- Las antememorias (M. Wilkes, 1965).
- El uso de unidades vectoriales (S. Gray, 1975).
- La multiplicidad de unidades aritméticas para números enteros y reales en coma flotante en los procesadores (IBM, 1990).

Desde el punto de vista de su utilización en la computación científica se pueden distinguir actualmente dos grupos: procesadores superescalares y procesadores vectoriales.

Podemos identificar los **procesadores vectoriales** por la existencia en su código máquina de instrucciones para el manejo de vectores. El trabajo con los vectores se realiza en unidades aritméticas segmentadas (*pipeline*), capaces de dividir las instrucciones en varias etapas y ejecutar etapas diferentes simultáneamente. Este tipo de procesadores son especialmente apropiados para la ejecución de códigos secuenciales, compuestos de un gran número de operaciones aritméticas entre datos. Ejemplos de ellos son los procesadores del Cray Y-MP, Cray X-MP, Alliant FX/80, etc.

Los **procesadores superescalares**, que han experimentado un considerable auge en los últimos años, resultan especialmente apropiados para ejecutar eficientemente cualquier tipo de código. En la práctica los programas están compuestos por diferentes tipos de operaciones (aritméticas, de entrada/salida, condicionales, de salto, etc.). Los procesadores superescalares disponen de varias unidades funcionales independientes, capaces de realizar en paralelo diferentes operaciones. El desarrollo de estos computadores está ligado a la evolución del diseño RISC (*Reduced Instruction Set Code* o código con un conjunto de instrucciones reducido) en los que el microlenguaje del computador está compuesto por un mínimo número de instrucciones muy simples y la tecnología CMOS. Ejemplos de estos procesadores son el RS/6000 de IBM, el DEC Alpha AXP, el MIPS R8000 y R10000 de Silicon Graphichs, etc.

En el caso de los sistemas con más de un procesador, o multiprocesadores, la mayoría de las arquitecturas actuales pueden ser incluidas dentro de dos categorías: SIMD y MIMD [Flynn 66, Hwang 84].

En las **SIMD** (*Single Instruction Multiple Data* o Simple Instrucción Múltiples Datos) todos los procesadores ejecutan las mismas instrucciones simultáneamente, pero sobre diferentes partes del

conjunto de datos a tratar. Para incrementar la flexibilidad de estas máquinas existe un sistema de máscaras sobre la actividad de los procesadores. Los procesadores son frecuentemente simples, es sencilla la construcción de las conexiones entre los procesadores y la memoria, y estas arquitecturas suelen contener una gran cantidad (miles) de nodos de procesamiento (por ejemplo: CM-2, MasPar, etc.). Esta arquitectura era hasta no hace mucho tiempo la preferida para la construcción de máquinas paralelas.

Los sistemas **MIMD** (*Multiple Instructions Multiple Data* o Multiples Instrucciones Múltiples Datos) son un conjunto de procesadores/computadores independientes, ejecutando cada uno de ellos su propio flujo de instrucciones y conectados a través de una red de comunicación (bus, red de conmutación dinámica (*crossbar, hipercubo, omega*, etc.), red estática de topología determinada, LAN, etc.). Los procesadores (o unidades de proceso) pueden ser secuenciales, vertoriales o superescalares. Los sistemas MIMD pueden clasificarse, en base al tipo de conexión y organización de la memoria del sistema, en memoria compartida y memoria distribuida.

En ambos casos la jerarquía de memoria de estos sistemas tiene dos o más niveles, cada uno de ellos con un tiempo de acceso distinto. Es lo que se denomina arquitectura NUMA (*Non Uniform Memory Addres*).

En los sistemas multiprocesador con memoria compartida (MMC) todos los procesadores (del mismo tipo) están conectados a una única memoria central, direccionando un vector común de memoria física, a través de una red de interconexión (ver la Figura 1). En la mayoría de ellos actualmente los procesadores están conectados a la memoria compartida a través de una antememoria (éstos se denominan multiprocesadores simétricos) para paliar en lo posible el "cuello de botella" que supone el acceso a la memoria central y que limita el número máximo de procesadores (alrededor de 16). Actualmente los procesadores poseen varios niveles de antememoria (L1: primer nivel, L2: segundo nivel, etc.). Generalmente está aceptado que este tipo de sistemas es más fácil de programar que los de memoria distribuida.

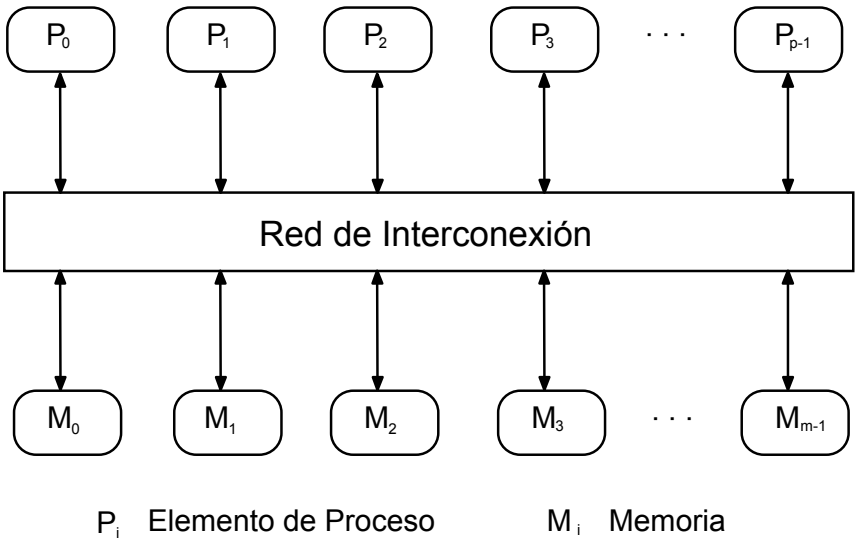


Figura 1. *Esquema de un multiprocesador con memoria compartida*.

Los sistemas multiprocesador con memoria distribuida (MMD), también denominados multicomputadores, aparecieron en 1982. En estos sistemas cada procesador (en este caso es más apropiado hablar de computador) tiene su propia memoria local, no accesible directamente por otros procesadores, y se comunica con el resto de procesadores mediante de paso de mensajes a través de algún tipo de red de comunicación (ver la Figura 2). Las topologías utilizadas para estas redes son muy variadas y van desde el bus o el anillo (propio del "cluster" de estaciones de trabajo), pasando

por la malla hasta el hipercubo, dentro de lo que se denominan redes estáticas. Además existen diversas implementaciones de redes dinámicas en las que un circuito (*router*) se encarga de encaminar los mensajes desde el procesador origen hasta el destino

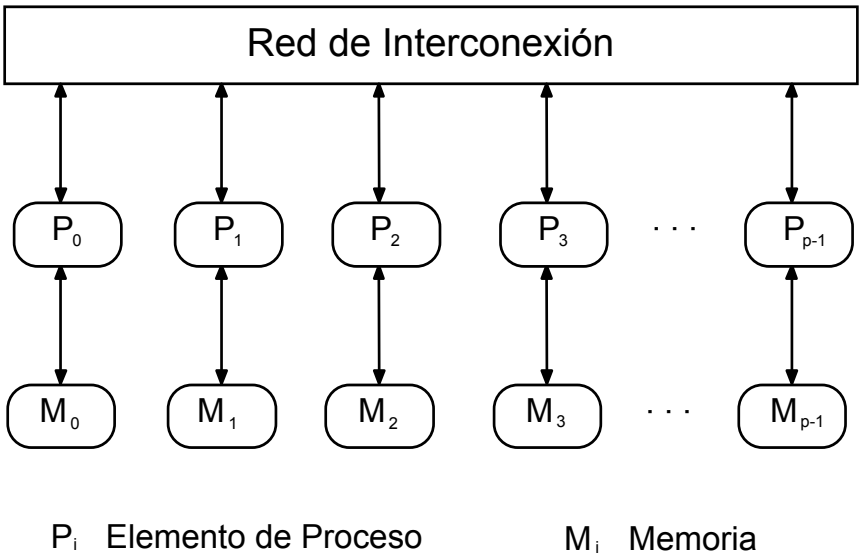


Figura 2. *Esquema de un multiprocesador con memoria distribuida.*

4.2.2 Metodologías de programación

Sobre estas arquitecturas se pueden establecer, a un nivel superior, diversos modelos de programación. Los modelos básicos que podemos encontrar, de más actualidad, son el de paralelismo de datos, el de paso de mensajes y el de memoria compartida. El paralelismo de datos tiene su origen en los computadores SIMD y fue desarrollado para permitir la migración de las aplicaciones SIMD a sistemas MIMD (un ejemplo lo podemos ver en la programación de la CM-2 de Thinking Machines). El paso de mensajes tiene su origen en el desarrollo de los primeros sistemas MIMD de memoria distribuida, los cuales intercambian datos mediante la transmisión de mensajes. El modelo de memoria compartida asume que todos los procesadores tienen acceso a una memoria global y tiene su origen en los computadores monoprocesador y en los multiprocesadores con memoria compartida (Alliant FX/80, Power Challenge de SGI) y una de sus particularizaciones es el denominado paralelismo de control. Desde el año 1991 este modelo de programación se está implantando en computadores MIMD con memoria distribuida en lo que se ha venido a denominar memoria virtualmente compartida (*Virtual shared memory* o *distributed shared memory*). Compañías como Kendall Square Research, Cray Research, Convex-HP y Meiko ofrecen ya este modelo de programación. La aplicabilidad de esta implementación quedará limitada por la eficiencia y las escalabilidad que pueda conseguirse en el acceso a datos remotos; por ello el énfasis tecnológico se pone en el diseño de redes de comunicación que soporten anchos de banda elevados de forma sostenida al aumentar el número de nodos.

La tecnología del software en la computación de altas prestaciones ha evolucionado durante los últimos veinte años lentamente hacia la estandarización. Primero en la opción de los lenguajes tales como el tradicional FORTRAN (77 y nuevas versiones como el 90 o el HPF), que permanece dominante en la computación científico-técnica, el C (++) y el SQL, estos últimos sobresaliendo en el campo comercial, y más recientemente en el caso de los sistemas operativos hacia el UNIX (con ligeras variantes dentro de cada fabricante).

En los computadores con el modelo de programación de memoria compartida cada fabricante posee un conjunto de directivas paralelizantes y vectorizantes propias para sus compiladores de lenguaje [Alliant-88], lo cual obliga a cambiarlas (o añadir las nuevas) cuando se desea trasladar el código a otro computador. En el caso del modelo de programación por paso de mensajes se está pasando de la anarquía reinante en cuanto a metodologías, soporte para la programación paralela y métodos de interacción, a cierta estandarización gracias a la aceptación del PVM (*Parallel Virtual Machine*) [Sunderam-94], de amplia implantación actualmente, y más recientemente del MPI [MPI-94] (*Message Passing Interface*). El diseño de implementaciones de estos estándares de comunicación se está extendiendo por todas las plataformas *hardware*: multiprocesadores con memoria compartida (Power Challenge de SGI), multicomputadores (SPx de IBM, CM-5 de TMC, T3D de Cray, etc.) y "clusters" de estaciones de trabajo. La potencialidad de este último grupo de sistemas se está incrementando día a día fruto de los nuevos estándares de comunicación que se están implantando como el ATM o la Fast Ethernet.

4.2.3 Prestaciones de los algoritmos paralelos

Las prestaciones de los algoritmos paralelos para multiprocesadores pueden evaluarse mediante diferentes parámetros (alguno de estos parámetros, como el tiempo de ejecución y los Megaflops, 10^6 flops, son asimismo aplicables en algoritmos no paralelos):

Tiempo de ejecución. Indica el tiempo necesario para resolver el problema mediante un determinado algoritmo. Podemos distinguir entre el tiempo secuencial, T_S, que es el tiempo necesario para resolver el problema en 1 procesador y el tiempo paralelo, T_p, que es el tiempo necesario en p procesadores.

Megaflops (o Megaflop/segundo). Evalúa la potencia de cálculo del sistema. Se obtiene como el cociente entre el coste teórico aritmético del algoritmo (medido en flops, es decir, operaciones en coma flotante) y el coste temporal del algoritmo (en segundos).

Aceleración (incremento de velocidad o *speedup*). Indica el incremento de velocidad obtenido al utilizar p procesadores. Se obtiene como el cociente $S_p = T_1 / T_p$. Este valor depende del tamaño del problema, por lo que es más correcto definir $S_p^n = S_p(n) = T_1(n) / T_p(n)$. Normalmente la aceleración de los algoritmos suele ser $S_p^n \le p$, salvo en los casos en los que al aumentar el tamaño del problema la arquitectura del computador favorece la distribución de los datos y/o el trabajo entre diversos procesadores (computadores en los que cada procesador tiene su propia antememoria). En estos casos $S_p^n > p$, obteniéndose lo que se conoce como *superspeedup*.

Eficiencia. Mide el grado de aprovechamiento de los procesadores y se calcula como el cociente $E_p^n = S_p^n / p$.

Escalabilidad. Degradación de la eficiencia del algoritmo al aumentar el número de procesadores y el tamaño del problema, manteniéndose constante la relación entre éstos. Se dice que un algoritmo es *escalable* para un problema de tamaño (local) k cuando $E_p^n = \text{constante}$, $n = kp$, $p = 1,2,\ldots$. La escalabilidad de los algoritmos, si se da, suele alcanzarse para valores de k elevados.

4.3 Multicomputadores

Los lenguajes imperativos (Fortran, C, Modula-2, etc.), que durante años han sido válidos para programar procesadores secuenciales, son asimismo útiles para desarrollar código para los procesadores vectoriales y superescalares y para los MMC. El procesamiento y traducción de un programa previo a su ejecución es una tarea que, en estos tipos de computadores, es elaborada por un compilador especializado. Existen compiladores capaces de generar código que puede aprovechar eficientemente los recursos de estos computadores.

A nivel de usuario, atendiendo cómo se gestionan los accesos a memoria en multicomputadores, como ya hemos comentado en la sección anterior, podemos distinguir entre entornos de programación donde para el usuario es indiferente el acceso a datos locales o remotos (memoria virtual compartida) y, por otro lado, entornos donde es el propio usuario el encargado de gestionar los accesos a los datos remotos (entornos de paso de mensajes) [Kumar 94]. Hay que hacer notar que un mismo computador a menudo puede ser programado de ambos modos (por ejemplo, el KSR-1, SGI PowerChallenge, Cray T3D, etc).

La ventaja principal del entorno de paso de mensajes es su eficiencia. Otras ventajas son su universalidad, expresividad y facilidad de depuración [Gropp 94]. El modelo de programación CSP (*Communicating sequential processes* o procesos comunicantes secuenciales) describe fielmente la filosofía de este entorno de programación [Hoare 78]. Además, este modelo presenta una ventaja adicional de gran interés. Prácticamente cualquier lenguaje imperativo puede adecuarse al modelo mediante el desarrollo o uso de unas pocas primitivas de comunicación. Así han surgido infinidad de pseudolenguajes, capaces de proporcionar el modelo de programación CSP, y adecuados para un tipo particular de arquitectura.

4.3.1 Distribución de datos

Desde el punto de vista del usuario, cuando queremos dar solución a un problema en un multicomputador (bajo el modelo CSP), la primera decisión que debe tomarse es la distribución de los datos. Esta decisión define muchas de las características de la solución, como el grado de paralelismo, la necesidad de comunicaciones, etc. y, en definitiva, la eficiencia del algoritmo. Así, la distribución de los datos debe intentar:

- Distribuir equitativamente la carga computacional entre los procesadores. Puesto que el tiempo total de un algoritmo paralelo en un multicomputador es igual al tiempo total en el procesador que más tarda en completar su ejecución, es necesario que todos los procesadores realicen aproximadamente el mismo trabajo.

- Minimizar las comunicaciones requeridas. Las comunicaciones entre procesadores establecen puntos de sincronización entre procesos y, en la mayor parte de los casos, demoras y esperas que disminuyen el grado de aprovechamiento de los recursos.

- Reducir las necesidades de memoria. La computación paralela sobre multiprocesadores es al mismo tiempo computación distribuida. Por ello, es posible resolver problemas de mayor tamaño en este tipo de arquitecturas si los datos se encuentran distribuidos.

A menudo estos parámetros resultan contrapuestos, de modo que la mejora de uno de ellos implica que otro o más empeoren.

Cuando nos enfrentamos a problemas de computación matricial son precisamente las matrices del algoritmo las que deben distribuirse entre los procesadores. Existen diversos modelos

extensamente conocidos para la distribución de las matrices, como por ejemplo las distribuciones en 1 dimensión (1-D) por bloques, cíclica por filas/columnas, antidiagonales, serpenteantes, etc. y las distribuciones en 2 dimensiones (2-D) malla, toroidal, etc. En esta tesis se han utilizado 2 distribuciones 1-D diferentes, por bloques y cíclica por bloques de columnas (estas distribuciones son fácilmente trasladables al caso de filas) y la distribución en 2-D toroidal.

En concreto se han utilizado las distribuciones en 1-D mencionadas para resolver las ecuaciones de Lyapunov utilizando el método de Hammarling. La principal objeción que se hace a las distribuciones en 1-D es su escasa escalabilidad, pues resulta difícil mantener las prestaciones en este tipo de distribuciones a medida que aumenta el número de procesadores. Sin embargo, debemos tener en cuenta que el problema bajo estudio es un problema de control en el modelo de espacio de estados representado por matrices densas. El método de Hammarling está orientado a columnas (o filas, dependiendo de cómo se aborde su implementación) y las matrices que aparecen en este modelo son de dimensión moderada (no más allá de 2000×2000). Además, por regla general, las distribuciones en 1-D resultan más eficientes para problemas de dimensión moderada que las distribuciones en 2-D. En problemas de mayor dimensión las matrices que intervienen ya no son densas sino dispersas y, por tanto, los métodos estudiados resultan inviables debido a su coste.

No obstante, se han desarrollado algoritmos paralelos basados en una distribución 2-D toroidal para resolver la ecuación de Lyapunov mediante la función signo matricial. Esto es debido a que se trata de un método iterativo en el que abundan los productos de matrices y las descomposiciones ortogonales, operaciones con un alto rendimiento con este tipo de distribución.

Consideremos las funciones div(·) y res(·) que devuelven el cociente y el resto, respectivamente, de la división entera, y supongamos que disponemos de p procesadores, $P_0, P_1, \ldots, P_{p-1}$.

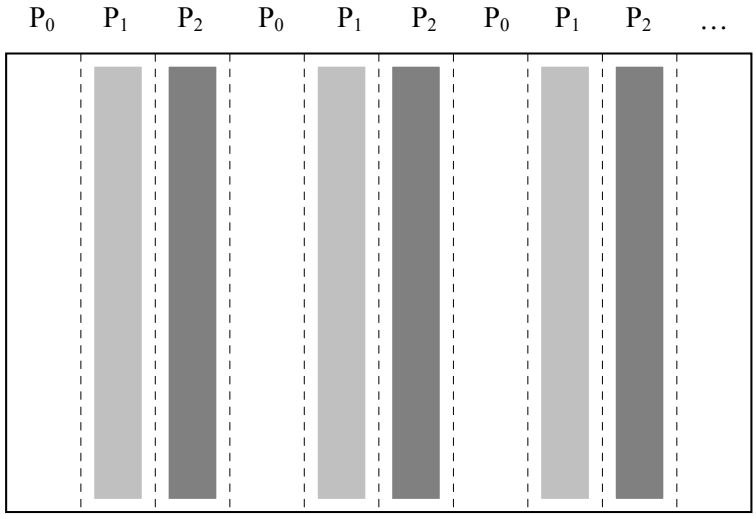


Figura 3. *Distribución cíclica por bloques de columnas para 3 procesadores*.

En la distribución cíclica por bloques de columnas se divide la matriz $M \in \mathsf{R}^{m \times n}$ en bloques de nb columnas consecutivas, es decir $M = [M_0, M_1, \ldots, M_{k-1}]$ donde $k = (n-1) \operatorname{div} nb + 1$, $M_i \in \mathsf{R}^{m \times nb}$, $0 \le i < k-1$, y $M_{k-1} \in \mathsf{R}^{m \times (n \operatorname{res} nb)}$. El bloque M_i se almacena entonces en el procesador $P_{i \operatorname{res} p}$. Esta distribución incluye como casos particulares la distribución cíclica por columnas ($nb = 1$) y la distribución por bloques de columnas ($nb = n \operatorname{div} p$). La distribución puede observarse, gráficamente, en la Figura 3.

No es necesario que los bloques de columnas almacenados localmente en un procesador queden exactamente en las mismas columnas de la matriz original. Si se quiere reducir la cantidad de memoria necesaria, es posible compactar el almacenamiento sobre la matriz local, de modo que el bloque M_g, que consta de las columnas (globales) $g\,nb+1,\ldots,(g+1)\,nb$, es a su vez el bloque de la matriz local M_l, $l = g \operatorname{div}(nb\ p)+1$, donde ocupa las columnas (locales) $l\,nb+1,\ldots,(l+1)\,nb$. De esta forma una matriz de dimensión *m* x *n* necesita (*m* x *n*)/*p* posiciones en cada procesador para ser almacenada de manera distribuida. En el caso de matrices triangulares se aproximan las dimensiones globales y locales por (*m* x *n*)/2 y (*m* x *n*)/(2*p*), respectivamente.

En la descripción de los algoritmos paralelos resulta más fácil no obstante trabajar con coordenadas globales (índices *g*) por lo que en algunos casos obviaremos la utilización de coordenadas locales (índices *l*). Asimismo, a partir de este momento y por simplicidad, consideraremos que los tamaños de las matrices son múltiplos exactos del número de procesadores y del tamaño de bloque, *nb* y *p*. Estas mismas consideraciones se repiten para simplificar la presentación de los algoritmos paralelos para cualquier tipo de distribución.

En la distribución por bloques de columnas se divide la matriz $M \in \mathsf{R}^{m\times n}$ en *p* bloques de *nb* columnas consecutivas, es decir, $M = [M_0, M_1, \ldots, M_{p-1}]$ donde $nb = n \operatorname{div} p$, $M_i \in \mathsf{R}^{m\times nb}$, $0 \le i < p-1$, salvo el último bloque $M_{p-1} \in \mathsf{R}^{m\times(nb+n \operatorname{res} p)}$. El bloque M_i se almacena entonces en el procesador P_i. Esta distribución puede observarse, gráficamente, en la Figura 4.

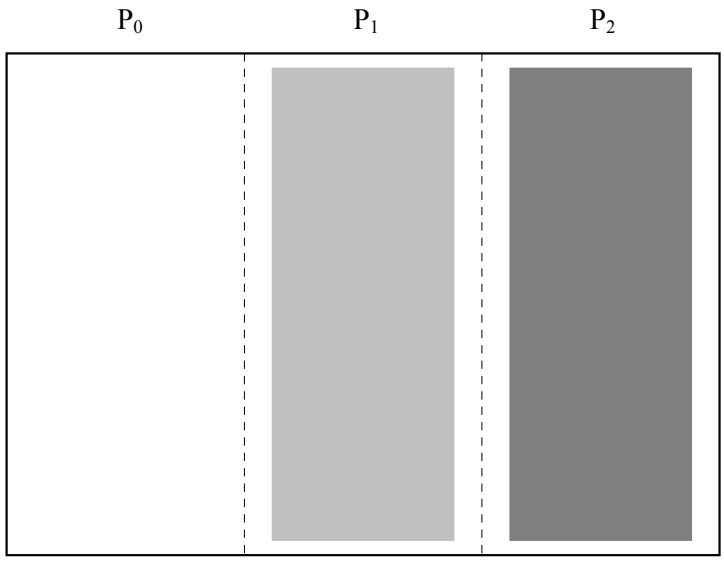


Figura 4. *Distribución por bloques de columnas para 3 procesadores*.

Al igual que en el caso cíclico, no es necesario que los bloques de columnas almacenados localmente en un procesador queden exactamente en las mismas columnas de la matriz original. Si se quiere reducir la cantidad de memoria necesaria, es posible compactar el almacenamiento sobre la matriz local, de modo que el bloque M_g, que consta de las columnas (globales) $g\,nb+1,\ldots,(g+1)\,nb$, es a su vez el bloque de la matriz local M_1, donde ocupa las columnas (locales) $1,\ldots,nb$. De esta forma una matriz de dimensión *m* x *n* necesita (*m* x *n*)/*p* posiciones en cada procesador para ser almacenada de manera distribuida. En el caso de matrices triangulares se aproximan las dimensiones globales y locales por (*m* x *n*)/2 y (*m* x *n*)/(2*p*), respectivamente.

Finalmente, consideremos que disponemos de un conjunto de $p \times q$ procesadores

$$\begin{matrix} P_{00} & P_{01} & \cdots & P_{0,q-1} \\ P_{10} & P_{11} & \cdots & P_{1,q-1} \\ \vdots & \vdots & \ddots & \vdots \\ P_{p-1,0} & P_{p-1,2} & \cdots & P_{p-1,q-1} \end{matrix}.$$

En la distribución toroidal se divide la matriz $M \in \mathsf{R}^{m \times n}$ en bloques M_{ij} de tamaño $mb \times nb$, es decir, $[M_{ij}]_{mb \times nb}$, $0 \le i < (m-1) \operatorname{div} mb + 1$, $0 \le j < (n-1) \operatorname{div} nb + 1$. Entonces el bloque M_{ij} se asigna al procesador $P_{i \operatorname{res} p, j \operatorname{res} p}$. Esta distribución incluye como casos particulares todas las distribuciones por filas y columnas en 1-D, tanto cíclicas como por bloques. La correspondiente descripción gráfica aparece en la Figura 5.

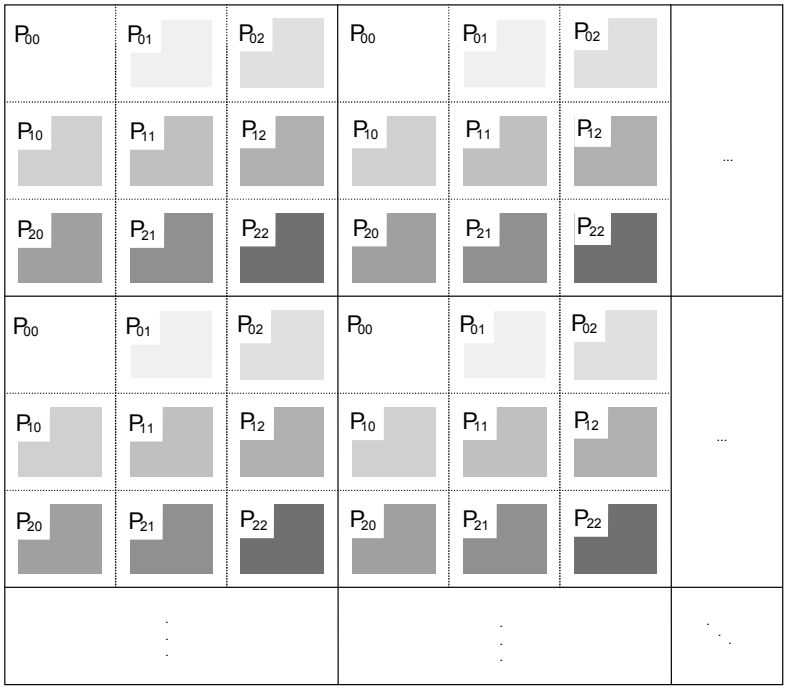


Figura 5. *Distribución toroidal para* 3 x 3 *procesadores*.

4.3.2 Comunicaciones

En los multicomputadores, la transferencia de información entre los procesadores se realiza mediante el paso de mensajes. Así pues, un algoritmo paralelo está compuesto de operaciones (fundamentalmente nos interesan las operaciones aritméticas) y comunicaciones que establecen puntos de sincronismo entre procesadores. Si no es posible solapar (simultanear) cálculos y comunicaciones, el coste total del algoritmo paralelo responderá a la suma del coste aritmético y el coste de comunicaciones.

Frente al coste aritmético, que se mide en flops, el cálculo del coste de comunicaciones es algo más complejo, aunque se puede normalizar para poderlo comparar con el aritmético.

a)

b)

c)

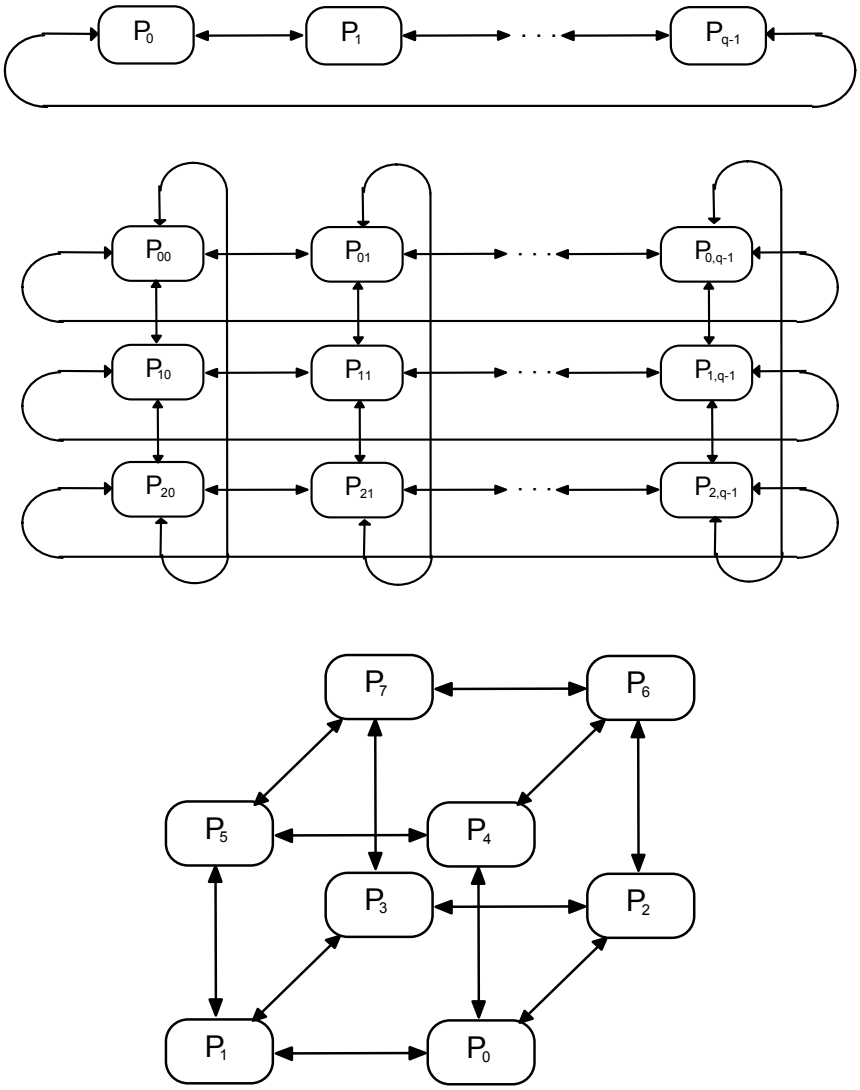


Figura 6. *Redes de interconexión con topología en (a) anillo de q procesadores, (b) toro de* $3 \times q$ *procesadores e (c) hipercubo de dimensión 3*.

Las comunicaciones dependen fuertemente del nivel físico de conectividad (red de interconexión) de la máquina. Las redes de interconexión más habituales son las estáticas y según la forma en que conecten a los procesadores establecerán una topología distinta. Podemos distinguir entre redes de conexión con topología en anillo, malla, toro, hipercubo, etc. (ver la Figura 6).

En principio, la velocidad de comunicación en estos sistemas distribuidos se mide en función de dos parámetros,

- la *latencia* o tiempo de establecimiento de la señal, α, y

- el tiempo necesario para transmitir una unidad de información a través de un enlace, τ, cuyo valor es inversamente proporcional al ancho de banda del enlace.

Estos parámetros permiten, por ejemplo, estimar el tiempo de transferencia de un mensaje de dimensión *n* a través de un enlace como: $\alpha + \tau n$.

De manera más general, podemos distinguir entre diferentes operaciones de comunicación como:

- Envío o recepción de un mensaje entre dos procesadores no necesariamente vecinos.

- Difusión de un mensaje. Un nodo envía un mensaje al resto de nodos del sistema.

- Reunión de mensajes. Un nodo recoge un mensaje de cada uno de los restantes nodos del sistema y realiza una operación de reducción sobre éstos (por ejemplo, calcula el máximo, la suma, etc.).

- Reunión global de mensajes. Cada nodo recoge un mensaje de cada uno de los restantes nodos del sistema y realiza una operación de reducción sobre éstos.

En la última generación de multicomputadores, la realización de un número cada vez mayor de estas operaciones queda en manos de complejos algoritmos de encaminamiento. Es decir, la máquina ofrece al usuario un conjunto de subrutinas de comunicación entre nodos (al menos para el envío, difusión y recepción de mensajes). El coste de estas subrutinas de comunicaciones dependerá del número de enlaces que se atraviesen, el número de paquetes en que se divida el mensaje, etc. [Dally 87, Dally 92]. Por simplicidad, para evaluar el coste de los algoritmos paralelos contaremos el número de mensajes que cada procesador envía (o difunde) y el tamaño de éstos.

En la descripción de los algoritmos paralelos utilizaremos las siguientes pseudo-subrutinas de comunicación:

- enviar(P_i, *msj*): El procesador que ejecuta la subrutina envía el mensaje *msj* al procesador P_i.

- recibir(P_i, *msj*): El procesador que ejecuta la subrutina recibe el mensaje *msj* del procesador P_i.

- difundir(*msj*): El procesador que ejecuta la subrutina difunde el mensaje *msj* a todos los procesadores. Por simplicidad, consideramos que también se envía el mensaje a si mismo.

Mediante estas subrutinas es fácil construir otros pseudo-algoritmos de comunicación más complejos.

4.3.3 Librerías de comunicaciones PVM y MPI

En los últimos años, el desarrollo de estándares de comunicaciones está permitiendo unificar la programación de los multicomputadores. De este modo han surgido librerías de comunicaciones, tales como PVM y MPI [Geist 94, Gropp 94], principalmente orientadas a los lenguajes de mayor difusión en la computación científica, Fortran y C.

Cualquiera de estas dos librerías, junto con el lenguaje de programación Fortran y los núcleos computacionales BLAS y LAPACK [Anderson 92, Dongarra 87a, Dongarra 88], componen un conjunto de herramientas suficientemente potente para permitir la resolución de numerosos problemas del álgebra matricial y de infinidad de otros problemas que responden a las mismas características.

La librería PVM (*Parallel Virtual Machine* o Máquina Virtual Paralela) está compuesta por una serie de programas que permiten a un conjunto heterogéneo de computadores trabajar concurrentemente [Geist 94]. PVM está diseñada para unir recursos computacionales y proporcionar

al usuario una plataforma paralela para ejecutar sus aplicaciones, independientemente del tipo de computadores de que se disponga y de su ubicación. Las funcionalidades que ofrece PVM son principalmente:

- Control de procesos. Asignación de identificadores de procesos, creación y eliminación de procesos, manejo de señales, gestión de máquinas virtuales, etc.

- Información sobre número de procesos en el sistema, identificadores de procesos, códigos de error y estado, etc.

- Operaciones sobre grupos. Creación, eliminación y modificación de grupos de procesos.

- Gestión de buffers de mensajes.

- Comunicaciones. Envío y recepción síncrona y asíncrona de mensajes, difusión global y por grupos, etc.

MPI (*Message Passing Interface* o Interfaz de Paso de Mensajes) es una librería de subrutinas que definen un estándar de comunicaciones [Gropp 94]. El MPI ha sido definido por el MPIF (MPI *Forum*), con la participación de más de 40 organizaciones, creadores de software, eminentes científicos, etc. que desde 1992 se han reunido para dar a luz una librería de comunicaciones para paso de mensajes que fuera ampliamente usada. Este estándar de comunicaciones fue formalmente establecido en Abril de 1994 (MPI 1.0).

Frente a PVM, que fue originalmente diseñada para entornos heterogéneos, MPI está más orientada hacia la programación de máquinas masivamente paralelas, aunque permite entornos heterogéneos. Así, MPI proporciona la mayor parte de las funcionalidades enumeradas en el caso de PVM salvo facilidades para la gestión (creación) de procesos y la configuración de máquinas virtuales.

Algunas de las características más importantes de la librería MPI son las siguientes:

- Permite comunicaciones eficientes. Eliminando el copiado de memoria a memoria (como ocurre en PVM) y permitiendo el solapamiento entre comunicaciones y cálculo.

- MPI es simple y muy potente. Cualquier programa paralelo y distribuido puede implementarse con tan solo 6 funciones, pero MPI dispone de hasta 125 funciones y procedimientos.

- Comunicaciones punto a punto y colectivas. Posee rutinas de reducción.

- Define contextos de comunicación para las operaciones colectivas.

- Comunicaciones síncronas y asíncronas, bloqueantes o no, con o sin memoria tampón.

- Permite definir nuevos tipos de datos (tipos MPI). Esto permite el envío de complicadas estructuras de datos muy fácilmente (sin las copias necesarias en PVM).

- Permite definir topologías de procesos virtuales. Esto permite la optimización de las comunicaciones en computadores con una determinada topología.

Actualmente los fabricantes de computadores están desarrollando librerías de MPI optimizadas para sus arquitecturas. En el caso de sistemas multiprocesadores con memoria compartida las comunicaciones se realizan mediante lectura/escritura en la memoria principal del computador,

utilizándose procesos o hilos de ejecución para implementar las comunicaciones no bloqueantes. En el caso de un *cluster* de estaciones de trabajo lo más habitual es utilizar una red Ethernet con protocolo TCP/IP.

4.3.4 Núcleos computacionales paralelos (ScaLAPACK)

Recientemente se han desarrollado y se siguen desarrollando dos iniciativas para facilitar el desarrollo de algoritmos numéricos paralelos en multicomutadores: ScaLAPACK (Scalable Linear Algebra PACKage) y PLAPACK (Parallel Linear Algebra PACKage). Éstas están basadas en núcleos computacionales paralelos que permiten al usuario el desarrollo de aplicaciones paralelas con un considerable ahorro de tiempo y trabajando a un nivel de especificación superior que el utilizado hasta ahora para el desarrollo de este tipo de código.

ScaLAPACK es una librería de rutinas de álgebra lineal de altas prestaciones para sistemas multiprocesadores basados en el paso de mensajes y redes de estaciones de trabajo que utilizan PVM y/o MPI. Se trata de una continuación del proyecto LAPACK (diseñado para estaciones de trabajo, computadores vectoriales y multiprocesadores con memoria compartida) y al igual que éste posee rutinas para resolver sistemas de ecuaciones lineales, problemas de mínimos cuadrados y problemas de valores propios. Sus rutinas están diseñadas para trabajar únicamente sobre matrices densas.

La librería PLAPACK aporta el concepto de programación orientada a objetos al álgebra matricial paralela. Esta librería utiliza MPI y dispone de rutinas paralelas para operaciones elementales como producto matriz-vector y producto de matrices, resolución de sistemas triangulares lineales, etc. Otras operaciones matriciales más complejas no están disponibles aún por lo que su aplicación resulta limitada.

Los principales objetivos de estas dos librerías son eficiencia, escalabilidad, fiabilidad, portabilidad, flexibilidad y facilidad de uso (haciendo que las interfaces para LAPACK y ScaLAPACK sean lo más parecidas posible). Muchos de estos objetivos se han alcanzado gracias a que el desarrollo de estas librerías se ha basado en estándares, sobre todo para las rutinas aritméticas elementales (BLAS o *Basic Linear Algebra Subprograms*) y las de comunicaciones a bajo nivel (BLACS o *Basic Linear Albebra Communication Subprograms*).

La librería ScaLAPACK está escrita en FORTRAN 77 en un estilo de memoria compartida (SIMD), utilizando el paso de mensajes explícito para la comunicación entre los procesadores. Las matrices se asume que están distribuidas según una topología 2D toroidal por bloques (ver Figura 5).

La topología utilizada para las comunicaciones entre los procesadores, en especial para las operaciones de difusión y recogida de datos, es especificada por el programador mediante un parámetro específico. Se puede seleccionar una topología por defecto (en ocasiones optimizada para la máquina), varios tipos de topologías en anillo, árbol, hipercubo o una topología heterogénea diseñada por el propio usuario.

ScaLAPACK posee tres tipos de rutinas:

- **Directoras** (*driver*). Resuelven problemas algebraicos estándar (ej. Sistemas de ecuaciones).

- **Aritméticas** (*computational*). Realizan tareas aritméticas básicas. Cada tarea directora llama a una secuencia de rutinas aritméticas.

- **Auxiliares** (*auxiliary*). Realizan ciertas subtareas o cálculos aritméticos de menor entidad.

La Figura 7 describe la jerarquía del software de ScaLAPACK. En ésta, los componentes bajo la línea de puntos, etiquetados como **Local**, son llamados por un procesador, con argumentos almacenados sólo en ése procesador. Los componentes sobre la línea, etiquetados como **Globales**, son rutinas paralelas síncronas, cuyos argumentos incluyen matrices y vectores distribuidos entre los múltiples procesadores. A continuación se describen brevemente cada uno de estos componentes.

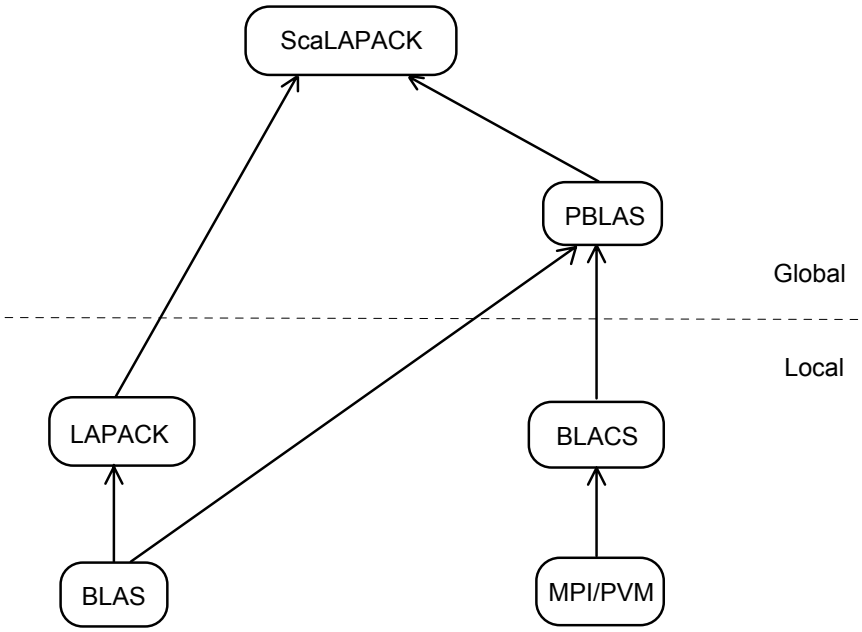


Figura 7. *Jerarquía software de ScaLAPACK*.

BLAS

El BLAS [Dongarra 87b, Dongarra 88] incluye subrutinas para operaciones de álgebra lineal tales como productos escalares, multiplicaciones matriz por vector y multiplicaciones de matrices, correspondientes a los niveles 1, 2 y 3 del BLAS, respectivamente. Utilizando operaciones de nivel BLAS-3 afinadas para una determinada arquitectura se pueden reducir los efectos de la jerarquía de memoria (fallos de página en las antememorias) y permitir que las operaciones en coma flotante se realicen casi a la máxima velocidad de la máquina.

LAPACK

El LAPACK [Anderson 92] (*Linear Algebra PACKage*) es una colección de rutinas para resolver sistemas lineales, problemas de mínimos cuadrados y problemas de valores propios y valores singulares. Las altas prestaciones que se obtienen con estas rutinas son debidas a que sus algoritmos están basados en llamadas al BLAS, sobre todo en multiplicaciones de matrices. Cada rutina tiene uno o más parámetros de afinado, tales como el tamaño de bloque para las rutinas del BLAS, para obtener las mejores prestaciones. Estos parámetros dependen de la máquina y se obtienen de una tabla, definida cuando el paquete es instalado, y referenciada en tiempo de ejecución.

Las rutinas del LAPACK han sido escritas como un solo hilo de ejecución y pueden ser ejecutadas en máquinas con memoria compartida donde esté disponible la librería BLAS paralela.

BLACS

El BLACS [Dongarra 91] es una librería de paso de mensajes diseñada para álgebra lineal. El modelo computacional consta de un conjunto de procesos unidimensional o bidimensional, donde cada proceso almacena parte de las matrices y los vectores. El BLACS posee rutinas de envío y recepción síncronas para transmitir una matriz o submatriz de un procesador a otro, difundir una matriz desde un procesador al resto, o realizar operaciones de reducción global.

PBLAS

El PBLAS es un BLAS paralelo, que permite el paso de mensajes y cuya interfaz en muy parecida al BLAS. Esta librería permite que así como el LAPACK hace llamadas al BLAS, el ScaLAPACK se haya construido a partir de llamadas al PBLAS. De esta forma la interfaz del ScaLAPACK es muy parecida, sino idéntica en ocasiones, a la del LAPACK.

La idea es que el PBLAS se convierta en un estándar para los sistemas en memoria distribuida como lo es el BLAS para los sistemas con memoria compartida.

4.4 Computadores paralelos

4.4.1 Alliant FX/80

Descripción de la arquitectura

Aunque ya desfasado, este computador ha marcado una tendencia en la construcción de computadores paralelos que ha influido mucho en la concepción de algunos computadores de altas prestaciones actuales, por lo que creemos que es interesante comentar su arquitectura [Alliant 87]. Se trata de un multiprocesador con memoria compartida, con autonomía de funcionamiento, pues soporta sus propios periféricos. Los procesadores son de dos tipos: IP, básicamente para control de periféricos y CE, para realizar operaciones en coma flotante, vectoriales y de gestión de la concurrencia. Las máquinas más modernas de la gama FX/80 incluyen procesadores ACE (CE avanzados) en lugar de los CE, siendo los primeros el doble de rápidos que los segundos. La velocidad de pico computacional de los ACE es de 23 Mflops.

El Alliant FX/80 posee ocho ACE's y dos niveles de memoria compartida. La memoria principal posee 8 bancos que pueden llegar a poseer cada uno 8 MBytes y está conectada mediante un bus a dos antememorias de 256 KBytes cada una. Las antememorias están conectadas a los procesadores a través de una red de interconexión de alta velocidad tipo *crossbar*. Acceder a la antememoria es de dos a tres veces más rápido que acceder a la memoria principal.

Además cada procesador tiene sus propias instrucciones de antememoria y su propio conjunto de registros vectoriales. En particular tienen 32 dobles registros de 8 bytes cada uno y la unidad vectorial de cada procesador está dividida en cuatro etapas. En la Figura 4.1 podemos ver en detalle la arquitectura de este computador.

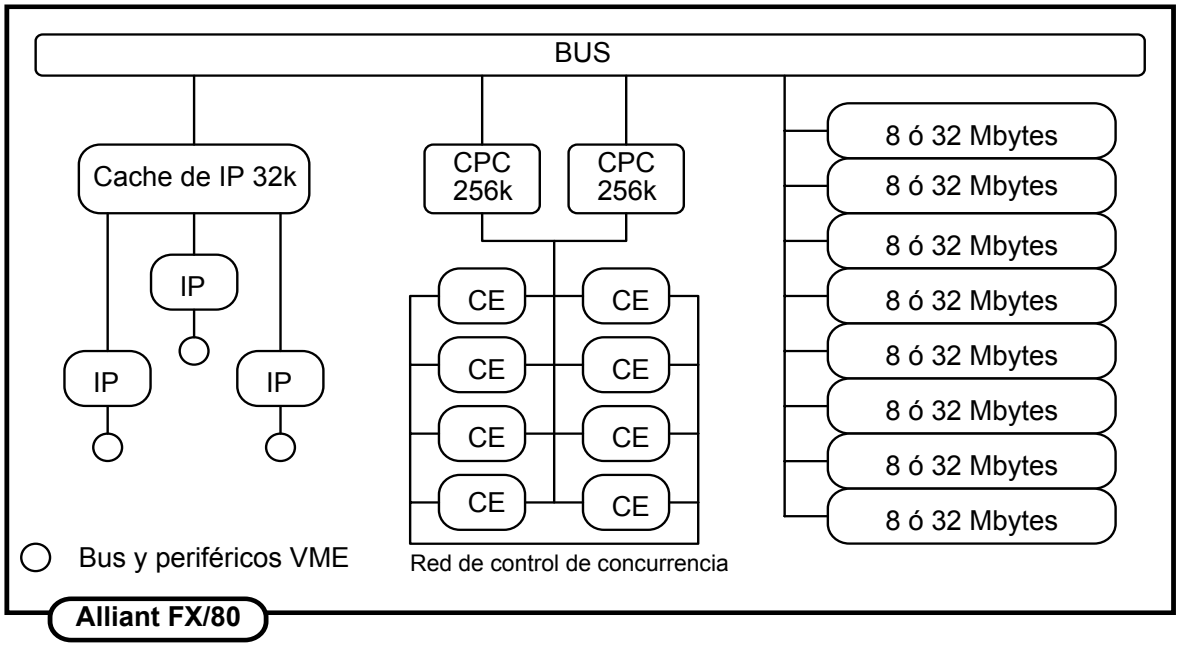


Figura 8. *Esquema de la arquitectura del Alliant FX/80.*

Entorno de programación

Básicamente existen dos metodologías para la programación en memoria compartida: Optimización de Bucles y Conjunto Explícito de Procesos. En esta última la esencia radica en la utilización de un lenguaje paralelo y técnicas de programación paralela, definiendo explícitamente los procesos que intervienen.

La optimización de bucles, utilizada en este computador, está ideada para la rápida paralelización de programas secuenciales. Podemos señalar que:

- La programación sigue siendo secuencial y es el compilador quien se encarga de paralelizar y/o vectorizar el programa, centrándose en los bucles.

- La aplicación se diseña con un enfoque secuencial, aunque sólo obtendremos buenos resultados con seguridad si ya se enfoca el código pensando en la paralelización que el compilador realizará. Es decir, si el programa es inherentemente secuencial, no habrá posible paralelización. En cambio si escribimos código paralelizable, el compilador le sacará partido.

- El programador indica las directivas de compilación para orientar el modo de paralelización-vectorización deseado y para informar de la no existencia de dependencias entre datos, pues el compilador es, por razones obvias, fuertemente conservador.

- En tiempo de ejecución, distintos procesadores ejecutan simultáneamente distintas iteraciones de un mismo bucle paralelizado. Si el coste de las iteraciones asignadas es diferente, los procesadores terminan en instantes diferentes y la eficiencia decrece. Posteriormente se realiza una sincronización, necesaria para mantener la consistencia de las operaciones.

El lenguaje utilizado para la implementación de los algoritmos es el Fortran. Se trata de Fortran 77, con las extensiones comúnmente aceptadas para el manejo de vectores y matrices [Alliant

88, Alliant 89]. El compilador es capaz de optimizar el código secuencial, paralelizando y vectorizando los bucles. Concretamente hay tres modos independientes de optimización:

- Optimización global (directiva **g**): del estilo de la realizada por los compiladores secuenciales clásicos.

- Optimización por vectorización (directiva **v**): se genera código para bucles que realiza de forma vectorial las operaciones sobre vectores y matrices.

- Optimización por concurrencia (directiva **c**): se genera código para bucles que realiza de forma paralela las operaciones sobre vectores y matrices.

Evidentemente, las mayores prestaciones se obtienen con los tres modos activos. Las optimizaciones de concurrencia y vectorización se combinan de dos modos:

- En caso de un sólo bucle, sin anidación, o en varios bucles anidados, cuando no es posible hacerlo de otra forma, el modo elegido es paralelizar y vectorizar el bucle, vectorizando por segmentos y ejecutando paralelamente todos los segmentos posibles. Es el llamado *Vector-Concurrent*.

- En caso de al menos dos bucles anidados optimizables, se prefiere el modo denominado COVI (*Concurrent-Outer, Vector-Inner*), paralelizándose el bucle más externo y vectorizando el más interno. Se consigue entonces la máxima eficacia del computador.

El compilador, al activar el usuario las directivas de compilación citadas, toma las decisiones que cree más convenientes, asegurándose que no existen dependencias entre datos. Éstas son el problema principal para paralelizar cualquier algoritmo, y aquí especialmente, de modo que el programador puede indicar la presencia o no de esas dependencias mediante directivas, que aparecerán en el programa como comentarios especiales del tipo: CVD$ <tab> Directiva. Exponemos ahora las empleadas en los programas:

- NoConcur: Indica que el siguiente bucle no debe paralelizarse. Se utiliza, por ejemplo, en bucles pequeños, pues el resultado sería menos eficiente.

- NoVector: Idem para la vectorización.

- NoSync: Indica, de forma más potente que la directiva NoDepCheck, que no existen dependencias entre los datos, de modo que pueden ejecutarse asíncronamente las distintas iteraciones del bucle afectado.

Las directivas no sólo pueden afectar a bucles, sino también a rutinas.

4.4.2 SGI Power Challenge

Descripción de la arquitectura

El computador PowerChallenge de Silicon Graphics es un multiprocesador con memoria compartida que consta de 16 procesadores superescalares de tipo MIPS R10000 [Silicon 94].

La antememoria de este computador está estructurada en dos niveles. El primer nivel está compuesto por una antememoria de datos interna de 16 Kbytes destinada al almacenamiento de datos enteros.

En el segundo nivel de antememoria se encuentra una memoria (o antememoria externa de datos) de 2 Mbytes (con posibilidad de ampliación a 4 Mbytes) destinada a almacenar datos en coma flotante y los datos enteros. La carga y descarga de datos en coma flotante se efectúa directamente sobre la antememoria externa sin necesidad de que éstos pasen por la antememoria interna.

La memoria RAM puede consistir de 1 hasta 8 módulos. En cada módulo se pueden instalar 8 bancos de 64 ó 256 Mbytes. Así, en total, la memoria de esta arquitectura puede variar entre 64 Mbytes y 16 Gbytes.

El MIPS R10000 es un procesador superescalar con tecnología RISC. La velocidad pico de computación de este procesador es de 360 Mflops. Las características generales de este procesador pueden resumirse en las siguientes:

Es capaz de ejecutar hasta 6 operaciones por ciclo de reloj, 2 de éstas con accesos a memoria, otras 2 en la unidad entera y las 2 restantes en la unidad en coma flotante.

Incorpora una antememoria de instrucciones interna de 16 Kbytes y una antememoria de datos de 4 Mbytes.

Dispone de una unidad funcional para el cálculo con enteros con 32 registros, una unidad funcional con otros 32 registros para el cálculo con datos en coma flotante, así como una unidad de gestión de memoria.

El bus de datos es de 256 bits y el bus de direcciones de 40 bits.

Incorpora un coprocesador que trabaja con aritmética IEEE de 64 bits, tanto para enteros como para números en coma flotante.

Es posible describir las unidades funcionales de cálculo de manera más detallada identificando los siguientes aspectos:

Unidad de cálculo con enteros. Esta unidad dispone de dos unidades aritmético-lógicas, una unidad para la multiplicación y división de enteros y una unidad de ramificación (cálculo de direcciones en instrucciones de salto). Todas estas unidades permiten el desarrollo de hasta 4 instrucciones por ciclo, por ejemplo, dos operaciones con enteros y dos accesos a datos.

Unidad de cálculo en coma flotante. Esta unidad contiene dos unidades segmentadas que permiten realizar simultáneamente 2 operaciones en coma flotante.

Los procesadores utilizan el POWERpath-2 como un protocolo de interconexión encargado de comunicar entre si los procesadores, sincronizar los mismos y realizar transferencias. Algunas de las características de la interconexión son:

1. Velocidad de trasferencia sostenida de 1,2 Gbyte/s.
2. Coherencia de escritura en antememoria mantenida por hardware.
3. Buses independientes de 256 bits (bus de datos) y 40 bits (bus de direcciones).
4. Memoria, antememorias y buses con códigos de detección de errores.
5. Sistema de señales síncronas a 47,6 MHz.

Entorno de programación

El sistema operativo IRIX ofrecido con el Silicon Graphics PowerChallenge es una versión multitarea y multiusuario del Unix V de la Universidad de California en Berkeley.

El compilador de Fortran disponible cumple el estándar ANSI que define el Fortran-77. Realiza su trabajo optimizador sobre código fuente estrictamente secuencial, detectando automáticamente fragmentos de código a partir de los cuales puede generar código objeto ejecutable en paralelo. Junto a este lenguaje podemos encontrar a nuestra disposición la librería optimizada BLAS así como la librería LAPACK.

El Silicon Graphics PowerChallenge dispone además de la librería de comunicaciones PVM y MPI por lo que, en la práctica, puede programarse también como un multiprocesador con memoria distribuida. Las comunicaciones en este caso se realizan a través de la memoria compartida.

4.4.3 Cray T3D

Descripción de la arquitectura

El Cray T3D es un multiprocesador con memoria virtualmente compartida desarrollado por Cray Research, perteneciente a la tipología de las máquinas masivamente paralelas [Cray 93c].

Alguna de sus características más relevantes son,

1. Multiprocesador MIMD.
2. Memoria físicamente distribuida pero direccionable globalmente.
3. Sincronización de bajo coste.
4. Alto ancho de banda y baja latencia.
5. Entrada/Salida de datos con alto ancho de banda.
6. Escalabilidad hasta miles de procesadores.

Este computador puede constar de hasta 2048 nodos, compuestos por un procesador superescalar de tipo DEC Alpha AXP 6220 y una circuitería de control, por lo cual resulta altamente escalable. Cada uno de los procesadores dispone de un micronúcleo de UNIX que se encarga básicamente del control de las comunicaciones entre los procesadores y de algunas operaciones sobre la memoria.

La red de interconexión forma una topología de toro bidireccional en 3 dimensiones. La red tiene un ancho de banda de 300 Mbytes/seg. en cualquiera de sus enlaces. La circuitería disponible en cada uno de los nodos permite que el computador funcione como una máquina con memoria virtualmente compartida. Así toda la memoria de este computador es direccionable globalmente, es decir, cada procesador puede direccionar localmente su propia memoria, pero además también puede acceder a la memoria de otro procesador por *hardware* sin necesidad de que éste último intervenga.

El DEC Alpha AXP es un microprocesador superescalar de tipo RISC compuesto por varias unidades funcionales independientes supersegmentadas, esto es, utilizan un mismo mecanismo de segmentación para el procesamiento de ciertos tipos de instrucciones. Este procesador tiene un pico computacional de 150 Mflops y es capaz de realizar varias operaciones por ciclo de reloj. Además el procesador dispone de una antememoria de datos y una antememoria de instrucciones capaces de almacenar 256 palabras de 64 bits cada una.

Las principales unidades funcionales son:

- **Unidad de cálculo con enteros.** Se encarga del desarrollo de operaciones con enteros de 64 bits, incluyendo operaciones aritméticas, lógicas, de comparación y de desplazamiento.

- **Unidad de cálculo en coma flotante.** Se encarga del desarrollo de operaciones con números en coma flotante de 64 bits. Estas operaciones incluyen aritmética en coma flotante, e instrucciones para la conversión de números de tipo entero y coma flotante.

- **Unidad de generación de instrucciones.** Realiza el cálculo de direcciones, actúa como tampón de escritura y se encarga de la gestión de las operaciones de carga y almacenamiento.

- **Unidad de búsqueda de instrucciones.** Realiza la búsqueda de instrucciones así como su lanzamiento, se encarga del control de los recursos, el cálculo del contador de programa y la predicción de saltos.

Entorno de programación

El usuario no tiene acceso al sistema operativo ni las herramientas de compilación del Cray T3D. Este computador dispone de un Cray Y-MP o un Cray C90 sobre el que se realizan estas tareas, conectado al Cray T3D mediante distintos tipos de redes (Ethernet, HiPPI, etc.). El computador Cray Y-MP o Cray C90 trabaja bajo el entorno UNICOS. Sólo el código ejecutable puede lanzarse en un sistema de colas y operar directamente sobre el Cray T3D.

El compilador de Fortran disponible cumple el estándar ANSI que define el Fortran-77. Junto a este lenguaje se dispone de la librería BLAS, y parcialmente de la librería LAPACK. Se ha podido comprobar que la optimización de algunas de las subrutinas de la librería BLAS deja mucho que desear.

El Cray T3D dispone además de la librería de comunicaciones PVM que permite utilizar esta máquina como un entorno de programación de paso de mensajes [Cray 93a]. Además esta máquina puede utilizarse también bajo un entorno de memoria virtualmente compartida mediante la librería *Data Sharing* [Cray 93a,Cray 93b].

4.4.4 IBM SP2

Descripción de la arquitectura

El IBM SP2 es un claro representante de los multiprocesadores con memoria distribuida o multicomputadores. Éste está basado en la tecnología de las estaciones de trabajo RS/6000. El que hemos utilizado dispone de 80 procesadores, pudiendo ampliarse hasta 256, y 256 Mbytes de

memoria RAM por procesador. La forma de funcionamiento es conocida como espacio compartido (*space-sharing*):

El multicomputador tiene un único planificador que asigna procesadores y reserva un tiempo de éstos a los usuarios según sus peticiones.

Los procesadores que componen este multicomputador se pueden clasificar en tres grupos diferentes:

- Servidores de compilación.
- Servidores de entrada/salida.
- Nodos para aplicaciones de usuarios.

Existen 3 redes de conexión diferentes que comunican todos estos nodos, exceptuando los servidores de compilación que únicamente se hallan conectados a la red Ethernet. Las 3 redes disponibles son:

- **HPS3** (*High Performance Switch* 3). Latencia: 51 μs. Ancho de banda: 53.6 Mbyte/s.

  La red HPS3 es el componente hardware que hace de esta agrupación de computadores RS/6000 un verdadero computador paralelo. Esta red está construida a partir de conmutadores bidireccionales 4×4. La unión de 8 conmutadores de estos forman una plataforma de conmutación a la cual se pueden conectar 16 procesadores. Estas plataformas de interconectan entre ellas para formar una red omega multietapa. La conmutación de paquetes a través de la red se realiza mediante un encaminamiento fijo de tipo *buffered wormhole* .

- **FDDI.** Latencia: 2 ms. Ancho de banda: 10 Mbyte/s.
- **Ethernet**. Latencia: 1 ms. Ancho de banda: 1 Mbyte/s.

En las dos últimas redes, el ancho de banda es compartido entre todos los nodos. Por el contario, en el SPH3 el ancho de banda es el existente entre cada par de procesadores.

Cada nodo de procesamiento de esta máquina es un procesador IBM SP2 RS/6000 POWER2 con las siguientes características:

- CPU superescalar RISC de 64 bits a 120 MHz.
- 32 Kbytes de antememoria para instrucciones y 32 Kbytes de antememoria para datos.
- Velocidad punta: 240 Mflops. Este procesador es capaz de realizar dos flops en aritmética de doble precisión por cada ciclo de reloj.
- Memoria central local RAM de 256 Mbytes.
- Adaptador de comunicaciones al HPS3. Controlado por la CPU a través de un bus micro chanel. Por lo tanto no es un procesador de comunicaciones completamente autónomo.
- Disco local de 1 Gbyte para el sistema operativo, área de *swap* y disco temporal.

Entorno de programación

Cada procesador ejecuta el sistema operativo tipo Unix en su versión AIX 3.2.5 de IBM. Actualmente el SP2 no tiene memoria globalmente distribuida, por tanto sólo se puede programar utilizando paso de mensajes y HPF. Los lenguajes de programación que posee este ordenador son:

- Lenguaje C.
- Lenguaje C++.
- Lenguaje Fortran-77.
- Lenguaje Fortran-90.

Los entornos paralelos de que dispone este multicomputador son los siguientes:

MPL (Message-Passing Library): Biblioteca de comunicaciones propia de IBM. No es portátil y, además, es de bajo nivel.

P4 (*Programming Portable Parallel Programs*): Biblioteca de comunicaciones desarrollada en el Laboratorio Nacional Argonne que sirve tanto para multiprocesadores como multicomputadores.

Chameleon: Biblioteca de comunicaciones, también desarrollada en el Laboratorio Nacional Argonne, basada en la anterior pero con numerosas características adicionales de alto nivel que facilitan la generación de programas paralelos.

Fortran M: Entorno de programación paralelo basado en el lenguaje Fortran. Es un conjunto de extensiones adicionales que permiten la creación de programas paralelos mediante la interconexión de procesos utilizando canales de comunicación.

MPI (*Message Passing Interface*): Biblioteca estándar de comunicaciones entre distintos procesadores. En el IBM SP2 existen dos versiones de esta biblioteca: la desarrollada por IBM y la desarrollada por varios investigadores del mencionado laboratorio.

4.5 Conclusiones

En este capítulo se han presentado taxonomías que agrupan los computadores atendiendo a su capacidad para procesar datos e instrucciones, a la gestión que realizan del espacio de direcciones de memoria y a su utilización en la computación científica. La segunda clasificación nos permite detectar un tipo de arquitectura con altas prestaciones computacionales: los multiprocesadores.

La programación de los multiprocesadores con memoria compartida no resulta muy diferente de la programación secuencial. En cambio la programación de los multicomputadores, si se quiere aprovechar al máximo los recursos de la máquina, debe realizarse bajo un entorno de paso de mensajes y este modelo presenta una programación mucho más compleja.

Uno de los parámetros principales que determina las prestaciones de los algoritmos paralelos sobre multicomputadores es la distribución de los datos. Tres han sido las distribuciones propuestas, distribución por bloques de columnas (filas), distribución cíclica por bloques de columnas (filas) y distribución toroidal 2-D.

La distribución por bloques de filas (columnas) es válida cuando se aplican algoritmos de frente de onda, aunque ello obligue a una redistribución de los datos durante la resolución del problema.

La distribución cíclica por bloques de filas (columnas) resulta especialmente apropiada para problemas de dimensión moderada.

La principal ventaja de la distribución toroidal 2-D es su escalabilidad, que la hace adecuada para tratar problemas de gran dimensión. La combinación de una distribución de datos apropiada, y un uso coherente de las subrutinas de comunicación, facilita en gran medida el desarrollo de algoritmos paralelos.

Por último, el estudio de la arquitectura y de las prestaciones de la máquina paralela sobre la cual se desea desarrollar un algoritmo es una de las tareas básicas en la programación. Así, se han descrito en este capítulo las características más importantes de las máquinas utilizadas durante la realización de esta tesis.

Capítulo 5

Algoritmos paralelos para resolver la ecuación de Lyapunov

5.1 Introducción

En este capítulo se describen las diversas aproximaciones para resolver las ecuaciones de Lyapunov en computadores paralelos y distribuidos utilizando el método de Hammarling y la función signo matricial.

En la sección 5.2 se presentan algoritmos paralelos basados en el método de Hammarling. Después del análisis de la dependencia de datos de este método en el apartado 5.2.1, se presentan en el apartado 5.2.2 algoritmos de grano fino y medio para multiprocesadores con memoria compartida. En los apartados 5.2.3.1 a 5.2.3.4 se describen algoritmos paralelos para multicomputadores. Los algoritmos presentados en los apartados 5.2.3.1 y 5.2.3.2 están basados en una distribución de los datos por bloques y resolución por frente de onda de antidiagonales. En los apartados 5.2.3.3 y 5.2.3.4 se describen algoritmos basados en la distribución cíclica de datos. La descripción de los algoritmos se ha realizado utilizando lenguaje pseudo-algorítmico como en el capítulo 3. En el caso de los algoritmos en los que se usa el paso de mensajes, se utilizan las pseudo-rutinas de comunicación descritas en el capítulo 4.

En la sección 5.3 se presentan los algoritmos paralelos utilizando el método de la función signo matricial. En el apartado 5.3.1 se describe como trabaja la librería ScaLAPACK. A continuación se describen los algoritmos para resolver la ecuación de Lyapunov obteniéndose la solución explícita (apartado 5.3.1) o para obtener el factor de Cholesky (apartado 5.3.2). Además se describen modificaciones de estos algoritmos para resolver la ecuación generalizada de Lyapunov (apartado 5.3.3) y para resolver las ecuaciones de Lyapunov acopladas (apartado 5.3.4). Dado que la implementación de estos algoritmos se ha realizado utilizando el ScaLAPACK, se presentarán básicamente las rutinas esta librería utilizadas en los pasos más importantes de los estos algoritmos.

5.2 Paralelización del Método de Hammarling

Como vimos en el capítulo 3 el método de Hammarling [Hammarling 82] nos permite resolver las ecuaciones de Lyapunov en tiempo continuo

$$A^T X + XA + C^T C = 0 , \tag{5. 1}$$

y en tiempo discreto

$$A^T XA - X + C^T C = 0 , \tag{5. 2}$$

obteniendo directamente el factor de Cholesky de la solución *X*, sin necesidad de realizar explícitamente el producto $C^T C$.

El primer paso en la resolución de las ecuaciones de Lyapunov mediante éste método es la reducción de la matriz de estados *A* a la forma real de Schur [Golub 89]. Así, es posible calcular una matriz *Q* ortogonal, tal que

$$A = QSQ^T ,$$

donde *S* será una matriz triangular superior por bloques, con bloques diagonales 1x1 ó 2x2, según se trate respectivamente de los valores propios reales de *A* o de bloques asociados a pares de valores propios complejos conjugados.

El algoritmo base de casi todas las implementaciones secuenciales actuales (también para computadores paralelos con memoria compartida) para la obtención de la forma real de Schur de una matriz general es el algoritmo iterativo QR de Francis con doble desplazamiento implícito [Francis 61]. Como paso previo a este algoritmo debemos reducir la matriz a la forma de Hessenberg [Golub 89][Dongarra 91]. Existen otros métodos iterativos basados en algoritmos tipo Jacobi [Stewart 85]. Éstos sin embargo no son convergentes globalmente [Eberlein 87] [Stewart 85] y poseen mayor coste computacional, dado que requieren un número más elevado de iteraciones (convergencia no cuadrática), aunque son muy competitivos para matrices simétricas [Berry 86] [Brent 85]. Actualmente se están replanteando algunas de estas ideas al comprobarse que las condiciones de parada para el algoritmo QR no son las adecuadas. Esto lleva a dar por finalizado el algoritmo QR antes de tiempo, con los consiguientes errores que acarrea. Un replanteamiento de las condiciones de finalización del algoritmo QR [Tisseur 96] nos lleva a tiempos similares entre éste y los algoritmos tipo Jacobi [Ahues 97].

En el caso de los multicomputadores se han hecho diversas propuestas del algoritmo QR encaminadas a una mayor reutilización de los datos. Una de ellas está basada en el incremento del número de desplazamientos en cada iteración del algoritmo, por ejemplo, mediante el uso de un algoritmo QR con múltiple desplazamiento implícito [Bai 89], con una multiplicidad $m \leq 6$. Este procedimiento tiene el problema de introducir mayor inexactitud en la elección de los desplazamientos elegidos para producir un mayor rapidez en la convergencia del algoritmo QR, y de un moderado incremento de coste.

Otro procedimiento es la aplicación de dobles desplazamientos de forma segmentada [Watkins 94]. Este método aporta unas notables ventajas para su implementación paralela (distribuida) sobre un número reducido de procesadores (computadores) con una distribución cíclica por bloques de filas o columnas, frente a los problemas de eficiencia en las versiones paralelas (distribuidas) del algoritmo QR con un único doble desplazamiento.

Para obtener un buen rendimiento paralelo de algoritmo QR con doble desplazamiento, van de Geijn propone un conjunto de estrategias de almacenamiento de datos por bloques antidiagonales [Geijn 88], lo que denomina bloques de Hankel. Esta organización también es posible utilizarla con buenos resultados para la etapa previa de reducción a la forma de Hessenberg y en los algoritmos tipo Jacobi.

Recientemente Ralha ha propuesto nuevos algoritmos para la reducción de matrices simétricas a la forma tridiagonal y la posterior obtención del sistema de valores propios utilizando transformaciones tipo Jacobi aplicadas sólo a un lado de la matriz [Ralha 95]. Así mismo, ha propuesto algoritmos aplicados por un lado de la matriz para la reducción de matrices no simétricas a la forma de Hessenberg utilizando transformaciones de Householder [Ralha 93], aunque su utilización para la reducción a la forma real de Schur es un problema de investigación actual.

La paralelización del algoritmo iterativo QR es un campo abierto donde se están produciendo nuevos avances continuamente. Este tema de por si presenta suficiente complejidad e interés para constituir el tema de una tesis. Nuestra tesis centra el análisis y estudio del método de Hammarling en la 2ª etapa de este método, es decir, en la resolución de la ecuación de Lyapunov reducida, donde la matriz de coeficientes está en forma real de Schur, y en métodos alternativos, basados en la función signo matricial, para resolver esta ecuación.

5.2.1 Dependencia de datos en el algoritmo de Hammarling

En este apartado analizamos la dependencia de datos existente en el algoritmo serie de Hammarling. Para este estudio elegiremos la ecuación de Lyapunov asociada a los sistemas en tiempo discreto, que aparece menos estudiada en la bibliografía existente. Sin embargo no hay pérdida de generalidad en nuestro análisis debido a la similitud de resolución entre los dos tipos de ecuaciones (tiempo continuo y tiempo discreto), lo que da como resultado una dependencia de datos similar.

Partiremos de la ecuación reducida de Lyapunov en tiempo discreto similar a la descrita en la ecuación (5.2)

$$S(LL^T)S^T - (LL^T) + RR^T = 0\,, \tag{5.3}$$

donde $S \in \mathsf{R}^{nxn}$ es la forma real de Schur de A, $\Lambda(A) = \{\lambda_i : |\lambda_i| < 1\}$, $R \in \mathsf{R}^{nxn}$ es triangular inferior (semi)definida positiva y $L \in \mathsf{R}^{nxn}$ es el factor de Cholesky de la solución X, $X = LL^T$.

Siguiendo el algoritmo de Hammarling dividimos las matrices S, L y R en los siguientes bloques

$$S = \begin{pmatrix} \lambda_{11} & 0 \\ \mathbf{s} & S_1 \end{pmatrix},\ L = \begin{pmatrix} \ell_{11} & 0 \\ \mathbf{l} & L_1 \end{pmatrix},\ R = \begin{pmatrix} \gamma_{11} & 0 \\ \mathbf{r} & R_1 \end{pmatrix}, \tag{5.4}$$

donde λ_{11} será un escalar (valor propio real de A, $\lambda_{11} = s_{11}$) o un bloque 2×2 (asociado a dos valores propios complejos conjugados). En el primer caso, también serán escalares $\ell_{11} = l_{11}$ y $\gamma_{11} = r_{11}$, siendo **s**, **l** y **r** vectores columna de *n*-1 elementos. En el segundo caso, serán bloques 2×2 tanto ℓ_{11} como γ_{11}, siendo **s**, **l** y **r** bloques de tamaño $(n-2) \times 2$.

En adelante, y por simplicidad, consideraremos que todos los valores propios de la matriz S son reales, por lo que se tratará de una matriz triangular inferior. De las ecuaciones (5.3) y (5.4) podemos deducir las siguientes ecuaciones

$$(s_{11}^2 - 1)l_{11}^2 = -r_{11}^2, \tag{5.5}$$

$$(s_{11}S_1 - I_{n-1})\mathbf{l} = -\alpha\mathbf{r} - \beta\mathbf{s}, \tag{5.6}$$

$$S_1 (L_1 L_1^T) S_1^T - (L_1 L_1^T) + RR^T = 0, \tag{5.7}$$

donde

$$\alpha = r_{11} l_{11}^{-1},\ \beta = s_{11} l_{11},\ RR^T = R_1 R_1^T + \mathbf{y}\mathbf{y}^T,\ \mathbf{y} = \alpha\mathbf{v} - s_{11}\mathbf{r},\ \mathbf{v} = S_1^T\mathbf{l} + \mathbf{s}l_{11}.$$

El elemento diagonal l_{11} se calcula directamente de la ecuación (5.5). Entonces, podemos resolver el sistema lineal triangular inferior de la ecuación (5.6) mediante sustitución progresiva y obtener el vector **l**. Por último, la ecuación (5.7) es una ecuación de Lyapunov en tiempo discreto de orden n-1 donde la matriz R tiene la estructura

$$R = \begin{pmatrix} R_1 & \mathbf{y} \end{pmatrix};$$

es decir, está formada por una matriz triangular inferior de orden $n-1 \times n-1, R_1$, y un vector columna de n-1 elementos, **y**. Podemos obtener el factor de Cholesky $\hat{R}$ del producto RR^T mediante una descomposición LQ:

$$R = \begin{pmatrix} R_1 & \mathbf{y} \end{pmatrix} = \begin{pmatrix} \hat{R} & 0 \end{pmatrix} \hat{Q},$$

donde $\hat{Q} \in \mathsf{R}^{n \times n}$ una matriz ortogonal y $\hat{R} \in \mathsf{R}^{n-1 \times n-1}$ es triangular inferior. La nueva ecuación reducida de Lyapunov de orden n-1

$$S_1 (L_1 L_1^T) S_1^T - (L_1 L_1^T) + \hat{R}\hat{R}^T = 0,$$

puede tratarse otra vez del mismo modo hasta que el problema está completamente resuelto. A continuación presentamos en forma de grafo la dependencia de datos del método de Hammarling para la ecuación de Lyapunov en tiempo discreto de orden n= 4.

Como se puede observar, el algoritmo de Hammarling está orientado a columnas. Esto quiere decir que si consideramos la j-ésima columna de L, es necesario conocer los elementos $L(j{:}i\text{-}1,j)$ antes de calcular el elemento $L(i,j)$. Si ahora consideramos el cálculo de la (j+1)-ésima columna L, primero es necesario calcular el elemento $L(j+1,j+1)$, que sólo precisa como nuevo dato del valor de $R(j+1,j+1)$ modificado después del cálculo de $L(j+1,j)$ en la iteración j. A continuación calculamos el elemento $L(j+2,j+1)$ para el que necesitamos el nuevo valor de $R(j+2,j+1)$ modificado después del cálculo de $L(j+2,j)$. Y así sucesivamente hasta completar el cálculo de la columna j+1. Los nodos etiquetados con $L(i,j)$ suponen el cálculo de un nuevo elemento de la matriz de incógnitas L. Los nodos etiquetados como $R(i,j)$ suponen la actualización de un elemento de la matriz de términos independientes R.

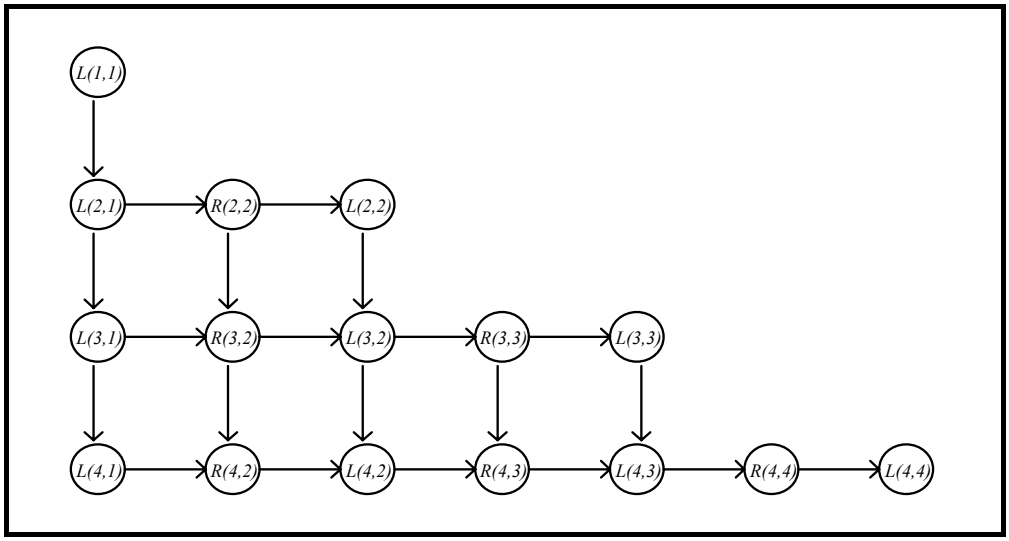


Figura 9. *Grafo de dependencia de datos para n=4 del algoritmo de Hammarling.*

Algoritmo 10 *[SH]*

para j= 1, n

1. Cálculo del elemento diagonal de la fila i.

$$\alpha = \sqrt{1 - S(j,j)^2}\ ,\ L(j,j) = \alpha^{-1} R(j,j)\ ,\ \beta = L(j,j)S(j,j)$$

2. Cálculo de los elementos subdiagonales de la columna j.

para i= j+1, n

$$L(i,j) = \left(-\alpha\, R(i,j) - \beta\, S(i,j) - S(j,j) \sum_{k=j+1}^{i-1} S(i,k)L(k,j)\right) / (S(i,i)S(j,j)-1)$$

fin para

3. Actualización de la matriz R para el siguiente paso.

3.1. Cálculo del vector **y**.

para i= j+1, n

$$y(i) = \alpha \left(L(j,j)S(i,j) + \sum_{k=j+1}^{i} S(i,k)L(k,j)\right) - S(j,j)\, R(i,j)$$

fin para

3.2. Cálculo y aplicación de las rotaciones de Givens que anulen **y**.

para i= j+1, n

3.2.1. Cálculo una la rotación de Givens tal que:

$$\begin{bmatrix} R(i,i) & y(i) \end{bmatrix} \begin{bmatrix} \cos\theta_i & \operatorname{sen}\theta_i \\ -\operatorname{sen}\theta_i & \cos\theta_i \end{bmatrix} = \begin{bmatrix} * & 0 \end{bmatrix}$$

3.2.2. Aplicación de la rotación de Givens a la columna de R.

para k= i, n

$$\begin{bmatrix} R(k,i) & y(k) \end{bmatrix} = \begin{bmatrix} R(k,i) & y(k) \end{bmatrix} \begin{bmatrix} \cos\theta_i & \operatorname{sen}\theta_i \\ -\operatorname{sen}\theta_i & \cos\theta_i \end{bmatrix}$$

fin para

fin para

fin para

fin

Del análisis del grafo de dependencias es posible observar que la secuencia de cálculo en la que se explota el máximo paralelismo inherente en el problema se obtiene cuando los elementos pertenecientes a la misma antidiagonal de L son calculados simultáneamente. Esta secuencia de cálculo se muestra en la Figura 10, donde las celdas se etiquetan con el correspondiente orden de cálculo en la secuencia.

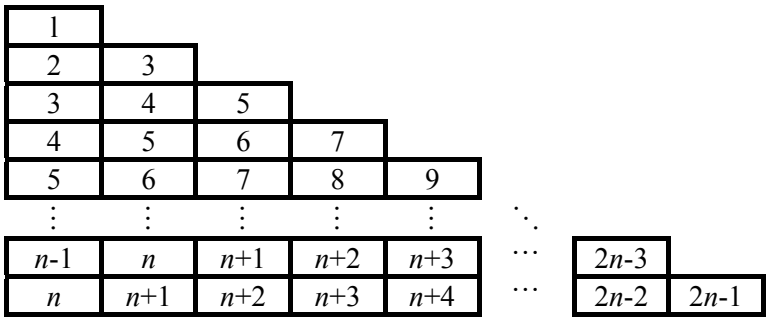


Figura 10. *Secuencia de resolución por antidiagonales de los elementos de L.*

Esta idea fue propuesta anteriormente por O´Leary [O'Leary 85] para la resolución de sistemas lineales triangulares en multiprocesadores con memoria distribuida y se conoce como *frente de onda de antidiagonales*. Nosotros la utilizaremos para calcular simultáneamente la solución de sistemas lineales triangulares y descomposiciones *LQ* [Claver 94].

5.2.2 Multiprocesadores con memoria compartida

Utilizando multiprocesadores con memoria compartida realizaremos dos aproximaciones para la resolución de la ecuación de Lyapunov en tiempo discreto siguiendo la idea de frente de onda de antidiagonales. En la primera todos los elementos de una antidiagonal de L se calculan simultáneamente en cada paso. De esta primera aproximación obtendremos algoritmos de grano fino apropiados para arquitecturas paralelas basadas en procesadores escalares o superescalares. En la segunda aproximación las columnas de la matriz L se dividen en subvectores de longitud t (de tamaño fijo o variable) y se realiza el cálculo completo de una antidiagonal de estos subvectores en cada paso. Este algoritmo de grano medio está especialmente indicado para multiprocesadores con unidades vectoriales.

5.2.2.1 Algoritmos paralelos de grano fino

El primer enfoque para resolver de forma paralela la ecuación de Lyapunov en tiempo discreto lo haremos orientado a elementos y a los algoritmos que diseñemos les daremos el nombre de

algoritmos de grano fino. Esto es debido a que, en cada fase paralela de la resolución del problema, cada procesador se encarga de obtener un elemento de la matriz solución *L*.

Del grafo de dependencias del método de Hammarling presentado en la Figura 9 se deduce que un elemento L_{ij} de la matriz resultado se puede calcular si se han calculado los elementos $\{L_{kl} : k \le i,\ l \le j\}$, y se han actualizado los elementos $\{R_{kj} :\ j \le k \le i\}$. Todos los elementos que cumplan estas premisas en un momento dado del proceso de resolución se podrán calcular concurrentemente. En particular, todos los elementos pertenecientes a una misma antidiagonal cumplen estas condiciones. Nuestra propuesta consiste en seguir una secuencia de cálculo de los elementos que pertenecen a una misma antidiagonal o que cumplan las condiciones anteriores. De la forma en que materialicemos estas ideas dependerán las diversas versiones de este procedimiento.

En una primera aproximación a la resolución del problema, abordamos el cálculo en paralelo de los elementos de una misma antidiagonal. El número de antidiagonales de una matriz *m*x*n* es de *nadiag*= *m*+*n*-1. En nuestro caso, la matriz es $n \times n$ y *nadiag*= 2*n*-1. Los elementos pertenecientes a una antidiagonal dada (*adiag*) de la matriz *L*, si tenemos en cuenta que ésta es triangular inferior, son

$$\{L_{ij} : [(adiag+2)/2] \le i \le \text{mín}(adiag,n),\ j= adiag - i+1\}.$$

Todos estos elementos están, lógicamente, por debajo de la diagonal principal de *L* o pertenecen a ella.

Nuestro algoritmo *PHWF* realiza un barrido secuencial a través de las *nadiag* antidiagonales de *L* siguiendo la secuencia de la Figura 10 y calcula los elementos pertenecientes a la antidiagonal en curso mediante el procedimiento *PGF*. En este procedimiento no necesitamos tener la variable de tipo vector *y*(:), sustituyéndola por una variable escalar, que la podemos almacenar sobre el elemento *R*(*i,j*), que ya no va a ser utilizado con posterioridad. Por otra parte, en el algoritmo serie sólo necesitábamos dos variables escalares para almacenar el *seno* y el *coseno* de las transformaciones de Givens. Ahora es necesaria la utilización de dos matrices $n \times n$ triangulares inferiores para poder almacenar las rotaciones que actualizan la matriz *R*.

Procedimiento 1 *PGF*(*i,j*): Calcula el elemento *L*(*i,j*).
si (*i*=*j*) entonces
1. Cálculo del elemento diagonal de la columna *j*.

$$\alpha(j) = \sqrt{1 - S(j,j)^2}\ ,\ L(j,j) = \alpha(j)^{-1} R(j,j)\ ,\ \beta(j) = L(j,j)S(j,j)$$

sino
2. Cálculo del elemento subdiagonal de la columna *j*.

$$L(i,j)= \left(-\alpha(j)\ R(i,j) - \beta(j)S(i,j) - S(j,j) \sum_{k=j+1}^{i-1} S(i,k)L(k,j)\right)/(S(i,i)S(j,j) -1)$$

3. Actualización de la matriz *R* para el siguiente paso.
3.1. Cálculo del escalar *y* (utilizando *R*(*i,j*) para actualizarlo).

$$y= \alpha(j) \left(L(j,j)S(i,j) + \sum_{k=j+1}^{i} S(i,k)L(k,j)\right) - S(j,j)R(i,j)$$

3.2. Aplicación de las rotaciones anteriores sobre *R*(*i,j*+1:*i*-1) e *y*:
para *k*= *j*+1,*i-1*

$$\begin{bmatrix} R(i,k) & y \end{bmatrix} = \begin{bmatrix} R(i,k) & y \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_{kj} & \text{sen}\,\theta_{kj} \\ -\text{sen}\,\theta_{kj} & \cos\theta_{kj} \end{bmatrix}$$

fin para

3.3. Cálculo y aplicación de la rotación de Givens ($\cos\theta_{ij}$,$\text{sen}\,\theta_{ij}$) tal que:

$$\begin{bmatrix} R(i,i) & y \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_{ij} & \text{sen}\,\theta_{ij} \\ -\text{sen}\,\theta_{ij} & \cos\theta_{ij} \end{bmatrix} = \begin{bmatrix} * & 0 \end{bmatrix}$$

fin si

fin

La secuencia de resolución de la matriz *L* si disponemos de $p \geq n/2$ procesadores tiene un número de pasos igual al número de antidiagonales *nadiag* (ver Figura 11). En el caso habitual de que dispongamos de un número menor de procesadores, cada antidiagonal se calcula en más de un paso (ver Figura 12). Esto provoca que para las antidiagonales con un número de elementos que no sea múltiplo del número de procesadores exista, en consecuencia, una pérdida importante en la eficiencia del algoritmo.

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

Figura 11. *Secuencia de resolución de los elementos de L para el algoritmo PHWF (n=10 y p=n/2).*

Por otra parte, el acceso a zonas de memoria muy distantes entre sí para matrices de medio y gran tamaño, al no poder resolver los elementos de una misma antidiagonal de forma completa sino en varias etapas, provocará multitud de cambios de contexto en computadores con antememoria, con la consiguiente pérdida de tiempo en las actualizaciones.

1									
2	3								
3	4	5							
4	5	6	7						
5	6	7	9	11					
6	7	9	11	13	15				
8	9	11	13	15	17	19			
10	12	13	15	17	19	21	22		
12	14	16	17	19	21	22	23	24	
14	16	18	20	21	22	23	24	25	26

Figura 12. *Secuencia de resolución de los elementos de L para el algoritmo PHWF (n=10 y p=3).*

Algoritmo orientado a bloques de columnas

Para evitar los problemas de localidad en los accesos a elementos de memoria lejanos entre si y teniendo en cuenta que el método de Hammarling está orientado a columnas, proponemos ahora un algoritmo orientado a bloques de columnas. Con ello conseguiremos aumentar la localidad de los accesos para matrices de tamaño elevado. Por otra parte, si elegimos de forma adecuada los bloques de

columnas, evitaremos, en lo posible, los inconvenientes del algoritmo anterior cuando el número de elementos de una antidiagonal no es múltiplo del número de procesadores. Esto nos indica que el tamaño de los bloques de columnas (*tbcol*) para que no aparezca este efecto debe ser múltiplo del número de procesadores, *p.*

El número de bloques de columnas (*nbcol*) será $nbcol = \lfloor (n-1)/tbcol \rfloor + 1$. Dado un bloque de columnas *bcol*, el número de antidiagonales de este bloque *nadiag* será

$$nadiag = n - tbcol\,(bcol-1) + \text{mín}(tbcol,\ n - tbcol(nbcol-1)) - 1.$$

El bloque de columnas *bcol* contiene las antidiagonales:

$$adiag = 2\ tbcol\,(bcol-1) + 1, \ldots, \text{mín}(n + tbcol \cdot bcol - 1,\ 2n-1).$$

Los elementos que calculamos para una antidiagonal *adiag* en un bloque de columnas *bcol* son los siguientes:

$$\{L_{ij} : pi \le i \le \text{mín}(adiag - tbcol\cdot(bcol-1),\ n),\ j = adiag - i + 1\},$$

donde $pi = \lfloor (adiag + 2) \rfloor / 2$ (si se cumple que (adiag ≤ 2·tbcol·bcol), o en caso contrario, $pi = adiag - tbcol - bcol + 1$.

El número de antidiagonales por bloque de columnas (*eadiag*) donde se consigue la máxima eficiencia en un bloque de columnas respecto de la versión con un sólo procesador será de

$$eadiag = \text{máx}(nadiag - 3\cdot(tbcol-1),\ 0).$$

En consecuencia, los rendimientos serán más elevados cuando mayor sea el ratio *n/tbcol* para un valor de *tbcol* dado.

1									
2	3								
3	4	5							
4	5	6	13						
5	6	7	14	15					
6	7	8	15	16	17				
7	8	9	16	17	18	22			
8	9	10	17	18	19	23	24		
9	10	11	18	19	20	24	25	26	
10	11	12	19	20	21	25	26	27	28

Figura 13. *Secuencia de resolución de los elementos de L para el algoritmo orientado a bloques de columnas PBHWF (n=10, tbcol=3 y p=3).*

Como podemos deducir de la Figura 13, este algoritmo, que llamaremos *PBHWF*, presenta baja eficiencia debido a que existen procesadores ociosos en las antidiagonales iniciales y finales de cada bloque de columnas y en los últimos bloques de la matriz donde *eadiag* = 0. En el siguiente apartado vemos como podemos paliar esta situación.

En este algoritmo podemos optimizar el almacenamiento de las rotaciones asociadas a las transformaciones de Givens que utilizamos para actualizar la matriz *R*. Las rotaciones quedan obsoletas cuando se ha actualizado completamente la matriz *R* al terminar de calcular los elementos de

una columna. Por lo tanto, cuando pasemos al siguiente bloque de columnas es posible utilizar el espacio de memoria en el que almacenamos las rotaciones anteriores para guardar las nuevas rotaciones que calcularemos. Esto supondrá un tamaño de almacenamiento de $2 \cdot n \cdot tbcol$ para senos y cosenos. Debido a un aumento considerable del coste [Hodel 92] y del espacio de almacenamiento, no es aconsejable acumular las rotaciones de Givens como ocurre en otros problemas [Bischof 87].

Algoritmo orientado a bloques de columnas con solapamiento.

En este tercer algoritmo, *PBSHWF,* realizamos un solapamiento entre el final de un bloque de columnas y el principio del siguiente. Con ello conseguimos aumentar la eficiencia en las antidiagonales finales e iniciales de los bloques de columnas, incrementando el valor de *eadiag*.

Cada bloque de columnas tiene ahora $inc = tbcol-1$ antidiagonales menos que en el caso anterior, excepto el primero y, en los casos en que n no sea un múltiplo de *tbcol*, el último bloque. Éstas son justamente las primeras de cada bloque. Las antidiagonales *adiag* de un bloque de columnas *bcol* relacionadas con las generales de L serán:

$$adiag = 2 \cdot tbcol \cdot (bcol-1)+inc+1,..., \text{ mín}(n+tbcol \cdot bcol-1, 2 \cdot n-1),$$

con las restricciones de contorno comentadas anteriormente.

1									
2	3								
3	4	5							
4	5	6	*	(*)					
5	6	7	*	13					
6	7	8	13	14	15				
7	8	9	14	15	16	*	(*)		
8	9	10	15	16	17	*	20		(*)
9	10	11	16	17	18	20	21	22	(*)
10	11	12	17	18	19	21	22	23	24
11	(12)		18	(19)		(22)	(23)		
12			19			(23)			

* elementos que se calculan en el solapamiento.
(*) elementos que en el solapamiento son superiores a la diagonal.

Figura 14. *Secuencia de resolución de los elementos de L para el algoritmo orientado a bloques columna con solapamiento PBSHWF (n= 10, tbcol= 3 y p= 3).*

Los elementos que calculamos para una antidiagonal *adiag* en un bloque de columnas *bcol* son:

$$\{L_{ij} : pi \le i \le adiag-tbcol \cdot (bcol-1)\ j= adiag-i+1\},$$

donde $pi= (adiag+2)/2$ si de cumple que $(adiag \le 2 \cdot tbcol \cdot bcol)$. En caso contrario $pi= adiag-tbcol-bcol+1$, salvo para los valores de $i > n$ donde

$$i= i-n+tbcol \cdot bcol \text{ y } j= j+tbcol,$$

realizándose el solapamiento con el siguiente bloque.

En la Figura 14 se puede comprobar como, mediante el solapamiento, reducimos el número de pasos (ver Figura 13). Incluso se consigue reducir el número de pasos respecto del algoritmo *PBHWF* cuando tenemos las mismas condiciones de limitación del número de procesadores.

5.2.2.2 Algoritmos paralelos de grano medio

El segundo enfoque para la resolución paralela de este problema está basado en el incremento del grano de resolución del problema. Lo que hemos dado en llamar algoritmos de grano medio. En éstos, cada procesador calcula en cada paso de la resolución paralela, no el elemento de una columna de L, sino un vector o conjunto de elementos contiguos pertenecientes a una columna de L.

El primer paso de esta idea es dividir las columnas de la matriz L en vectores v_{ij}, de longitud t. Por sencillez, consideramos que todos los vectores son de igual tamaño. Si n es la dimensión de L, dividiremos la *j-ésima* columna de L en h vectores $(v_{1j},...,v_{hj})$, donde $h= n/t$. Por lo tanto, la matriz L queda dividida de la siguiente forma:

$$L=\{v_{ij} : 1\le i \le h,\ 1\le j \le n\}.$$

Los elementos del vector v_{ij} serán:

$$v_{ij}=(L((i\text{-}1)\ t+1,j),..., L(\mathrm{i}\cdot t,j))^T.$$

Lógicamente, los elementos $L(k,j)$ del vector v_{ij} que se encuentran por encima de la diagonal principal ($k < j$) son nulos.

La secuencia que seguiremos para la resolución de la ecuación será la misma que la definida en la sección anterior, pero ahora con un tamaño de grano mayor. Veremos que aparecen problemas de localidad y eficiencia que trataremos de resolver utilizando técnicas similares. Abordar el problema con un grano mayor nos permite aprovechar, cuando sea posible, el procesamiento vectorial que anteriormente no pudimos explotar de forma intensiva. Como contrapartida, aparecen más problemas en la extracción del paralelismo para las zonas de contorno de la matriz L.

En esta primera aproximación de grano medio realizaremos un barrido por antidiagonales de vectores de la matriz L. Si tenemos en cuenta las consideraciones anteriores, el número de antidiagonales *nvadiag* será de $t+n$-1. Los vectores pertenecientes a una antidiagonal de vectores dada, v*adiag,* serán:

$$\{v_{ij} : pi \le j \le \mathrm{mín}(vadiag,\ n),\ i = vadiag - j+1\},$$

siendo $pi= 1$ si se cumple que *(vadiag* $\le h$*).* En caso contrario $pi= adiag\text{-}h+1$. De esta forma todos los vectores de la antidiagonal están por debajo de la diagonal principal de vectores v_{ii}, $i= 1,..., h$.

Nuestro primer algoritmo, que llamaremos PHWM calculará cada uno de los vectores pertenecientes a una misma antidiagonal de vectores mediante el procedimiento *PGM*.

Procedimiento 2 *PGM*(i',j,t): Calcula el vector *v(i', j)* de dimensión t.
$i= \mathrm{máx}((i'\text{-}1)\cdot t+1, j)$
1. Cálculo del elemento diagonal de la columna j.
si ($i = j$) entonces

$\alpha(j) = \sqrt{1 - S(j,j)^2}$, $L(j,j) = \alpha(j)^{-1} R(j,j)$, $\beta(j) = L(j,j)S(j,j)$

i= *j*+1

fin si

ifin= mín(*it'*,*n*)

2. Cálculo de los elementos subdiagonales de la columna *j*.

para *l*= *i, ifin*

$$L(l,j) = \left(-\alpha(j)\, R(l,j) - \beta(j) S(l,j) - S(j,j) \sum_{k=j+1}^{l-1} S(l,k) L(k,j)\right) / (S(l,l) S(j,j) - 1)$$

fin para

3. Cálculo del vector *y*.

para *l*= *i, ifin*

$$y(l) = \alpha(j) \left(L(j,j) S(l,j) + \sum_{k=j+1}^{l} S(l,k)\, L(k,j)\right) - S(j,j)\, R(l,j)$$

fin para

4. Actualización de la matriz *R* para el siguiente paso.

4.1. Aplicación de las rotaciones anteriores sobre *R* e *y*:

para *l*= *i, ifin*

para *k*= *j*+1, *i*-1

$$\begin{bmatrix} R(l,k) & y(l) \end{bmatrix} = \begin{bmatrix} G(l,k) & y(l) \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_{kj} & \operatorname{sen}\theta_{kj} \\ -\operatorname{sen}\theta_{kj} & \cos\theta_{kj} \end{bmatrix}$$

fin para

fin para

para *l*= *i, ifin*

4.2. Cálculo de la rotación de Givens ($\cos\theta_{lj}$,$\operatorname{sen}\theta_{lj}$) tal que:

$$\begin{bmatrix} R(l,j) & y(l) \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_{lj} & \operatorname{sen}\theta_{lj} \\ -\operatorname{sen}\theta_{lj} & \cos\theta_{lj} \end{bmatrix} = \begin{bmatrix} * & 0 \end{bmatrix}$$

4.3. Aplicación de las rotaciones de Givens al bloque actual de la columna de *R*.

para *k*= *l, ifin*

$$\begin{bmatrix} R(k,l) & y(k) \end{bmatrix} = \begin{bmatrix} R(k,l) & y(k) \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_{lj} & \operatorname{sen}\theta_{lj} \\ -\operatorname{sen}\theta_{lj} & \cos\theta_{lj} \end{bmatrix}$$

fin para

fin para

fin

En el procedimiento *PGM* podemos sacar partido del posible potencial vectorial del procesador. Evidentemente, esto será rentable para matrices de tamaño medio o grande, y si elegimos un tamaño de vector *t* de división de la matriz *L* que nos permita un máximo rendimiento de la unidad vectorial del procesador. Esto sucederá siempre que *t* sea un múltiplo del tamaño de los registros vectoriales asociados a la unidad vectorial y hará que se reduzca el número de accesos a la antememoria. La secuencia temporal de resolución de la matriz *L*, si disponemos de un número de procesadores $p = n/t$, es igual al número de antidiagonales de vectores *nvadiag* (ver Figura 15).

1															
1	2														
1	2	3													
2	3	4	5												
2	3	4	5	6											
2	3	4	5	6	7										
3	4	5	6	7	8	9									
3	4	5	6	7	8	9	10								
3	4	5	6	7	8	9	10	11							
4	5	6	7	8	9	10	11	12	13						
4	5	6	7	8	9	10	11	12	13	14					
4	5	6	7	8	9	10	11	12	13	14	15				
5	6	7	8	9	10	11	12	13	14	15	16	17			
5	6	7	8	9	10	11	12	13	14	15	16	17	18		
5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Figura 15. *Secuencia de resolución de los vectores de L para el algoritmo paralelo general (n=16, $p=\lfloor n/t \rfloor$ y t=3).*

1															
1	2														
1	2	3													
2	3	4	5												
2	3	4	5	7											
2	3	4	5	7	9										
3	4	5	7	9	11	13									
3	4	5	7	9	11	13	15								
3	4	5	7	9	11	13	15	17							
4	5	7	9	11	13	15	17	19	20						
4	5	7	9	11	13	15	17	19	20	21					
4	5	7	9	11	13	15	17	19	20	21	22				
6	8	10	11	13	15	17	19	20	21	22	23	24			
6	8	10	11	13	15	17	19	20	21	22	23	24	25		
6	8	10	11	13	15	17	19	20	21	22	23	24	25	26	
8	10	12	14	16	18	19	20	21	22	23	24	25	26	27	28

Figura 16. *Secuencia de resolución de los vectores de L para el algoritmo paralelo general PHWM (n= 16, p= 3 y t= 3).*

Al igual que en el caso de grano fino, cuando dispongamos de un número menor de procesadores, cada antidiagonal de vectores se calcula en más de un paso (ver Figura 16). Esto supone un aumento de tiempo en la resolución y una pérdida de eficiencia en el cálculo de las antidiagonales de vectores que no sean múltiplos del número de procesadores. Se puede observar como se repiten en esta aproximación los problemas que encontramos en los algoritmos de grano fino.

Algoritmo orientado a bloques de columnas.

Persiguiendo los mismos objetivos que con el algoritmo *PBHWCF*, proponemos un algoritmo orientado a bloques de columnas, *PBHWM*. Al igual antes se debe elegir el tamaño de los bloques de columnas *tbcol*. En dicha elección intervendrán ahora dos factores: el tamaño de los registros asociados a la unidad vectorial (*tuv*) y el número de procesadores *p*. La conclusión a la que fácilmente se llega, y la más sencilla, es que si tomamos *t* y *tbcol* iguales y múltiplos de *tuv* y *p* (ver Figura 17), podremos esperar el máximo de las prestaciones posibles. Al igual que en el caso de grano fino estas conclusiones estarán condicionadas por el tráfico de datos ocasionado en la antememoria.

El número de bloques columna *nbcol* tendrá la misma expresión que en el caso de grano fino y el número de antidiagonales de un bloque dado *bcol* será,

$$nadiag=t+nbcol-bcol,$$

salvo en el último bloque donde tendremos que $nadiag=n-tbcol\cdot nbcol$ cuando n no sea un múltiplo de t.

Los vectores pertenecientes a una misma antidiagonal de vectores $vadiag\in[1,nadiag]$ del bloque *bcol* serán:

$$\{v_{ij} : pi \le j \le \min(t(bcol-1)+vadiag,\ t\cdot bcol),\ i= vadiag - j+1\},$$

siendo $pi= t\cdot(bcol-1)$ si $(vadiag \le h-bcol+1)$. En caso contrario $pi= t\cdot(bcol-1)+vadiag - h+bcol+1$.

1															
1	2														
1	2	3													
2	3	4	9												
2	3	4	9	10											
2	3	4	9	10	11										
3	4	5	10	11	12	16									
3	4	5	10	11	12	16	17								
3	4	5	10	11	12	16	17	18							
4	5	6	11	12	13	17	18	19	22						
4	5	6	11	12	13	17	18	19	22	23					
4	5	6	11	12	13	17	18	19	22	23	24				
5	6	7	12	13	14	18	19	20	23	24	25	27			
5	6	7	12	13	14	18	19	20	23	24	25	27	28		
5	6	7	12	13	14	18	19	20	23	24	25	27	28	29	
6	7	8	13	14	15	19	20	21	24	25	26	28	29	30	31

Figura 17. *Secuencia de resolución de los vectores de L para el algoritmo paralelo orientado a bloques columna PBHWM (n=10, p=3, tuv=3 y t=3).*

Algoritmo orientado a bloques columna con solapamiento.

Aplicamos en este tercer algoritmo, *PBSHWM*, la técnica de solapamiento entre bloques columna que ya vimos en el apartado anterior para grano fino. En este caso, el solapamiento no es de elementos sino de vectores, como es evidente. Con ello paliamos la pérdida de eficiencia al principio y al final de la resolución de cada columna, provocada por la división en bloques columna del algoritmo anterior. De este modo eliminamos en lo posible la existencia de procesadores ociosos al principio y al final de cada bloque.

Manteniendo el mismo número de antidiagonales que en el algoritmo anterior, ahora los vectores a calcular de una antidiagonal dada *vadiag* del bloque columna *bcol* serán:

$$\{v_{ij} : pi \le j \le \min(t(bcol-1)+vadiag,\ t\cdot bcol),\ i= vadiag - j+1\},$$

siendo $pi= t\cdot(bcol-1)$ salvo en los dos últimos bloques columna donde no habrá nunca solapamiento y en los que $pi = t\cdot(bcol-1)$ si $(vadiag \le h-bcol+1)$, o en caso contrario $pi =t\cdot(bcol-1)+vadiag-h+bcol+1$.

Los vectores v_{ij} de las antidiagonales para los que $i > h$ son asignados convenientemente al siguiente bloque columna según la siguiente relación:

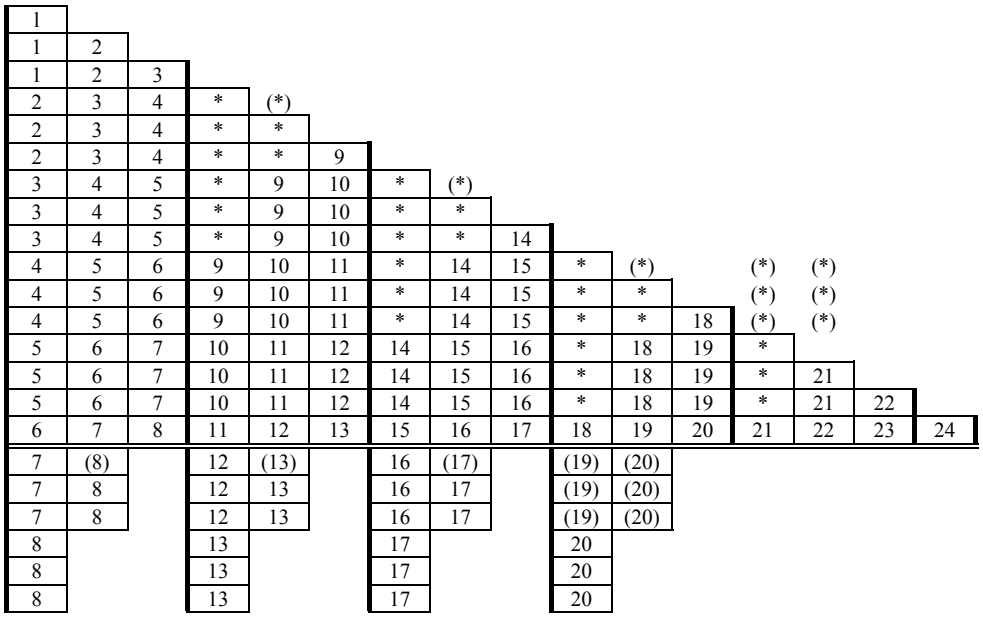
$$i' = i\text{-}h+bcol \text{ y } j' = j+tbcol.$$

El cálculo del vector asociado $v_{i'j'}$ se realiza siempre y cuando la antidiagonal de vectores asociada sea menor o igual que la asociada a v_{ij} , o lo que es lo mismo,

$$i+j \geq i'+j'$$

y además se cumpla que $i' \geq j'$.

En la Figura 18 podemos observar el solapamiento para un caso concreto y se evidencia el aumento de la eficiencia respecto del algoritmo anterior.



* elementos que se calculan en el solapamiento.
(*) elementos que no se calculan en el solapamiento por ser superiores a la diagonal.

Figura 18. *Secuencia de resolución de los vectores de L para el algoritmo paralelo orientado a bloques columna con solapamiento PBSHWM (n=10, p=3 y t=3).*

Si el orden del problema *n* no es múltiplo de *t*, los algoritmos de grano medio pierden eficiencia cuando se calculan los últimos vectores de cada columna. También se perderá parte de la eficiencia cuando se calculen los últimos bloques de columnas de la ecuación debido a que se reduce el paralelismo. Cuando mayor es el valor de *t*, mayor será la pérdida de eficiencia. Para incrementar las prestaciones del algoritmo en el cálculo de los últimos bloques columna de la ecuación, podemos adaptar el tamaño del vector *t* que resolvemos en cada paso del algoritmo [Hodel 92]. También podemos realizar otra aproximación consistente en combinar los algoritmos de grano fino y medio, utilizando uno y otro dependiendo de la eficiencia que presenten en cada etapa de la resolución del problema. A este algoritmo lo denominaremos *PBSHWC*.

Análisis teórico de los algoritmos.

En este apartado presentamos el análisis teórico de los algoritmos para mulriprocesadores con memoria compartida. Para ello hemos utilizado el programa *Mathematica* [Wolfram 88]. Consideramos como en los apartados anteriores que la matriz de estados o matriz de coeficientes, A, de la ecuación de Lyapunov en tiempo discreto sólo posee valores propios reales y que por lo tanto S, su forma real de Schur, es triangular inferior.

El algoritmo de Hammarling secuencial *SH* tiene, como vimos en la sección 3, tres pasos para la resolución de cada columna. La contribución de cada uno de ellos al coste global del algoritmo (normalizado en flops) es la siguiente:

$$C_{s1} = 5n,$$

$$C_{s2} = \frac{n^3}{3} + \frac{5n^2}{2} - \frac{17n}{2}$$

$$C_{s3} = \frac{4n^3}{3} + \frac{11n^2}{2} - \frac{25n}{2}$$

El coste global del algoritmo serie es por lo tanto $C_{SH} = \frac{5n^3}{3} + 8n^2 + O(n)$. Si se dispone de un procesador vectorial, el coste teórico será ahora de $\frac{5\beta n^3}{3} + \frac{(14+5\alpha)n^2}{2} + O(n)$, donde α es coste de inicio o llenado de la unidad vectorial y β el tiempo dedicado a cada operación, obviamente con $\beta \leq 1$. Este coste se obtiene del análisis de los bucles y la dependencia de datos en cada etapa.

El coste teórico para una paralelización directa de este algoritmo en un multiprocesador con memoria compartida con p procesadores escalares es de $\frac{5n^3}{3p} + O(n^2)$, el mismo que para el algoritmo *PBSHWF* utilizando p procesadores. El coste para la misma paralelización directa del algoritmo *SH*, si los procesadores son vectoriales es de $\frac{5\beta n^3}{3} + O(n^2)$.

Si comparamos el coste obtenido para el algoritmo *PBSHWM* con la paralelización directa del algoritmo *SH,* tanto escalar como vectorial, vemos que tienen las mismas expresiones.

Existen parámetros que no pueden ser tenidos en cuenta en este estudio del coste y que dependen del computador. El aprovechamiento de estos parámetros como son el nivel de sincronización en la paralelización, la eficiencia en la utilización de los procesadores y las unidades vectoriales, y la optimización de los accesos a memoria en computadores con memoria jerarquizada, etc., debe analizarse para cada implementación concreta.

5.2.3 Multiprocesadores con memoria distribuida

Recientemente se han desarrollado algoritmos paralelos en multiprocesadores con memoria distribuida para resolver sistemas triangulares lineales. Estos están basados en tres esquemas básicos: algoritmos de difusión/recogida [Romine 86], frente de onda (*wavefront*) [Heath 88] y cíclicos [Chamberlain 86, Li 87, Li 88]. Consecuencia de éstos, se han desarrollado algoritmos de difusión/recogida [Kågström 89, Kågström 90] y algoritmos cíclicos [Marqués 91, Marqués 92

Marqués 93, Quintana 93], basados en el método de Schur o en el de Hessenberg-Schur para resolver las ecuaciones de Sylvester y Lyapunov en multiprocesadores con memoria distribuida. Nuestro trabajo está centrado en los algoritmos de frente de onda y cíclicos, que han demostrado ser muy eficaces en la resolución de sistemas triangulares lineales.

5.2.3.1 Algoritmos de frente de onda

A partir del estudio de la dependencia de datos realizado en la apartado 5.2.1, y procediendo de forma similar a como se ha hecho para memoria compartida [Claver 96], nuestro algoritmo recorre las *adiag* antidiagonales de L y, utilizando el procedimiento *PGF* descrito en el apartado 5.2.2.1, calcula en paralelo los elementos $L(i,j)$ pertenecientes a la misma antidiagonal en cada paso.

Este algoritmo obtiene prestaciones óptimas cuando el número de procesadores es $p = n/2$. Sin embargo, en la práctica p es mucho menor y cada antidiagonal deberá resolverse en más de un paso. Por simplicidad se asume que n es múltiplo de p. Teniendo en cuenta que el método de Hammarling está orientado a columnas, y para reducir los problemas en el acceso a memoria, proponemos una resolución de la ecuación por bloques de columnas. Así, si $c = n/p$, cada procesador P_i, $i = 0,\ldots, p\text{-}1$, resuelve secuencialmente las columnas $L(:,i)$, $L(:,i+p),\ldots, L(:,i+(c\text{-}1)p)$. Al finalizar la resolución del problema, la matriz solución L se encontrará distribuida cíclicamente por columnas entre los procesadores.

Esta distribución es especialmente indicada para un vector lineal de procesadores, resolviendo en cada paso una antidiagonal de elementos dentro de un bloque de columnas. Así, dado un bloque de columnas $h = 0,\ldots, c\text{-}1$, los elementos de L pertenecientes a la antidiagonal $i = hp,\ldots, n+p\text{-}1$ de éste son

$$L(i, hp), L(i\text{-}1, hp+1),\ldots, L(i\text{-}p+1, h(p+1)\text{-}1).$$

No se consideran los elementos sobre la diagonal principal ni aquellos que sobrepasen las dimensiones de la matriz solución L.

Como hemos visto en el procedimiento *PGF*, para el cálculo del elemento $L(i,j)$ necesitamos acceder a la fila $S(i,:)$, la fila $R(i,:)$ actualizada, los elementos diagonales $S(j,j),\ldots,S(i\text{-}1,i\text{-}1)$, los elementos $L(j:i\text{-}1,j)$ y las rotaciones $(sin\theta_{j+1,j},\cos\theta_{j+1,j}),\ldots,(sin\theta_{i-1,j},\cos\theta_{i-1,j})$. Por lo tanto, como cada procesador resuelve las columnas que le corresponden de forma completa, el procesador que calcule el elemento $L(i,j)$, debe poseer los elementos anteriormente calculados de la *j-ésima* columna, así como sus rotaciones asociadas. Además, necesitará las filas *i-ésimas* de S y R, así como los elementos diagonales de S.

Proponemos pues una distribución por bloques de filas de las matrices S y R entre todos los procesadores similar a la propuesta por Heath y Romine [Heath 88]. En la Figura 19 se muestra esta distribución de datos para p=5.

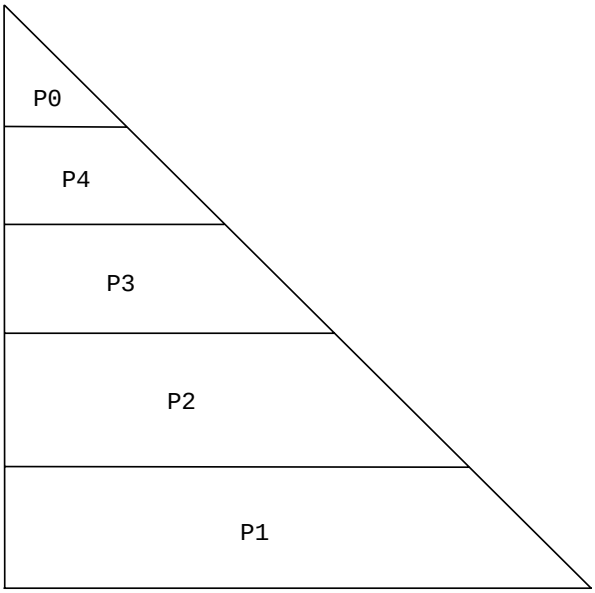


Figura 19. *Distribución de las matrices S y R para los algoritmos de frente de onda (p=5).*

Alternativamente, para problemas de tamaño reducido, la matriz *S* puede estar replicada en todos los procesadores al no ser modificada en todo el proceso de resolución.

En nuestro caso la topología más idónea será la de un anillo unidireccional, de forma que el procesador $P_{j \text{ res } p}$ calcula el elemento $L(i,j)$, actualiza los elementos de $R(i,j+1{:}i)$ y los envía junto con los elementos de $S(i,j+1{:}i)$ al procesador $P_{(j+1) \text{ res } p}$. En la Figura 20 se muestra la secuencia de resolución para n=9 y p=3, donde el número de procesador y el paso en el que se realiza el cálculo se muestra para cada elemento $L(i,j)$.

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

Figura 20. *Secuencia de resolución por frente de onda de antidiagonales para n=9 y p=3 (Procesador, Paso).*

Como vemos en la Figura 20, la solución *L* queda distribuida cíclicamente por columnas. Nuestro algoritmo, que denominamos *PDHWF*, utiliza una cola (buffer) de parejas de filas de *S* y *R*. Esta cola se utiliza para almacenar la distribución de estas dos matrices, y para acondicionar dicha

distribución de datos durante el proceso de resolución del problema. Inicialmente, cada cola almacena las filas correspondientes a *S* y *R* de acuerdo con la distribución por bloques de filas de la Figura 19. La información almacenada en las colas se reduce progresivamente a medida que se resuelve el problema en una fila de cada matriz en cada iteración. Para este propósito, se definen dos procedimientos: **envía_cabeza** y **añade_cola**. La función **envía_cabeza**(*Buffer*) toma las filas de *S* y *R* que hay en la cabeza de la cola de un procesador (p.e., P_j) y las envía al siguiente procesador del anillo (p. e., $P_{(j+1) \text{ res } p}$), eliminando posteriormente estas dos filas de la cabeza de la cola. La función **añade_cola**(*Buffer, S*(*i*,:)*, R*(*i*,:)) añade al final de la cola las filas *S*(*i*,:) y *R*(*i*,:). El algoritmo funciona como una unidad *segmentada* que es realimentada constantemente. En el algoritmo *PDHWF* no se han considerado las condiciones especiales de los primeras y últimas etapas de la resolución del problema.

Algoritmo 11 *[PDHWF]*

Memoria: *Buffer*((*S*(*n/p,n*), *R*(*n/p,n*)), *L*(*n,n/p*)

Funciones: **envía_cabeza**(*Buffer*), **añade_cola**(*Buffer, filaS, filaR*)

```
para j= 0, n-1
    si (j ∈ micol) entonces
        para i= j, n-1
            si (i = j) entonces
                recibir( P(j+p-1) res p , S(j, j), R(j, j))
                envía_cabeza(Buffer)
                PGF(j,j)
            sino
                recibir( P(j+p-1) res p , S(i,j:i), R(i,j:i))
                PGF(i,j);
                añade_cola(Buffer, S(i,j+1:i),R(i,j+1:i))
                envía_cabeza(Buffer)
            fin si
        fin para
    fin si
fin para
fin
```

Incrementando el grano de resolución

Para incrementar en lo posible la relación entre el tiempo dedicado al cálculo aritmético frente al de comunicación de los datos, aprovechando al máximo la presencia de los datos en el procesador, podemos aumentar el grano de resolución de nuestro problema. En concreto, el algoritmo puede ser orientado a la resolución en cada paso de vectores de cierto tamaño *t* o de bloques de tamaño $n_b \times m_b$.

La contrapartida a esta opción será la reducción del paralelismo del problema, sobre todo en las últimas etapas. La forma en que esto afecta al coste temporal de la resolución depende del número de procesadores, del tamaño del problema, de las características de potencia de computación de cada nodo y del ancho de banda de las comunicaciones.

Para resolver el problema utilizando una aproximación de grado medio, dividimos las columnas de L en vectores de longitud t. Por simplicidad consideramos que la dimensión de la matriz L, n, es múltiplo de t. Así, la columna *j-ésima* de L se divide en $f=n/t$ vectores, $L(:,j) = (v_{0j},\ldots,v_{f-1,j})^T$, donde $v_{ij} = (L_{it,j},\ldots,L_{(i+1)t-1,j})$ y los elementos L_{kj}, $k < j$, son cero. El procedimiento *PGM,* que vimos en el apartado 5.2.2, resuelve el vector v_{ij} de tamaño t perteneciente a la columna j.

La distribución de las matrices de datos S y R es la misma que la propuesta anteriormente, por bloques de filas. La topología utilizada será también un anillo unidireccional, de forma que el procesador $P_{j \text{ res } p}$ calculará el vector v_{ij}, actualizará las filas $R(it:(i+1)t-1, j+1:(i+1)t-1)$ y las enviará junto con las filas $S(it:(i+1)t-1, j+1:(i+1)t-1)$ al procesador $P_{((j+1) \text{ res } p}$. El algoritmo *PDHWM*, muy parecido como era de esperar al algoritmo *PDHWF*, aplica el procedimiento descrito. Las funciones sobre la cola (*Buffer*) ahora afectarán a t filas de S y R cada vez. El manejo de la cola se hace de forma que se optimizan los datos almacenados y enviados. En la Figura 21 se muestra la secuencia de resolución del algoritmo *PDHWM* para $n= 9$, $p=3$ y $t=2$.

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

Figura 21. *Secuencia de resolución por vectores con n=9, p=3 y t=2(Procesador, Paso).*

En el algoritmo *PDHWM*, que se muestra a continuación, también se pueden tener en cuenta las consideraciones hechas para multiprocesadores con memoria compartida por Hodel y Polla [Hodel 92] y por Claver *et al.* [Claver 96] respecto a la variación adaptativa del valor de t y el cambio de resolución de vectores a la resolución de elementos, respectivamente. En ambos casos, el cambio de grano en la resolución se hace atendiendo a los resultados obtenidos experimentalmente en cada computador.

Algoritmo 12 *[PDHWM]*

Memoria: *Buffer*((*S*(*n/p*,*n*),*R*(*n/p*, *n*))), *L*(*n, n/p*)

Funciones: **envía_cabeza**(*Buffer, t*), **añade_cola**(*Buffer, bloque_filaS, bloque_filaR*)

para $j= 0, n-1$

```
si (j ∈ micol) entonces
    para i= 0, n/t-1
        si ((i+1)t-1 = j) entonces
            si (vector ∈ diagonal) entonces
                recibir( P_{(j+p-1) res p} , (S, R)(j:(i+1)t-1, j:(i+1)t-1))
                envía_cabeza(Buffer, t)
                PGM(i,j,t)
                añade_cola(Buffer, (S, R)(j+1:(i+1)t-1, j+1:(i+1)t-1)
            sino
                recibir( P_{(j+p-1) res p} , (S,R)(it:(i+1)t-1, j:(i+1)t-1))
                PGM(i,j,t)
                añade_cola(Buffer, (S,R)(it:(i+1)t-1, j+1:(i+1)t-1))
                envía_cabeza(Buffer, t)
            fin si
        fin si
    fin si
  fin para
fin para
fin
```

Para resolver la ecuación de Lyapunov utilizando una aproximación de grano grueso dividimos la matriz L en bloques de tamaño $n_b \times m_b$. Para simplificar, asumimos que la dimensión de L, n, es múltiplo de n_b y m_b. Así, la matriz L la consideraremos dividida en $n/n_b \times m/m_b$ bloques L_{ij}, donde el bloque $L_{ij} = L(in_b : (i+1)n_b\text{-}1,\ jm_b : (j+1)m_b\text{-}1)$ (los elementos L_{kj}, $k < j$, son cero). A continuación presentamos el procedimiento $PGB(i,j,n_b,m_b)$ que calcula el bloque L_{ij}.

Procedimiento 3 *[PGB(i,j,nb,mb)]*: Calcula el bloque L_{ij}

```
para j'= jm_b, (j+1)m_b-1
    para i'= in_b, (i+1)n_b-1
        PGF(i', j')
    fin para
fin para
fin
```

El procedimiento *PGB* podría estar orientado a la resolución de vectores (fila o columna) [Claver 97]. Esta aproximación supone mayor nivel de localidad de los accesos a memoria y, por lo tanto, unas mejores prestaciones.

La distribución de los datos y la topología utilizada es la misma que para los casos de grano fino y medio. En el algoritmo de grano grueso u orientado a bloques, *PDHWB*, el bloque L_{ij} es

calculado por el procesador $P_{j \text{ res } p}$, que actualizará el bloque $R(in_b{:}(i{+}1)n_b{-}1,\ (j{+}1)m_b{:}(i{+}1)m_b{-}1)$ y lo enviará junto con el bloque $S(in_b{:}(i{+}1)n_b{-}1,\ (j{+}1)m_b{:}(i{+}1)m_b{-}1)$ al procesador $P_{(j+1) \text{ res } p}$. La Figura 9 muestra la secuencia de resolución de L mediante el algoritmo *PDHWB* para n=9, p=2, n_b=2 y m_b=2.

0,1								
0,1	0,1							
0,2	0,2	1,3						
0,2	0,2	1,3	1,3					
0,3	0,3	1,4	1,4	0,6				
0,3	0,3	1,4	1,4	0,6	0,6			
0,4	0,4	1,5	1,5	0,7	0,7	1,8		
0,4	0,4	1,5	1,5	0,7	0,7	1,8	1,8	
0,5	0,5	1,6	1,6	0,8	0,8	1,9	1,9	0,10

Figura 22. *Secuencia de resolución de L para el algoritmo PDHWB con n=9, p=2, n_b=2, m_b=2 (Procesador, Paso).*

El algoritmo *PDHWB* funciona de forma parecida al algoritmo *PDHWM*. Cada procesador calcula completamente un bloque columna en cada iteración antes de pasar al siguiente. Por ejemplo, el procesador P_i calcula secuencialmente los bloques columna

$$L(:,(i+cp)m_b{:}(i+cp+1)m_b{-}1), \text{ donde } c = 0,\ 1\ ,...,\ (n/pn_b)\ {-}1.$$

Los procedimientos **añade_cola** y **envia_buffer** actúan simultáneamente en este caso con n_b filas cada vez.

Algoritmo 13 *[PDHWB]*

Memoria: *Buffer*((*S*(*n/p*,*n*),*R*(*n/p*, *n*))), *L*(*n, n/p*)

Funciones: **envía_cabeza**(*Buffer, n_b*), **añade_cola**(*Buffer, bloque_filaS, bloque_filaR*)

para j= 0, n/m_b-1

si ($j \in$ *mibloquecol*) entonces

para i= 0, n/n_b-1

si ((i+1)n_b-1 = jm_b) entonces

si (*bloque* $\in$ *bloquediagonal*) entonces

recibir($P_{(j+p-1) \text{ res } p}$, $(S, R)(in_b{:}(i{+}1)n_b{-}1, jm_b{:}(i{+}1)n_b{-}1)$)

envía_cabeza(*Buffer, n_b*)

PGB(i,j,n_b,m_b)

añade_cola(*Buffer*, (*S, R*)(mín((*i*+1)n_b,(*j*+1)m_b):(*i*+1)n_b-1,
mín((*i*+1)n_b,(*j*+1)m_b): (*i*+1)n_b-1))
sino
recibir($P_{(j+p-1)\,\mathrm{res}\,p}$, (*S, R*)(*in*$_b$:(*i*+1)n_b-1, (j+1)m$_b$:(*i*+1)n_b-1))
PGB(*i,j,*n_b,m_b)
añade_cola(*Buffer*, (*S, R*)(*in*$_b$:(*i*+1)n_b-1, *jm*$_b$:(*i*+1)n_b-1))
envía_cabeza(*Buffer,* n_b)
fin si
fin si
fin para
fin si
fin para
fin

Análisis teórico de los algoritmos.

A continuación presentamos el análisis teórico de los algoritmos frente de onda. Hemos utilizado el programa *Mathematica* [Wolfram 88] para realizar dicha evaluación. Consideramos que la matriz de estados o matriz de coeficientes, *A*, de la ecuación de Lyapunov en tiempo discreto sólo posee valores propios reales y que por lo tanto, *S*, su forma real de Schur, es triangular inferior tal y como hemos venido haciendo a lo largo de estos apartados.

En el análisis de los algoritmos consideramos que el coste de comunicación entre dos procesadores conectados directamente está modelado por la expresión lineal

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

donde σ es el coste de inicio del mensaje, independientemente de su longitud, τ es el coste incremental por unidad de longitud y *M* es la longitud del mensaje en palabras (en nuestro caso una palabra equivale una unidad de almacenamiento de un número en coma flotante). Por simplicidad consideramos los valores de σ y τ normalizados respecto a su relación con el coste de cálculo, y por lo tanto los trataremos en flops.

En el coste de las comunicaciones sólo se tiene en cuenta la recepción y no el envío, que para nuestro estudio lo consideraremos inmediato. Se trata pues de un modelo asíncrono simplificado, pero a pesar de ello será suficiente para nuestros objetivos.

Antes de comenzar el estudio de los algoritmos, veamos algunas definiciones preliminares. Denominamos $C_A(i,j)$ al coste aritmético para calcular completamente el elemento $L(i,j)$, $C_C(i,j)$ será el coste de comunicaciones asociado a éste y $C_E(i,j)$ el coste debido a la espera producida hasta recibir los datos necesarios para el cálculo del elemento.

El coste del algoritmo *PDHWF* para el elemento $L(i,j)$ lo denominaremos $C_{PDHWF}(i,j)$ y vendrá definido por la siguiente expresión

$$C_{PDHWF}(i,j) = C_A(i,j)+C_C(i,j)+C_E(i,j),$$

donde, $C_A(i,j)= 10i-10j+24$, $C_C(i,j)$ es el coste de comunicación para transmitir las filas $S(i,j:i)$ y $R(j,j:i)$, y viene dado por la expresión

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

y $C_E(i,j)$ es el coste de tiempo de espera entre el cálculo del elemento $L(i,j)$ y el $L(i+1, j-1)$, y está definido por la ecuación

$$C_E(i,j)= \text{máx}\{0, C_A(i,j) - C_C(i+1,j-1)\},$$

que, como puede verse en la expresión $C_A(i,j) - C_C(i+1,j-1)$, será teóricamente siempre negativa, salvo al principio y al final de la resolución de cada columna de L. Por lo tanto podemos despreciar las esperas en el cálculo del coste para este algoritmo.

Puesto que el procesador P_0 es el que tiene un mayor coste global, podemos definir un límite superior para el coste total del algoritmo paralelo definido por la expresión

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

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

Si analizamos la expresión del coste global, y teniendo en cuenta sólo los términos de orden cúbico, podemos ver como la eficiencia del algoritmo tiene como límite superior

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

expresión que depende del parámetro τ de las comunicaciones.

Consideremos ahora el algoritmo *PDHWM*, donde se calcula L en vectores v_{ij} de dimensión t. El coste del algoritmo *PDHWM,* $C_{PDHWM}(i,j)$, es

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

donde $C_A(k,j)$=10k-10j+24, $C_C(i:i+t-1,j)$ es el coste de comunicación para transmitir las submatrices $S(i:i+t-1,j:i+t-1)$ y $R(i:i+t-1,j:i+t-1)$, y viene definido por la expresión

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

y $C_E(i:i+t-1,j)$ es el coste de la espera debida a la diferencia de coste entre el cálculo del vector $L(i:i+t-1,j)$ y el $L(i+t:i+2t-1,j-1)$, y la podemos expresar como

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

Al igual que para *PDHWF* podremos despreciar las esperas también para este caso. El límite superior del coste total de este algoritmo viene definido por la expresión

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

donde $j=(l-1)p+1$ e $i=(q-1)t+1$.

Al aumentar el grano de resolución vemos que si consideramos únicamente aquellos términos de orden cúbico, el coste es el mismo que para *PDHWF*. Sólo se detecta una reducción del coste en el orden cuadrático debido al término que depende de la iniciación de cada mensaje, $\sigma/p\,\tau$. Este menor coste en la comunicación, junto con un mayor grano de computación puede conseguir un incremento de las prestaciones del algoritmo debido a la mayor localidad y reutilización de los datos. Este efecto se verá acentuado cuando incrementemos el tamaño del vector. Este estudio y sus efectos pueden extenderse para el caso en que el algoritmo, en vez de resolver un vector en cada paso, resuelva un bloque de la matriz solución *L*. Como contrapartida, sabemos que el incremento del grano de resolución reduce el paralelismo inherente de un problema, por lo que la elección del grano de resolución siempre supone un compromiso que depende fuertemente de las características del computador.

5.2.3.2 Algoritmos de frente de onda adaptativos

Con el objetivo de incrementar las prestaciones en los algoritmos de frente de onda orientados por bloques presentamos en este apartado unos algoritmos que adaptan el valor del grano computacional durante la resolución del problema. Esta aproximación fue introducida para resolver el mismo problema por Hodel y Polla [Hodel 92] y por Claver *et al.* [Claver 94, Claver 96] para sistemas multiprocesadores con memoria compartida.

El algoritmo propuesto, *PADHWB*, adapta su bloque de resolución $n_b \times m_b$ cuando se va a resolver el siguiente bloque de columnas del problema. El nuevo tamaño de bloque será, si se requiere el cambio de tamaño, de $(n_b/r)\times(m_b/r)$, donde *r* es un entero positivo. Hemos llamado a *r* el *factor de reducción* de los bloques de resolución.

Para decidir la reducción del tamaño del bloque que utilizamos para resolver el problema, se ha definido una función booleana denominada **cond**. La función **cond** tiene como parámetros el índice del bloque columna que se está resolviendo en ese momento y el número de procesadores, *p*. La función **cond** decide si el algoritmo debe reducir el bloque de resolución en función del número de los pasos que necesitan los procesadores para resolver un bloque columna. La adaptación del tamaño del bloque se detiene cuando se alcanza un tamaño de bloque mínimo $n_b^{min} \times m_b^{min}$. Los valores de *condbloque* y el tamaño de bloque de resolución mínimo dependen de las características del computador y deben determinarse de forma experimental.

Función 1 cond(*j*,*p*)

si ($n_b > n_b^{min}$) entonces
  si ($\frac{n-jn_b}{n_b} <$ *condbloque*) entonces
    devuelve *verdadero*
  sino
    devuelve *falso*
  fin si
sino

devuelve *falso*
fin si
fin

Algoritmo 14 *[PADHWB]*

Memoria: *Buffer*((*S*(*n/p*,*n*),*R*(*n/p*, *n*))), *L*(*n*, *n/p*)

Funciones: **envía_cabeza**(*Buffer,* n_b), **añade_cola**(*Buffer, bloque_filaS, bloque_filaR*)

Constantes: n_b^s, m_b^s, n_b^{min}, m_b^{min}

$n_b = n_b^s$, $m_b = m_b^s$

$nf_b = n / n_b^s$, $mf_b = n / m_b^s$

para *j*= 0, mf_b -1
si (**cond**(*j*,*p*)) entonces
$m_b = m_b / r$, $mf_b = rmf_b$
$n_b = n_b / r$, $nf_b = rnf_b$
j=rj
fin si
si (*j* ∈ *mibloquecol*) entonces
para *i*= *0,* nf_b-1
si ((*i*+1)n_b-1 ≥ *j*m_b) entonces
si (*bloque* ∈ *bloquediagonal*) entonces
recibir($P_{(j+p-1) \bmod p}$, (*S, R*)(*i*n_b:(*i*+1)n_b-1, *j*m_b:(*i*+1)n_b-1))
si ((*mi_id* = *p* -1) & (**cond**(*j*+1,*p*))) entonces
para *k*= 0, *r*-1
envía_cabeza(*Buffer,* n_b/r)
fin para
sino
envía_cabeza(*Buffer,* n_b)
fin si
PGB(*i*,*j*,n_b,m_b)
añade_cola(*Buffer*, (*S, R*)(mín((*i*+1)n_b,(*j*+1)m_b):(*i*+1)n_b-1,
mín((*i*+1)n_b,(*j*+1)m_b): (*i*+1)n_b-1))
sino
recibir($P_{(j+p-1) \bmod p}$, (*S, R*)(*i*n_b:(*i*+1)n_b-1, (j+1)m_b:(*i*+1)n_b-1))
PGB(*i*,*j*,n_b,m_b)
añade_cola(*Buffer*, (*S, R*)(*i*n_b:(*i*+1)n_b-1, *j*m_b:(*i*+1)n_b-1))

```
                    si ((mi_id = p -1) & (cond(j+1,p))) entonces
                        para k= 0, r-1
                            envía_cabeza(Buffer, n_b/r)
                        fin para
                    sino
                            envía_cabeza(Buffer, n_b)
                    fin si
                fin si
            fin si
        fin para
    fin si
fin para
fin
```

5.2.3.3 Algoritmos cíclicos

Los algoritmos cíclicos que describimos en este apartado están basados en los trabajos de Chamberlain [Chamberlain 86], Heath y Romine [Heath 88] y Li y Coleman [Li 88], entre otros, para resolver sistemas lineales triangulares en multiprocesadores con memoria distribuida y posteriormente extendidos y modificados por Li y Coleman [Li 89] y por Eisenstat *et al.* [Eisenstat 88]. La topología utilizada en estos trabajos es la del anillo unidireccional y el hipercubo. En nuestro caso, consideraremos inicialmente el estudio de una topología de anillo de procesadores, aunque, como veremos en el siguiente apartado, podemos considerar la topología de anillo inscrita en otra topología de grado de conexión mayor (hipercubo, malla, toro) y aprovechar ésta para la reducir el tiempo dedicado a las comunicaciones.

En estos algoritmos las matrices del problema son divididas y distribuidas cíclicamente entre los procesadores por filas (columnas) (ver Figura 23). Esta distribución posee buenas propiedades de equilibrio de la carga en los procesos de factorización de matrices que generalmente preceden a la resolución de un sistema triangular. Para simplificar la descripción de los algoritmos definimos una función denominada **map**(*j*), que nos indicará el procesador al que pertenece la *j-ésima* fila (columna) de los datos distribuidos.

En los algoritmos cíclicos, por el anillo de procesadores circula un segmento (Buffer) de tamaño p-1. Este segmento contiene información necesaria para calcular en cada momento un elemento de la matriz solución; después el segmento es actualizado y enviado al siguiente procesador. Esta comunicación intenta solaparse con la actualización de ciertas variables intermedias necesarias para el cálculo de un nuevo elemento cuando el segmento vuelva a circular por el mismo procesador.

Otro tipo de topologías de interconexión, como la de bus, puede ser muy adecuada para este tipo de algoritmos debido a que, al no existir más que un mensaje transmitiéndose en cada momento, no existe ningún cuello de botella debido a la red de interconexión.

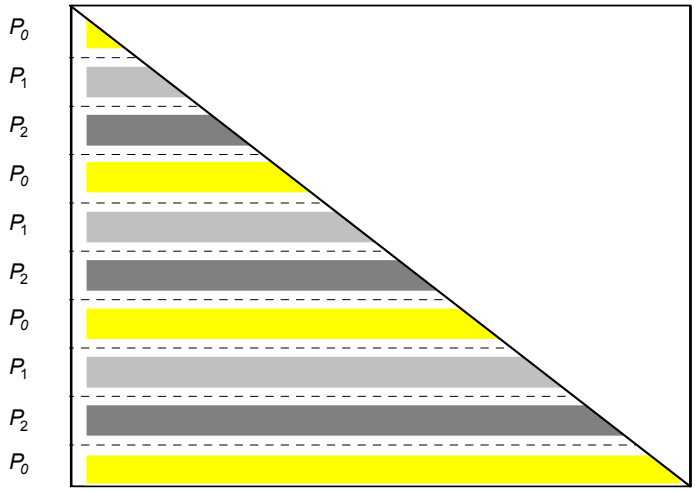


Figura 23. *Distribución cíclica por filas de las matrices de datos para n=10 y p=3*.

Utilizando esta filosofía, la resolución del problema se lleva a cabo secuencialmente por columnas, es decir, todos los procesadores colaboran para resolver una columna cada vez. La resolución de cada columna la dividimos en dos partes: En la primera, que llamamos *resolución*, se calcula completamente la columna solución y consiste básicamente en la resolución de un sistema triangular. En la segunda, que denominamos *triangulación*, se actualiza la matriz de términos independientes; en definitiva se calcula una factorización LQ que anula el vector **y**. Como podemos observar en la dependencia de datos del problema (Figura 9), la división en dos fases de la resolución de una columna es posible al no necesitar la actualización de la matriz de términos independientes durante el cálculo de la columna solución.

0,1								
1,2	1,9							
2,3	2,10	2,17						
0,4	0,11	0,18	0,22					
1,5	1,12	1,19	1,23	1,27				
2,6	2,13	2,20	2,24	2,28	2,32			
0,7	0,14	0,21	0,25	0,29	0,33	0,34		
1,8	1,15	1,22	1,26	1,30	1,34	1,35	1,36	
2,9	2,16	2,23	2,27	2,31	2,35	2,36	2,37	2,38

Figura 24. *Secuencia de resolución para n=9 y p=3 (Procesador, Paso).*

La secuencia de resolución del problema será la mostrada en la Figura 24. Como era de esperar ésta parece secuencial, pero ya hemos comentado que la obtención de rendimiento en este método está en el solapamiento entre la circulación del segmento y las actualizaciones que preparan el cálculo de los siguientes elementos. La matriz solución quedará almacenada, al igual que los datos, cíclicamente por filas.

A continuación se muestra el algoritmo cíclico propuesto (*PHCF*). En la fase de resolución de cada columna el segmento que circula entre los procesadores está constituido por p-1 elementos de L ya calculados, y durante esta circulación del segmento se actualiza el vector correspondiente de términos independientes. En la fase de triangulación el segmento contiene 2(p-1) elementos,

correspondientes a los (p-1) senos y (p-1) cosenos ya calculados y la circulación de este segmento debe solaparse con la actualización de la matriz independiente y de los elementos del vector **y**.

Algoritmo 15 *[PHCF]*

Memoria: $S(n/p,n)$, $R(n/p, n)$, $L(n/p, n)$

$seg(n)$, $Cos(n)$, $Sen(n)$

para j= 0, n-1

1. Fase de resolución de $L(:,j)$

1.2. Cálculo del elemento diagonal y preparación de b.

si ($j \in$ *mifila*) entonces

$\alpha = \sqrt{1 - S(j,j)^2}$, $L(j,j) = \alpha^{-1} R(j,j)$, $\beta(j) = L(j,j)S(j,j)$

difundir($L(j,j)$)

difundir($S(j,j)$)

si no

recibir(**map**(j),$seg(j)$)

recibir(**map**(j),$S(j,j)$)

$\beta = seg(j)S(j,j)$, $\alpha = \sqrt{1 - S(j,j)^2}$

fin si

para i= j+1, n-1

si ($i \in$ *mifila*) entonces

$b(i)$= -$\alpha R(i,j)$ –$\beta\, S(i,j)$

fin si

fin para

para i= j+1, n-1

si ($i \in$ *mifila*) entonces

1.3. Recibe el segmento de (p-1) elementos de L.

si ($i \neq j$+1) entonces

recibir(**map**(i-1), seg(máx(i-p+1, j+1):i-1))

fin si

1.4. Cálculo de $L(i, j)$

$$L(i,j) = \left(b(i) - S(j,j) \sum_{k=\max(i-p+1,\, j+1)}^{i-1} S(i,k)\, seg(k) \right) / (S(i,i)S(j,j) - 1)$$

$seg(i)=L(i,j)$

1.5. Envía el segmento actualizado de (p-1) elementos de L.

si ($i \neq n$-1) entonces

enviar(**map**(i+1), seg(máx(i-p+2, j+1):i))

fin si

1.6. Actualiza el vector auxiliar b.

para l=i+1, n-1

si ($l \in$ *mifila*) entonces

$$b(l) = b(l) - S(j,j) \sum_{k=\max(i-p+1,\, j+1)}^{i} S(i,k)\, seg(k)$$

fin si

fin para

fin si

fin para

2. Fase de triangulación.

2.1. Cálculo del vector **y**.
para i= j+1, n-1
si ($i \in$ *mifila*) entonces

$$y(i)= \alpha\left(seg(j)S(i,j) + \sum_{k=j+1}^{i} S(i,k)seg(k)\right) - S(j,j)R(i,j)$$

fin si
fin para
para i=j+1,n-1
si ($i \in$ *mifila*) entonces
2.2. Recibe el segmento de senos y cosenos.
si ($i \neq j$+1) entonces
recibir(**map**(i-1), *Cos*(máx(i-p, j+1):i-1))
recibir(**map**(i-1), *Sen*(máx(i-p, j+1):i-1))
fin si
2.3. Aplica las rotaciones de Givens previas sobre R(i,i) e y.
para k= máx(i-p, j+1), i-1
$\cos\theta_k = Cos(k)$, $\operatorname{sen}\theta_k = Sen(k)$

$$\begin{bmatrix} R(i,k) & y(i) \end{bmatrix} = \begin{bmatrix} R(i,k) & y(i) \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_k & \operatorname{sen}\theta_k \\ -\operatorname{sen}\theta_k & \cos\theta_k \end{bmatrix}$$

fin para
2.4. Calcula la rotación de Givens ($\cos\theta_i$,$\operatorname{sen}\theta_i$) tal que:

$$\begin{bmatrix} R(i,i) & y(i) \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_i & \operatorname{sen}\theta_i \\ -\operatorname{sen}\theta_i & \cos\theta_i \end{bmatrix} = \begin{bmatrix} * & 0 \end{bmatrix}$$

2.5. Envía el segmento de senos y cosenos.
Cos(i)= $\cos\theta_i$, Sen(i)= $\operatorname{sen}\theta_i$
si ($i \neq n$-1) entonces
enviar(**map**(i+1), *Cos*(máx(i-p+1, j+1):i))
enviar(**map**(i+1), *Sen*(máx(i-p+1, j+1):i))
fin si
2.6. Aplica las rotaciones de Givens al resto de filas.
para k= máx(i-p, j+1), i
para l=i+1, n-1
si ($l \in$ *mifila*) entonces

$$\begin{bmatrix} R(l,k) & y(l) \end{bmatrix} = \begin{bmatrix} R(l,k) & y(l) \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_k & \operatorname{sen}\theta_k \\ -\operatorname{sen}\theta_k & \cos\theta_k \end{bmatrix}$$

fin si
fin para
fin para
fin si
fin para
fin

En este algoritmo se produce un incremento en el tiempo de latencia de circulación del segmento (tiempo necesario para que el segmento complete una vuelta por el anillo) a medida que se incremente el número de procesadores. En el análisis teórico de los algoritmos, estudiaremos en detalle esta circunstancia.

También podemos intercalar las dos fases en las que se divide el cálculo, éstas son, resolución y triangulación, como se muestra en el algoritmo *PHCF2*. Este algoritmo podemos implementarlo de modo que dos segmentos, en vez de uno, estén circulando entre los procesadores. Reducimos por lo

tanto el tiempo de latencia entre mensajes y en consecuencia los tiempos de espera que puedan darse. Otra opción posible es el agrupamiento de los segmentos de las dos etapas en uno sólo de tamaño 3(*p*-1), consiguiendo un incremento en el tiempo de circulación del segmento.

Algoritmo 16 *[PHCF2]*

Memoria: $S(n/p,n)$, $R(n/p, n)$, $L(n/p, n)$

$seg(n)$, $Cos(n)$, $Sen(n)$

para j= 0, n-1

si ($j \in mifila$) entonces

$\alpha = \sqrt{1 - S(j,j)^2}$, $L(j,j) = \alpha^{-1} R(j,j)$, $\beta(j) = L(j,j)S(j,j)$

difundir($L(j,j)$)

difundir($S(j,j)$)

si no

recibir(**map**(j), $seg(j)$)

recibir(**map**(j), $S(j, j)$)

$\beta = seg(j)S(j,j)$, $\alpha = \sqrt{1 - S(j,j)^2}$

fin si

para i= j+1, n-1

si ($i \in mifila$) entonces

$b(i)$= -$\alpha R(i, j)$ –$\beta S(i, j)$

fin si

fin para

para i= j+1, n-1

si ($i \in mifila$) entonces

1. Fase de resolución de $L(i,j)$

1.1. Recibe el segmento de (p-1) elementos de L.

si ($i \neq j$+1) entonces

recibir(**map**(i-1), seg(máx(i-p+1, j+1):i-1))

fin si

1.2. Cálculo de $L(i, j)$

$$L(i,j) = \left(b(i) - S(j,j) \sum_{k=\max(i-p+1,\, j+1)}^{i-1} S(i,k)seg(k)\right) / (S(i,i)S(j,j)-1)$$

$seg(i)=L(i,j)$

1.3. Envía el segmento actualizado de (p-1) elementos de L.

si ($i \neq n$-1) entonces

enviar(**map**(i+1), seg(mín(i-p+2, j+1):i))

fin si

1.4. Actualiza el vector auxiliar b.

para l=i+1, n-1

si ($l \in mifila$) entonces

$$b(l)=b(l) - S(j,j) \sum_{k=\max(i-p+1,\, j+1)}^{i} S(i,k)\, seg(k)$$

fin si

fin para

2. Fase de triangulación.

2.1. Cálculo del vector **y**.

$$y(i) = \alpha \left(seg(j)S(i,j) + y(i) + \sum_{k=\max(i-p+1,\, j+1)}^{i} S(i,k)seg(k)\right) - S(j,j)R(i,j)$$

2.2.Recibe el segmento de senos y cosenos.
si ($i \neq n$-1) entonces
recibir(**map**(i-1), *Cos*(máx(i-p, j+1):i-1))
recibir(**map**(i-1), *Sen*(máx(i-p, j+1):i-1))
fin si
2.3. Aplica las rotaciones de Givens previas sobre R(i,i) e y.
para k= j+1, i-1
$\cos\theta_k$ = *Cos*(k), $\text{sen}\,\theta_k$= *Sen*(k)

$$\begin{bmatrix} R(i,k) & y(i) \end{bmatrix} = \begin{bmatrix} R(i,k) & y(i) \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_k & \text{sen}\,\theta_k \\ -\text{sen}\,\theta_k & \cos\theta_k \end{bmatrix}$$

fin para
2.4. Calcula la rotación de Givens ($\cos\theta_i$,$\text{sen}\,\theta_i$) tal que:

$$\begin{bmatrix} R(i,i) & y(i) \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_i & \text{sen}\,\theta_i \\ -\text{sen}\,\theta_i & \cos\theta_i \end{bmatrix} = \begin{bmatrix} * & 0 \end{bmatrix}$$

2.5. Envía el segmento de seno y coseno.
Cos(i)= $\cos\theta_i$, *Sin*(i)= $\text{sen}\,\theta_i$
si ($i \neq n$-1) entonces
enviar(**map**(i+1), *Cos*(máx(i-p+1, j+1):i))
enviar(**map**(i+1), *Sen*(máx(i-p+1, j+1):i))
fin si
2.6. Aplica las rotaciones de Givens al resto de filas.
para l=i+1, n-1
si ($l \in$ *mifila*) entonces

$$y(l) = y(l) + \sum_{k=\max(i-p,\,j+1)}^{i} S(l,k)\,seg(k)$$

in si
fin para
fin si
fin para
fin para
fin

En máquinas donde el coste de comunicaciones sea muy elevado comparado con la capacidad computacional de sus nodos el siguiente paso será aumentar el grano de la resolución [Li 88]. Esto nos lleva a una distribución de los datos por bloques de filas como generalización de la distribución de la Figura 23. Consideraremos que q es el número de filas de cada bloque.

Los algoritmos por bloques que se pueden derivar de esta distribución son generalizaciones de los algoritmos PHCF y *PHCF2* descritos anteriormente. El algoritmo cíclico por bloques relacionado con *PHCF*, que denominaremos *PHCM*, haría circular en cada una de sus etapas, resolución y triangulación, segmentos de tamaño q(p-1) y 2q(p-1), respectivamente.

Algoritmo 17 *[PHCM]*

para j= 0, n-1
para i= j, n-1, q
1. Fase de resolución de L(i:i+q-1, j)
fin para
para i= j, n-1, q

2. Fase de triangulación para (R, $\mathbf{y}(i{:}i+q-1)$)

fin para

fin para

fin

En el algoritmo *PHCM2* se muestra la generalización del algoritmo *PHCF*, donde sólo circulará un segmento de tamaño $3q(p-1)$.

Algoritmo 18 *[PHCM2]*

para j=0, n-1

para i=j, n-1, q

1. Fase de resolución de $L(i{:}i+c-1,j)$

2. Fase de triangulación para (R, $\mathbf{y}(i{:}i+c-1)$)

fin para

fin para

fin

Otro posible algoritmo que aumenta el grano de resolución del problema, sin tener que variar la distribución de los datos que vimos en la Figura 23, consiste en calcular en cada paso q elementos pertenecientes a la misma fila. Debido a la dependencia de datos del algoritmo de Hammarling, en este caso no se podrán separar las fases de resolución y triangulación (ver Figura 2) tal y como lo hemos hecho anteriormente. En la Figura 25 mostramos la secuencia de resolución del problema utilizando este nuevo algoritmo que denominaremos *PHCMR*.

Como se puede observar, en el algoritmo *PHCMR* no se separan las fases de resolución y triangulación. Como consecuencia de ello algunas variables del algoritmo, como **y**, pasarán de ser vectores de n elementos a matrices de $n \times q$ y el segmento que circulará entre los procesadores será de tamaño $3q(p-1)$.

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

Figura 25. *Secuencia de resolución para el algoritmo PHCMR con n=9, p=3 y q=3 (Procesador, Paso).*

Algoritmo 19 *[PHCMR]*

para j=0, n-1, q

para i= j, n-1

1. Fase de resolución y triangulación ($L(i, j{:}j+q\text{-}1)$;R, $\mathbf{y}(i, j{:}j+q\text{-}1)$)

fin para

fin para

fin

Análisis teórico de los algoritmos

En el análisis de estos algoritmos el coste computacional (aritmético) $C_A(i,j)$ para el elemento $L(i,j)$, lo dividimos en dos partes C_R y C_T, asociadas a las fases *resolución* y *triangulación*, respectivamente. De esta forma tendremos que $C_A(i,j)=C_R(i,j)+C_T(i,j)$. Cada una de las fases, resolución y triangulación, se puede dividir a su vez en dos etapas, la primera que denominaremos de cálculo incluirá las operaciones necesarias para calcular un elemento cuando el segmento llega a un procesador. La segunda está dedicada a la actualización que, como hemos descrito, se intenta solapar con la circulación del segmento o segmentos a través de los procesadores. Así , para la fase de resolución (triangulación) los costes asociados a cada una de estas partes serán C_{RC} y C_{RA} (C_{TC} y C_{TA}). Definimos como $C_{kp}= \sigma +k(p\text{-}1)\tau$ el coste de comunicación requerido para transmitir un segmento de tamaño $k(p\text{-}1)$.

Consideramos en nuestro primer análisis el algoritmo *PHCF* en el que para cada columna se realiza primero la fase de resolución y después la fase de triangulación. En la fase de resolución se realiza en primer lugar el cálculo un elemento de L, después se transmite un segmento de tamaño (p-1), y a continuación se actualiza el resto de la columna. El coste asociado a la fase de resolución del elemento $L(i,j)$ será

$$C_R^{PHCF}(i,j) = C_{RC}^{PHCF}(i,j) + C_{RA}^{PHCF}(i,j) + C_{RE}^{PHCF}(i,j),$$

donde $C_{RE}^{PHCF}(i,j)$ es el coste debido al tiempo de espera del segmento para este elemento en la fase de resolución. Éste viene determinado por la diferencia entre el coste de la fase de actualización para elemento $L(i,j)$ y el coste temporal necesario para que el segmento recorra los p-1 procesadores, realizando cada uno de ellos la fase de cálculo de un elemento, y sea recibido otra vez el segmento ya actualizado

$$C_{RE}^{PHCF}(i,j) = \max\{0, \sum_{k=i-p+1}^{i-1} C_{RC}^{PHCF}(k,j) + pC_p - C_{RA}^{PHCF}(i-p,j)\}.$$

El término correspondiente a la circulación del segmento, pC_p, se ha definido de este modo suponiendo que la recepción de datos en nuestro modelo de comunicaciones no es bloqueante. Si suponemos un modelo de comunicaciones con recepción bloqueante, el término asociado a la circulación del segmento debe ser $(p-1)C_p$.

En la fase de triangulación se realiza en primer lugar el cálculo un elemento de $y(i)$ y se obtiene la rotación de Givens necesaria para anularlo, después se transmite un segmento de tamaño $2(p\text{-}1)$ formado por los senos y cosenos de las rotaciones que intervienen todavía en alguna actualización y a continuación se hace la actualización de las rotaciones en los elementos de **y** asociados a la columna en curso. El coste asociado a la fase de triangulación del elemento $L(i,j)$ será

$$C_T^{PHCF}(i,j) = C_{TC}^{PHCF}(i,j) + C_{TA}^{PHCF}(i,j) + C_{TE}^{PHCF}(i,j),$$

donde $C_{TE}^{PHCF}(i,j)$ es el coste debido al tiempo de espera del segmento para este elemento en la fase de triangulación. Éste viene determinado por la diferencia entre el coste de la fase de actualización para elemento $L(i,j)$ y el coste temporal necesario para que el segmento recorra los p-1 procesadores, realizando cada uno de ellos la fase de cálculo de una rotación, y sea recibido otra vez el segmento ya actualizado

$$C_{TE}^{PHCF}(i,j) = \text{máx}\{0, \sum_{k=i-p+1}^{i-1} C_{TC}^{PHCF}(k,j) + pC_{2p} - C_{TA}^{PHCF}(i-p,j)\}.$$

El coste total del algoritmo viene dado por

$$C_{PHCF} = \sum_{j=1}^{n} \sum_{l=j/p}^{n/p} C_{PHCF}(i,j) = \tfrac{5n^3}{3p} + C_{RE}^{PHCF} + C_{TE}^{PHCF} + O(n^2),$$

donde $i=(l\text{-}1)p+1$.

Es difícil expresar el coste total debido a las esperas, pero podemos estudiar cuál será el comportamiento de éstas en cada una de las fases del algoritmo *PHCF*. En la fase de resolución el coste de las esperas para cada elemento será

$$C_{RE}^{PHCF}(i,j) = \text{máx}\{0, (p^2 + 4p - 5) + p(\sigma + (p-1)\tau) - 4(n + p - i - 1)\}.$$

Así, las esperas que se puedan producir en esta fase aparecerán en cuando se resuelvan los últimos elementos de cada columna, pues será cuando el término $2p(n\text{-}i\text{-}1)$ será más pequeño y el trabajo de actualización menor.

En la fase de triangulación el coste de la espera para cada elemento será de

$$C_{TE}^{PHCF}(i,j) = \text{máx}\{0, (6p^2 + 4p - 10) + p(\sigma + 2(p-1)\tau) - 6(n + p - i - 1)\}.$$

Las esperas en esta fase también aparecerán y serán mayores a medida que resolvamos cada columna y podemos ver como éstas serán mayores que en la fase anterior debido a las comunicaciones. Podemos observar también que las esperas se incrementarán al aumentar el número de procesadores.

Consideramos ahora el análisis del algoritmo *PHCF2* en el que para cada elemento se realizan juntas las etapas de cálculo, tanto de la fase de resolución como la de triangulación, después se transmite un segmento de tamaño $3(p\text{-}1)$ y a continuación se hace la actualización. El coste para calcular el elemento $L(i,j)$ será

$$C_{PHCF2}(i,j) = C_A^{PHCF2}(i,j) + C_E^{PHCF2}(i,j),$$

donde $C_E^{PHCF}(i,j)$ es el coste debido al tiempo de espera del segmento. Este viene determinado por la diferencia entre el coste de la fase de actualización para elemento $L(i,j)$ y el coste temporal necesario para que el segmento recorra los p-1 procesadores, realizando cada uno de ellos la fase de cálculo, y sea recibido otra vez el segmento ya actualizado

$$C_E^{PHCF2}(i,j) =$$

$$\text{máx}\{0, \sum_{k=i-p+1}^{i-1}\left(C_{RC}^{PHCF2}(k,j) + C_{TC}^{PHCF2}(k,j)\right) + pC_{3p} - \left(C_{RA}^{PHCF2}(i-p,j) + C_{TA}^{PHCF2}(i-p,j)\right)\}$$

El coste total de este algoritmo viene dado por

$$C_{PHCF2} = \sum_{j=1}^{n}\sum_{l=j/p}^{n/p} C_{PHCF2}(i,j) = \tfrac{5n^3}{3p} + C_E^{PHCF} + O(n^2),$$

donde i=(l-1)p+1.

Si estudiamos el comportamiento de las esperas en este algoritmo, sabiendo que las fases de resolución y triangulación se realizan juntas, obtenemos que el coste de las esperas para cada elemento serán de

$$C_{RE}^{PHCF2}(i,j) = \text{máx}\{0, (6(p-1)(i-j) + p(12-2p) - 11)) + p(\sigma + 3(p-1)\tau) - 3(n+p-i-1)\}.$$

Como se puede observar se produce un incremento de las esperas respecto del algoritmo *PHCF*. Por lo tanto el algoritmo *PHCF2* nos dará unos resultados peores que el algoritmo *PHCF*.

Para paliar las esperas en el algoritmo *PHCF2* podemos resolver las etapas de resolución y triangulación por separado, teniendo las esperas para resolver un elemento las expresiones

$$C_{RE}^{PHCF2}(i,j) = \text{máx}\{0, \sum_{k=i-p+1}^{i-1} C_{RC}^{PHCF2}(k,j) + pC_p - (C_{RA}^{PHCF2}(i-p,j) + C_T^{PHCF2}(i-p,j)\}$$

y

$$C_{TE}^{PHCF2}(i,j) = \text{máx}\{0, \sum_{k=i-p+1}^{i-1} C_{TC}^{PHCF2}(k,j) + pC_{2p} - (C_{TA}^{PHCF2}(i-p,j) + C_R^{PHCF2}(i-p,j)\}.$$

Pero ambas expresiones están relacionadas por lo que seguirán existiendo esperas en el algoritmo. En el mejor de los casos sólo aparecerá la espera $C_{TE}^{PHCF2}(i,j)$ que tendrá la expresión

$$C_{RE}^{PHCF2}(i,j) = \text{máx}\{0, (6(p-1)(i-j) + p(9-2p) - 6)) + p(\sigma + 3(p-1)\tau) - 5(n+p-i)\}.$$

Para reducir el coste debido a las comunicaciones, se han propuesto diversos algoritmos que incrementan el grano de resolución del problema (los algoritmos *PHCM, PHVM2* y *PHCMR*). El incremento del grano de resolución redunda en una reducción en el tiempo de comunicación al reducirse el número se mensajes. Así, si la latencia, σ, es mucho mayor que la inversa del ancho de banda, τ, es posible mejorar las prestaciones del algoritmo. Del estudio realizado podemos deducir que el algoritmo que nos ofrecerá mejores prestaciones es el *PHCM*, propuesto como extensión del algoritmo *PHCF*. Como contrapartida tendremos que las esperas no se van a reducir sustancialmente

pues sólo disminuimos la componente de comunicaciones asociada al envío de cada mensaje o latencia y estamos reduciendo el paralelismo implícito del problema al reducir el número de pasos en su resolución.

5.2.3.4 Algoritmos cíclicos modificados

Li y Coleman [Li 89] y Eisensat *et al.* [Eisensat 88] proponen estrategias dirigidas a reducir las esperas de los algoritmos cíclicos en la resolución de sistemas triangulares lineales mediante la división del segmento que circula entre los procesadores en subsegmentos. Existen varias opciones para conseguir este objetivo.

La primera opción consiste en que una vez utilizado por un procesador un subsegmento, éste es inmediatamente enviado al siguiente procesador. Se consigue entonces un efecto de tubería, por eso a los algoritmos cíclicos con esta modificación se les denomina algoritmos cíclicos segmentados (*pipelined cyclic algorithms*). Se consigue que varios procesadores estén trabajando con el mismo segmento simultáneamente, aunque se produce un incremento en la latencia en la transmisión de los mensajes. Para contrarrestar este efecto y tener una reducción efectiva de las esperas habrá que experimentar con el tamaño de los subsegmentos para cada computador.

La segunda opción es enviar cada subsegmento a un procesador distinto. Así, los procesadores más alejados topológica o geográficamente en el anillo unidireccional o hipercubo del procesador que tiene el segmento pueden ir adelantando las actualizaciones [Li 89]. Este tipo de algoritmo se denominan algoritmos cíclicos con transferencia adelantada o con atajos (*short-cut cyclic algorithms*). Con esta filosofía se proponen diversos esquemas atendiendo a la forma en que se divide el segmento en subsegmentos y a qué procesadores son enviados dichos subsegmentos.

La forma más simple de dividir el segmento de tamaño p-1 es hacerlo en q partes iguales o subsegmentos [Li 89]. Así, si consideramos la resolución de un sistema triangular lineal mediante eliminación progresiva, un procesador dado **map**(i) enviará siempre el primer subsegmento al siguiente procesador en el anillo **map**(i+1) y los q-1 subsegmentos restantes al resto de los procesadores equidistantes alrededor del anillo en intervalos correspondientes al tamaño del subsegmento (ver la ilustración izquierda de la Figura 26). Por otra parte, durante la resolución de cada elemento, cada procesador recibirá q subsegmentos, como se muestra a la derecha de la Figura 26. En esta Figura y en las siguientes, los procesadores vienen indicados por círculos que contienen los números de procesador, conectados mediante una topología de anillo. Los mensajes están indicados mediante flechas, con los tamaños de los mensajes indicados en ellas. La suma de los tamaños de los mensajes es siempre igual a p-1. La presencia de una flecha no implica necesariamente una conexión física entre los procesadores.

Una descripción completa de este tipo de algoritmos cíclicos modificados puede consultarse en [Li 89]. Nosotros consideraremos como en [Li 89] que q divide a p, aunque una generalización de esta circunstancia para un valor arbitrario de q es fácil de implementar. Este tipo de algoritmos, en los que los subsegmentos tienen todos el mismo tamaño y donde los procesadores destino se encuentran uniformemente espaciados los denominaremos algoritmos cíclicos con transferencia adelantada uniforme (*uniform short-cut cyclic algorithm* [Eisensat 88]).

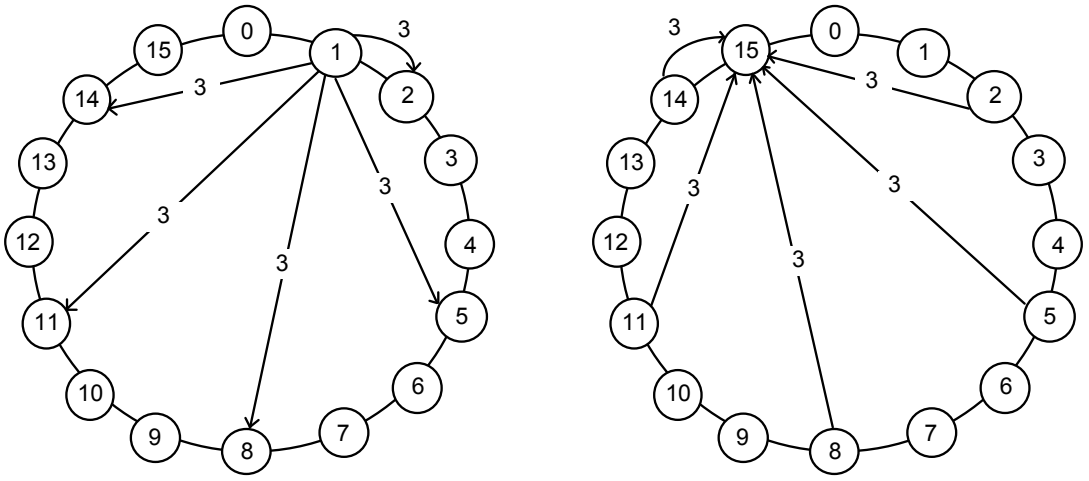


Figura 26. *Mensajes enviados por el nodo 1 y mensajes recibidos por el nodo 15 durante el algoritmo cíclico con transferencia adelantada uniforme; arcos etiquetados con el tamaño del mensaje.*

Puesto que el último subsegmento recibido por un procesador en el cálculo de un elemento cualquiera es el primero en ser enviado al siguiente procesador en el anillo es posible que no sea la mejor opción que todos los subsegmentos tengan el mismo tamaño. Así, parece lógico pensar que obtendremos mejores resultados si se envían segmentos de tamaño pequeño a los procesadores más cercanos y segmentos de tamaño grande a los procesadores más alejados [Eisensat 88]. Este tipo de algoritmos de denominan algoritmos cíclicos con transferencia adelantada geométrica (*geometric short-cut cyclic algorithm* [Eisensat 88]). Estos algoritmos deben su nombre a que el tamaño de los subsegmentos viene dada por una progresión geométrica. De este modo, un procesador enviará subsegmentos de tamaños 2, 4,…, $p/2$ a los procesadores destino correspondientes que estarán espaciados también del mismo modo a lo largo del anillo. Sólo habrá un segmento de tamaño 1 que será enviado al procesador que le precede inmediatamente en el anillo al que envía los segmentos, de tal forma que la suma de todos los tamaños de subsegmentos sea de p-1. El esquema de comunicación para este tipo de algoritmos se ilustra en la Figura 27.

Dependiendo de la arquitectura del computador, lo procesadores fuente y destino en la transmisión de un mensaje adelantado, pueden o no estar conectados físicamente por una enlace directo. Cuando no existe la conexión (en un hipercubo, por ejemplo, esta situación es inevitable tanto para el caso uniforme como para el caso geométrico) el mensaje es encaminado hasta el procesador destino a través de procesadores intermedios. Esto supone un incremento en el coste total del algoritmo y por lo tanto, una reducción de las prestaciones esperadas. Con el objetivo de que todas transmisiones adelantadas de subsegmentos se realicen a través de enlaces directos, Eisensat *et al*. [Eisensat 88] han desarrollado una variante del algoritmo cíclico con transferencia adelantada geométrica que denominan algoritmo cíclico con transferencia adelantada directa (*direct short-cut cyclic algorithm*) pensada, para multicomputadores con topología de hipercubo.

Los resultados experimentales en [Eisensat 88] demuestran claramente que, para la resolución de sistemas triangulares, los algoritmos que presentan mejores prestaciones y mayor escalabilidad son los algoritmos con transferencia adelantada geométrica en el caso del anillo y la versión directa de estos algoritmos para la topología hipercubo.

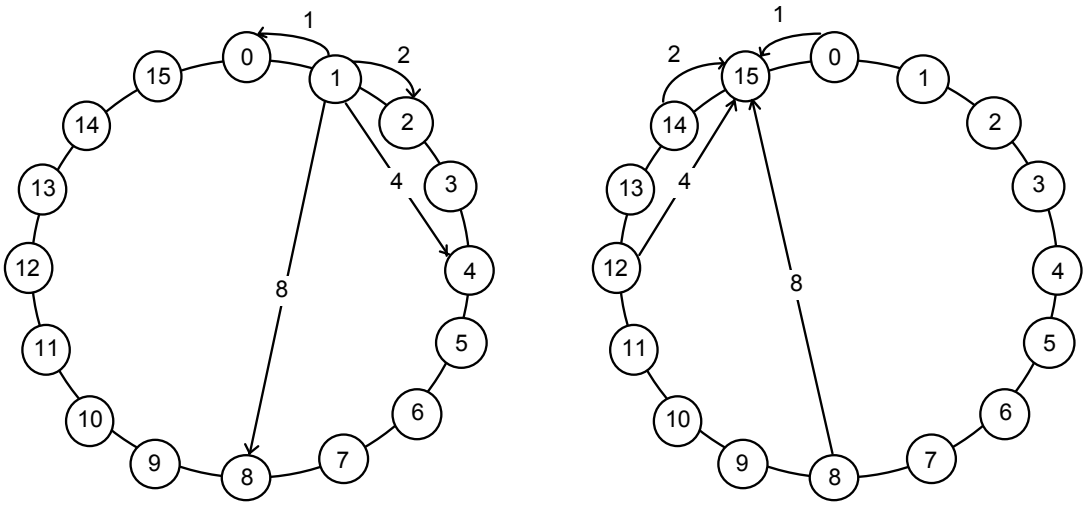


Figura 27. *Mensajes enviados por el nodo 1 y mensajes recibidos por el nodo 15 durante el algoritmo cíclico con transferencia adelantada geométrica; arcos etiquetados con el tamaño del mensaje.*

Estos algoritmos cíclicos modificados diseñados para resolver sistemas triangulares pueden adaptarse para resolver las ecuaciones de Lyapunov, para tiempo discreto (o continuo). Así, se han diseñado algoritmos modificando el algoritmo *PHCF* mediante transferencia adelantada uniforme (algoritmo *PHCFU*) y geométrica (algoritmo *PHCFG*). Al igual que en el algoritmo *PHCF*, nuestros algoritmos cíclicos modificados aplican las ideas expuestas en la resolución de cada columna de la matriz solución, funcionando en dos etapas, una primera de resolución y una segunda de triangulación.

5.2.3.5 Algoritmos cíclicos frente de onda

Los resultados en [Eisensat 88] fueron obtenidos en multicomputadores muy equilibrados (la velocidad de las comunicaciones y la velocidad de cálculo de los procesadores es muy parecida). Sin embargo, en los multicomputadores actuales el incremento en la velocidad de los computadores es mucho mayor que el de las redes de interconexión. Por otra parte existen actualmente muchos sistemas multiprocesadores con topología de bus que tienen pocos procesadores, pues a partir de 4 las prestaciones del sistema comienzan a degradarse considerablemente. Por todo ello proponemos para resolver las ecuaciones de Lyapunov una variación en los algoritmos cíclicos antes descritos basada en que los procesadores resuelvan al mismo tiempo no una única columna cada vez sino un bloque de r columnas, con n múltiplo de r.

Algoritmo 20 *[PHCWF]*.

para j= 0, n-1, r
  para i= j, n-1
    para k= 0, r-1
      1. Fase de resolución de $L(i,j+k)$
      2. Fase de triangulación para (R, $\mathbf{y}(i,j+k)$)
    fin para
  fin para
fin para

fin

Este algoritmo, que denominaremos *PHCWF*, lo podemos obtener a partir del algoritmo *PHCF2*, ya que en este algoritmo las etapas de resolución y triangulación asociadas a un elemento solución se realizan juntas y permite que el cálculo de varias columnas pueda evolucionar simultáneamente (con las restricciones debidas a la dependencia de datos). Se mantiene como en los algoritmos cíclicos anteriores una distribución cíclica de las filas de S y R, y los resultados se almacenarán también de forma cíclica. Sin embargo, la ecuación se resuelve de forma parecida a un frente de onda de antidiagonales de tamaño r como se muestra en la Figura 28. Esta característica hace que a estos algoritmos los denominemos algoritmos cíclicos de frente de onda. Conseguimos de este modo un mayor aprovechamiento del paralelismo implícito que posee el problema.

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

Figura 28. *Secuencia de resolución para el algoritmo PHCWF con n=9, p=3 y r=p (Procesador, Paso).*

En el algoritmo *PHCF2* había un único segmento de tamaño 3(p-1) circulando por el anillo de procesadores si enviábamos unidos los segmentos de las fases de resolución y triangulación. En el algoritmo *PHCWF* circularán r segmentos de tamaño 3(p-1) en el caso de que $r < p$. Si ($p/2 < r < p$) y mantenemos las dos fases separadas, circularán p segmentos, de tal forma que si circulan t segmentos ($t \le r$) de tamaño (p-1), simultáneamente lo harán p-t segmentos de tamaño 2(p-1).

Para $r > p$, tenemos que circulan como máximo p segmentos. Estos serán de igual tamaño en el caso de que los segmentos de las fases de resolución y triangulación se envíen juntos y distintos en el caso de mantener dos fases separadas de cálculo para cada elemento.

Un caso de especial interés es aquél en que $r=p/2$, ya que podemos tener circulando cuando existan dos fases de cálculo separadas, segmentos de tamaño (p-1) y 2(p-1), es decir, la mitad de los procesadores estará en la fase de resolución mientras que la otra mitad estará en la fase de triangulación. Se tratará por lo tanto de un doble frente de onda, cada uno asociado a una fase.

Para valores de $r > p$ tenemos, como muestra la Figura 29, para el caso particular de $r=n$, que se consigue el mismo grado de paralelismo que aparecía en el caso de los algoritmos de frente de onda. Incluso es posible un aumento de prestaciones, hasta cierto punto, debido a la reducción considerable de las esperas. La contrapartida a estas ventajas será la pérdida de la localidad de los cálculos al aumentar el número de columnas que se calculan simultáneamente. Por otra parte, un incremento en el grano de resolución (por ejemplo, mediante vectores) no será una buena opción ya que este tipo de algoritmos se obtienen como generalización del algoritmo *PHCF2* y, como vimos en el apartado anterior, las esperas se incrementan a medida que resolvemos una columna. Esto es debido

a un mayor coste en el cálculo de los elementos de la solución a medida que resolvemos una columna y a un menor tiempo en la actualización de los restantes elementos de ésta. Todo ello supondrá, aún sin incrementar el grano de resolución, que las esperas que están asociadas al cálculo de cada elemento sean diferentes. Por otra parte en los computadores modernos los algoritmos con un número elevado de pequeñas comunicaciones son penalizados en sus prestaciones. Todo ello nos lleva a proponer una variación en el algoritmo *PHCWF* que consistirá en agrupar el cálculo de los elementos de varias columnas pertenecientes a la misma fila y agrupar también el envío de los segmentos correspondientes a ellas.

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

Figura 29. *Secuencia de resolución para el algoritmo PHCWF con n=9, p=3 y r=n (Procesador, Paso).*

Si $L(i, j{:}j+r-1)$ es el vector fila de un bloque de columnas que tiene que calcular el procesador **map**(i), podemos dividir uniformemente el vector en subvectores de tamaño q de tal forma que en cada paso se calcule un subvector fila $L(i, j+q(k-1){:}\ j+qk-1)$ donde k=1, 2, ... , r/q. Los segmentos que se envían son todos de tamaño $3q(p-1)$. A este algoritmo cíclico de frente de onda en el que se resuelve cada vez un subvector fila de tamaño uniforme lo denominaremos *PHCWFU* y en la Figura 30 podemos ver la secuencia de resolución.

0,1								
1,2	1,2							
2,3	2,3	2,4						
0,4	0,4	0,5	0,5					
1,5	1,5	1,6	1,6	1,10				
2,6	2,6	2,7	2,7	2,11	2,11			
0,7	0,7	0,8	0,8	0,12	0,12	0,13		
1,8	1,8	1,9	1,9	1,13	1,13	1,14	1,14	
2,9	2,9	2,10	2,10	2,14	2,14	2,15	2,15	2,16

Figura 30. *Secuencia de resolución para el algoritmo PHCWFU con n=9, p=3, r=4 y q=2 (Procesador, Paso).*

Otra opción es dividir el vector fila según una progresión geométrica que determina nuestro frente de onda. Dado que el cálculo de los elementos más a la izquierda del vector fila $L(i, j:j+r-1)$ tiene un coste mayor que el de los elementos más a la derecha, con esta división se puede equilibrar el coste del cálculo entre los subvectores del vector fila que ahora no serán iguales. Pero la relación de costes entre los elementos de una fila de la solución no son exponenciales, por lo que proponemos una división heurística en la que se prueba para cada caso la división que nos permita obtener mejores prestaciones para una máquina concreta. A este algoritmo lo denominaremos *PHCWFN*.

Por último se propone una versión adaptativa del algoritmo *PHCWFU*, y que denominaremos *PAHCWFU*. El propósito de este algoritmo es adaptar el tamaño del bloque de columnas que resuelven los procesadores a medida que se calcula la solución de la ecuación, como muestra la Figura 31.

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

Figura 31. *Secuencia de resolución para el algoritmo PAHCWFU con n=9, p=3, r=6 y3 y q=2 y1 (Procesador, Paso).*

En este algoritmo, cuando se cumpla una cierta condición *condbloque* definida por la función **cond**, el tamaño de los bloques columna que están resolviendo los procesadores se reducirá en un factor *f,* y el nuevo tamaño de bloque será de *r/f*. Para mantener el mismo nivel de paralelismo, se reducirá en la misma proporción el tamaño del subvector que se calcula en cada paso para un vector fila del bloque columna; en consecuencia el nuevo tamaño de subvector será de *q/f*. Los valores de *condbloque,* los valores de *r* y *f*, y el tamaño mínimo del bloque columna a resolver, q^{min}, para obtener las mejores prestaciones del algoritmo dependen de la máquina y se determinarán de forma experimental.

Función 2 cond(*j,p*)
si ($q > q^{min}$) entonces
si ($\frac{n-jq}{q} <$ *condbloque*) entonces
devuelve *verdadero*
sino
devuelve *falso*
fin si
sino
devuelve *falso*
fin si
fin

Análisis teórico de los algoritmos

El coste de estos algoritmos se realiza de forma similar al de los algoritmos cíclicos, aunque ahora sólo consideraremos el caso en el que se envía un único segmento de tamaño 3(*p*-1). Consideramos de mayor interés el análisis del algoritmo cíclico frente de onda *PHCWF* pues los algoritmos *PHCFU* y *PHCFG* siguen un patrón de comportamiento similar al algoritmo *PHCF* con el envío separado de los segmentos asociados a la fase de resolución y triangulación.

El coste del algoritmo *PHCWF* para calcular el elemento $L(i,j)$ será

$$C_{PHCWF}(i,j)= C_A^{PHCWF}(i,j)+C_E^{PHCWF}(i,j),$$

donde $C_E^{PHCWF}(i,j)$ es el coste debido al tiempo de espera del segmento. Éste viene determinado por la diferencia entre el coste de la fase de actualización para elemento $L(i,j)$ y el coste temporal necesario para que el segmento recorra los p-1 procesadores, realizando cada uno de ellos la fase de cálculo, y sea recibido otra vez el segmento ya actualizado

$$C_E^{PHCWF}(i,j) = \max\{0,\left(\sum_{k=i-p+1}^{i-1} C_{RC}^{PHCWF}(k,j)+C_{TC}^{PHCWF}(k,j)\right)+(p-r)C_{3p}-$$

$$\left(\sum_{k=j+1}^{j+r-1} C_{RC}^{PHCWF}(i-p,k)+C_{TC}^{PHCWF}(i-p,k)\right)-C_{RA}^{PHCWF}(i-p,j)+C_{TA}^{PHCWF}(i-p,j)\}.$$

El coste total de este algoritmo viene dado por

$$C_{PHCWF} = \sum_{j=1}^{n}\sum_{l=j/p}^{n/p} C_{PHCWF}(i,j) = \tfrac{5n^3}{3p}+C_E^{PHCWF}+O(n^2),$$

donde $i=(l-1)p+1$.

Si estudiamos el comportamiento de las esperas en este algoritmo, considerando que las fases de resolución y triangulación se realizan simultáneamente, obtenemos que el coste de las esperas para cada elemento será de

$$C_{RE}^{PHCWF}(i,j) = \max\{0,\ (6(p-1)(i-j)+p(7-2p)-11)+p(\sigma+3(p-1)\tau)$$

$$-(6(r-1)(i-j)+r(14-3r)-11)-3(n+p-i-1)\}.$$

En el caso particular de que $r=p$ el coste de las esperas asociado a un elemento $L(i,j)$será

$$C_{RE}^{PHCWF}(i,j) = \max\{0,\ p^2(3\tau+1)+p(\sigma-3\tau-10)-3(n-i-1).$$

Como se puede observar, se produce una disminución de las esperas respecto del algoritmo *PHCF2*. Teóricamente éstas pueden llegar incluso a ser nulas. El estudio de los algoritmos *PHCWFU* y *PHCWFG*, no lo realizaremos por ser muy similar al del algoritmo *PHCWF*, sólo diferenciándose de éste en el grano de las comunicaciones.

5.3 Paralelización del Método de la función signo matricial

En esta sección abordamos la paralelización de los algoritmos de resolución de las ecuaciones de Lyapunov mediante el método de la función signo matricial. Como hemos podido ver en el capítulo 3, en los algoritmos basados en este método predominan los productos de matrices, las inversiones de matrices y la resolución de sistemas triangulares. Por esa razón, la paralelización de estos algoritmos la hemos abordado mediante un nivel de descripción más alto que en el caso del método de Hammarling. Así, si los algoritmos paralelos basados en el método de Hammarling se han descrito e implementado utilizando operaciones aritméticas básicas y, en el caso de paso de mensajes, comunicaciones punto a punto, los algoritmos paralelos basados en la función signo matricial se describirán e implementarán utilizando operaciones aritméticas matriciales y comunicaciones de más alto nivel.

Dado que las librerías paralelas descritas en el capítulo 4, en particular el ScaLAPACK, están imprimiendo actualmente un nuevo estilo en el diseño de programas paralelos en computación numérica, tanto en sistemas multiprocesadores con memoria compartida como distribuida, utilizaremos las rutinas de estas librerías, junto con las del LAPACK, para describir en detalle como se han diseñado e implementado estos algoritmos.

Antes de abordar la descripción de los algoritmos paralelos para resolver las ecuaciones de Lyapunov, veamos como implementa el ScaLAPACK uno de los problemas más característicos del Álgebra Lineal como es la factorización LU de una matriz sobre una malla de 3×3 procesadores.

5.3.1 Ejemplo de paralelización en el ScaLAPACK: Factorización LU

Dada una matriz $A \in \mathsf{R}^{n \times n}$, queremos obtener su descomposición LU, $A=LU$, donde $L \in \mathsf{R}^{n \times n}$ es una matriz triangular inferior unitaria y $U \in \mathsf{R}^{n \times n}$ es una matriz triangular superior. El método utilizado para esta factorización se basa en la división en bloques de la matriz A de la siguiente forma

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}. \tag{5.8}$$

Si dividimos de igual modo las matrices L y U tenemos que

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}. \tag{5.9}$$

La correspondencia entre los bloques de A y las matrices L y U es la siguiente:

$$A_{11} = L_{11}U_{11}; \tag{5.10}$$

$$A_{21} = L_{21}U_{11}, \text{ de donde } L_{21} = A_{21}U_{11}^{-1}; \tag{5.11}$$

$$A_{12} = L_{11}U_{12}, \text{ de donde } U_{12} = L_{11}^{-1}A_{12}; \tag{5.12}$$

$$A_{22} = L_{21}U_{12} + L_{22}U_{22}\text{, de donde } \widetilde{A}_{22} = L_{22}U_{22} = A_{22} - L_{21}U_{12}. \quad (5.13)$$

De las expresiones anteriores se deduce que puede resolverse la descomposición LU de la ecuación (5.10), obteniendo L_{11} y U_{11}, y a continuación calcular L_{21} y U_{12} de las expresiones (5.11) y (5.12), respectivamente. Por último, de la ecuación (5.13) obtenemos la matriz $\widetilde{A}_{22}$ que puede volver a factorizarse del mismo modo que A sin necesidad de volver a utilizar más las submatrices A_{11}, A_{21} y A_{12}.

La matriz A se distribuye en la malla de procesadores de forma toroidal por bloques de tamaño $n_b \times n_b$, como se ilustra en la Figura 32 (salvo los últimos bloques fila y columna que pueden tener un tamaño menor).

P_{00}	P_{01}	P_{02}	P_{00}	P_{01}
P_{10}	P_{11}	P_{12}	P_{10}	P_{11}
P_{20}	P_{21}	P_{22}	P_{20}	P_{21}
P_{00}	P_{01}	P_{02}	P_{00}	P_{01}
P_{10}	P_{11}	P_{12}	P_{10}	P_{11}

Figura 32. *Distribución toroidal por bloques de la matriz A en la malla de procesadores 3×3.*

Así los bloques de la matriz A según la expresión (5.8) quedan definidos en la distribución de la matriz sobre la malla como se ilustra en la Figura 33.

A partir de esta distribución, y siguiendo el procedimiento descrito en las ecuaciones (5.8) - (5.13), podemos describir los pasos que se siguen para implementar este algoritmo en un sistema multiprocesador con memoria distribuida:

1. $A_{11} = L_{11}U_{11}$. La submatriz A_{11} se encuentra en el procesador P_{oo}, por lo que la factorización LU de este bloque se realiza localmente en este procesador. A continuación P_{oo} difunde U_{11} al resto de los procesadores de la columna 0, P_{10} y P_{20}, y L_{11} al resto de los procesadores de la fila 0, P_{01} y P_{02}.

2. $L_{21} = A_{21}U_{11}^{-1}$. La submatriz A_{21} está distribuida entre los procesadores P_{x0} que resuelven este sistema triangular superior para obtener la submatriz resultado L_{21}. Esta submatriz quedará distribuida del mismo modo que A_{21}.

3. $U_{12} = L_{11}^{-1}A_{12}$. La submatriz A_{12} está distribuida entre los procesadores P_{0y} que resuelven este sistema triangular inferior para obtener la submatriz resultado U_{12}. Esta submatriz quedará distribuida del mismo modo que A_{12}.

4. $\widetilde{A}_{22} = L_{22}U_{22} = A_{22} - L_{21}U_{12}$. Esta operación consiste en una actualización de A_{22} mediante un producto de matrices. Para ello, los procesadores P_{x0} difunden sus bloques de la submatriz L_{21} a los procesadores de su propia fila (por ejemplo, P_{00} difunde sus bloques de a los procesadores P_{01} y P_{02}). De forma similar los procesadores P_{0y} difunden sus bloques de la submatriz U_{12} a los procesadores de

su propia columna (por ejemplo, P_{00} difunde sus bloques de U_{12} a los procesadores P_{10} y P_{20}). Una vez esta información está disponible, el bloque A_{22} se actualiza localmente mediante un producto de matrices.

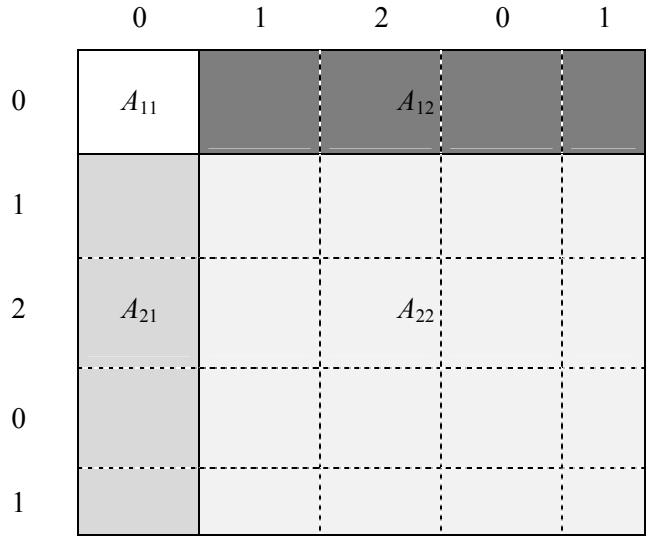


Figura 33. *División por bloques de la matriz A según la expresión* (5.8) *y su distribución en una malla* 3×3. *Los números alrededor de la matriz constituyen las componentes x e y del identificador de cada procesador.*

5.3.1.1 Análisis teórico del algoritmo para la factorización LU

En este apartado analizamos el coste temporal debido al cálculo y a las comunicaciones de un algoritmo paralelo que realiza la factorización LU (sin pivotamiento), siguiendo la filosofía de la librería ScaLAPACK [Blackford 97]. En esta librería paralela, las matrices están cíclicamente distribuidas por bloques en una malla de $p_r \times p_c$ procesadores; por razones de escalabilidad, sólo consideraremos topologías cuadradas ($p_r = p_c$, $p = p_r \times p_c = \sqrt{p} \times \sqrt{p}$).

En nuestro análisis el coste de comunicaciones seguirá el modelo lineal definido en apartados anteriores para el paso de mensajes en multicomputadores, es decir, $t = \sigma + M\tau$, donde σ es el coste de inicio del mensaje, independientemente de su longitud, τ es el coste incremental por unidad de longitud y M es la longitud del mensaje en palabras.

Analizaremos en primer lugar el coste de cálculo y de comunicaciones para la etapa k+1 del procedimiento de factorización y, a partir de éstos, obtendremos el coste global.

Sea $A \in \mathsf{R}^{n \times n}$ la matriz que vamos factorizar de la forma $A=LU$ y $r \times r$ el tamaño de bloque de resolución, siendo n múltiplo de r. El coste de los pasos diferentes de la etapa k+1 de la factorización es el siguiente [Dongarra 94]:

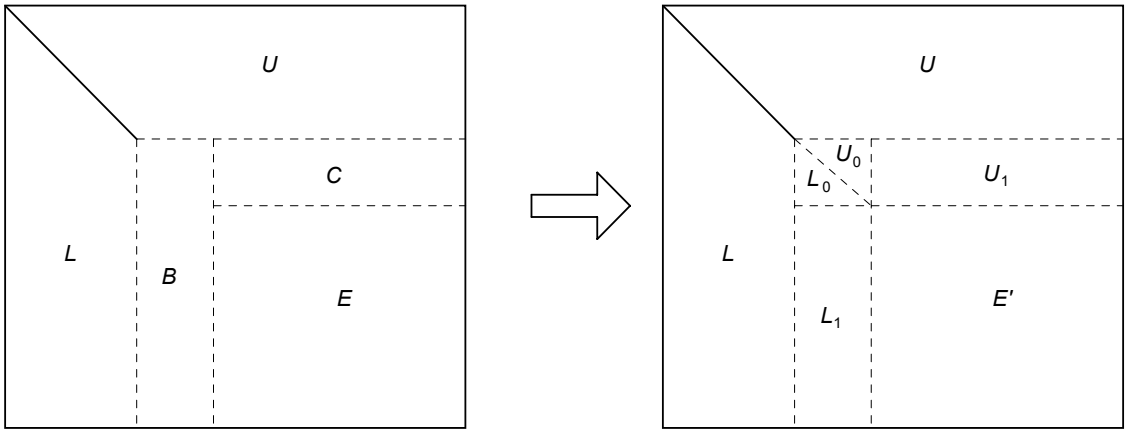


Figura 34. *Etapa k*+1 *de la factorización LU donde se muestra como son actualizados los bloques B, C y E. Las submatrices trapezoidales L y U han sido ya factorizadas en pasos anteriores. L posee kr columnas y U posee kr filas. En el (k*+1*)-ésimo paso, se calculan r nuevas columnas de L y r nuevas filas de U.*

1. La columna de procesadores que posee el bloque *B* (*k*+1)-*ésimo* (ver la Figura 34 (izquierda)) colaboran en su factorización. Esta factorización se divide en *r* subpasos, tantos como columnas del bloque *B*. Para la *i-ésima* columna del bloque,

1.a) Se difunde el elemento diagonal al resto de los procesadores de la misma columna. El coste en un hipercubo o un arbol binario es:

$$\log_2(\sqrt{p})\sigma$$

(asumimos que es la latencia la que domina la comunicación en este caso).

1.b) Se calculan los multiplicadores en paralelo entre todos los nodos. Coste aproximado:

$$\frac{n-kr-i}{\sqrt{p}}.$$

1.c) Se actualiza entre todos los procesadores el resto del bloque *B* mediante una actualización de rango 1. Coste aproximado:

$$\frac{n-kr-i}{\sqrt{p}}(r-i).$$

2. Se difunde el bloque factorizado de *L* (los bloques L_0 y L_1 en la Figura 34 (derecha)). Coste aproximado:

$$\log_2(\sqrt{p})\left(\sigma+\frac{n-kr}{\sqrt{p}}r\tau\right).$$

3. Se calcula U_1 resolviendo $L_0U_1=C$ en la fila de nodos que pose *C*. Coste aproximado:

$$\frac{n-(k+1)r}{\sqrt{p}}r$$

4. Se difunde U_1:

$$\log_2(\sqrt{p})\left(\sigma+\frac{n-(k+1)r}{\sqrt{p}}r\tau\right).$$

5. Se realiza la actualización de rango r de bloque E concurrentemente entre todos los procesadores. Coste aproximado:

$$2r\frac{\left(n-(k+1)r\right)^2}{p}.$$

Después de este paso, la siguiente columna de procesadores está lista para comenzar con la etapa (k+2).

Para obtener el coste total, las cantidades anteriores deben acumularse para todos los bloques factorizados ($k=1,\ldots,n/r$). Si consideramos que r es fijo y pequeño tendremos que la contribución total de cada uno de los pasos al coste global será:

$$\sum_k\sum_i(1.a)=n\log_2(\sqrt{p})\sigma.$$

$$\sum_k\sum_i(1.b)=\frac{n^2}{2\sqrt{p}}+O(n).$$

$$\sum_k\sum_i(1.c)=\frac{n^2r}{4\sqrt{p}}+O(n).$$

$$\sum_k(2)=\log_2(\sqrt{p})\left(\frac{n}{r}\sigma+\frac{n^2}{2\sqrt{p}}\tau+O(n)\tau\right).$$

$$\sum_k(3)=\frac{n^2}{2\sqrt{p}}+O(n).$$

$$\sum_k(4)=\log_2(\sqrt{p})\left(\frac{n}{r}\sigma+\frac{n^2}{2\sqrt{p}}\tau+O(n)\tau\right).$$

$$\sum_k(5)=\frac{2}{3}\frac{n^3}{p}+\frac{2}{\sqrt{p}}O(n^2).$$

Obteniéndose de este modo una aproximación superior del coste total del algoritmo

$$C_T = \frac{2}{3}\frac{n^3}{p} + \frac{2}{\sqrt{p}}O(n^2) + \log_2(\sqrt{p})O(n)\sigma + \log_2(\sqrt{p})\left(\frac{2}{\sqrt{p}}\right)O(n^2)\tau\,.$$

5.3.2 Algoritmo para resolver la ecuación de Lyapunov

A continuación presentamos la paralelización del algoritmo *SILE* para resolver la ecuación de Lyapunov en tiempo continuo que denominaremos *PSILE*. En este algoritmo, y en los algoritmos subsiguientes, se indica en cada paso la operación a realizar entre llaves, a modo de comentario, y a continuación la rutina o rutinas del LAPACK, BLAS o ScaLAPACK más importantes utilizadas para su implementación. En el caso de que un paso del algoritmo requiera la utilización de más de una rutina, se indica a la derecha de la rutina y entre llaves la operación que realiza.

Las rutinas más importantes de la librería paralela ScaLAPACK utilizadas en las implementación de los algoritmos basados en la función signo matricial son las siguientes:

- PDGETRF: Factorización LU con pivotamiento parcial.
- PDTRSM: Resolución de un sistema triangular lineal.
- PDGETRI: Inversión de una matriz a partir de los factores LU.
- PDGEMM: Producto de matrices.
- PDGEQPF: Factorización QR con pivotamiento de columnas.

Algoritmo 21 *[PSILE]*

1. $\{X = Q\}$

   PDLACPY

2. {mientras $\|A + I_n\|_1 > tol$ }

   2.1. $\{V = A\}$

   PDLACPY

   2.2 $\{V = LUP$, mediante la descomposición LU con pivotamiento parcial$\}$

   PDGETRF

   2.3. $\{\gamma = |\det(A)|^{1/n} = \prod_{k=1}^{n} |u_{kk}|^{1/n} = \exp\left(1/n \sum_{k=1}^{n} \log|u_{kk}|\right)\}$

   DGSUM2D $\{\exp\left(1/n \sum_{k=1}^{n} \log|u_{kk}|\right)\}$

   2.4. $\{W = XA^{-1}$, mediante eliminación progresiva y sustitución regresiva$\}$

   PDLACPY $\{W=X\}$

   PDLAPIV $\{W=WP^T\}$

PDTRSM $\{W = WU^{-1}\}$

PDTRSM $\{W = WL^{-1}\}$

2.5. $\{W = A^{-T}W\}$

PDLAPIV $\{W=PW\}$

PDTRSM $\{W = U^{-T}W\}$

PDTRSM $\{W = L^{-T}W\}$

2.6. $\{X = \frac{1}{2}\left(\frac{1}{\gamma}X + \gamma W\right)\}$

PDSCAL $\{X = \frac{1}{2\gamma}X\}$

PDAXPY $\{X = X + \frac{1}{2}\gamma W\}$

2.7. $\{A = \frac{1}{2}\left(\frac{1}{\gamma}A + \gamma A^{-1}\right)\}$

PDGETRI $\{V = A^{-1}\}$

PDSCAL $\{A = \frac{1}{2\gamma}A\}$

PDAXPY $\{A = A + \frac{1}{2}\gamma V\}$

{fin mientras}

3. $\{X = -\frac{1}{2}X\}$

PDSCAL

fin

5.3.3 Algoritmo para resolver la ecuación de Lyapunov para el Factor de Cholesky

A continuación presentamos el algoritmo paralelo *PSILCE*, basado en el algoritmo para resolver la ecuación de Lyapunov en tiempo continuo *SILCE* que obtiene directamente el factor de Cholesky de su solución.

Algoritmo 22 *[PSILCE]*

1. $\{Y = C,\ k = 0\}$

PDLACPY

2. {mientras $\|A + I_n\|_1 > tol$}

2.1. $\{V = A\}$

PDLACPY

2.2. $\{k = k + 1\}$

2.3. $\{V = LUP$, mediante la descomposición LU con pivotamiento parcial$\}$

PDGETRF

2.4. $\{\gamma = |\det(A)|^{1/n} = \prod_{k=1}^{n} |u_{kk}|^{1/n} = \exp\left(1/n \sum_{k=1}^{n} \log|u_{kk}| \right)\}$

DGSUM2D $\{\exp\left(1/n \sum_{k=1}^{n} \log|u_{kk}| \right)\}$

2.5. $\{W = YA^{-1}$, mediante eliminación progresiva y sustitución regresiva $\}$

PDLACPY $\{W=Y\}$

PDLAPIV $\{W=WP^T\}$

PDTRSM $\{W = WU^{-1}\}$

PDTRSM $\{W = WL^{-1}\}$

2.6. $\{A = \frac{1}{2}\left(1/\gamma A + \gamma A^{-1}\right)\}$

PDGETRI $\{V = A^{-1}\}$

PDSCAL $\{A = \frac{1}{2\gamma} A\}$

PDAXPY $\{A = A + \frac{1}{2}\gamma V\}$

2.7. $\{\gamma = \sqrt{\gamma}\}$

2.8. {si $k \le \lfloor \log_2 \frac{n}{m} \rfloor$ entonces}

2.8.1. $\{Y = \frac{1}{\sqrt{2}} \begin{bmatrix} 1/\gamma Y \\ \gamma YW \end{bmatrix}\}$

PDGEMM $\{W = \frac{\gamma}{\sqrt{2}} YW\}$

PDSCAL $\{Y = \frac{1}{\gamma\sqrt{2}} Y\}$

PDLACPY $\{Y = \begin{bmatrix} Y \\ W \end{bmatrix}\}$

{sino}

2.8.2. $\{\frac{1}{\sqrt{2}} \begin{bmatrix} 1/\gamma Y \\ \gamma YW \end{bmatrix} = U \begin{bmatrix} Y \\ 0 \end{bmatrix}$ mediante una factorización QR$\}$

PDGEMM $\{W = \frac{\gamma}{\sqrt{2}} YW\}$

PDSCAL $\{Y = \frac{1}{\gamma\sqrt{2}} Y\}$

PDLACPY $\{Y = \begin{bmatrix} Y \\ W \end{bmatrix}\}$

PDGEQPF $\{\begin{bmatrix} Y \\ W \end{bmatrix} = U \begin{bmatrix} Y \\ 0 \end{bmatrix}\}$

{fin si}

{fin mientras}

3. $\{ Y = -\frac{1}{\sqrt{2}} Y \}$

PDSCAL

fin

5.3.4 Algoritmo para resolver la ecuación de Lyapunov generalizada

A continuación presentamos el algoritmo paralelo *PSILGE* para resolver la ecuación de Lyapunov generalizada en tiempo continuo *SILGE.*

Algoritmo 23 *[PSILGE]*

1. $\{ V_E = E$, $E = L_E U_E P$, mediante la descomposición LU$\}$

PDLACPY

PDGETRF

2. $\{ \gamma_E = \left|\det(E)\right|^{1/n} = \prod_{k=1}^{n} \left|(u_E)_{kk}\right|^{1/n}$,$\}$

3. $\{ X = Q \}$

PDLACPY

4. {mientras $\left\|A + L_E\right\|_1 > tol \left\|L_E\right\|_1 \}$

4.1. $\{ V = A \}$

PDLACPY

4.2 $\{ V = LUP$, mediante la descomposición LU con pivotamiento parcial $\}$

PDGETRF

4.3. $\{ \gamma = \left|\det(A)\right|^{1/n} = \prod_{k=1}^{n} \left|u_{kk}\right|^{1/n} = \exp\left(1/n \sum_{k=1}^{n} \log\left|u_{kk}\right| \right) \}$

DGSUM2D $\{ \exp\left(1/n \sum_{k=1}^{n} \log\left|u_{kk}\right| \right) \}$

4.4. $\{ W = A^{-1} V_E$, mediante eliminación progresiva y sustitución regresiva $\}$

PDGETRS

4.5. $\{ \gamma = \gamma_A / \gamma_E \}$

4.6. $\{ A = \frac{1}{2}\left(1/\gamma V + \gamma V_E W \right) \}$

PDLACPY

PDGEMM

4.7. $\{ X = \frac{1}{2}\left(1/\gamma X + \gamma W^T X W \right) \}$

PDGEMM $\{ Q = \frac{1}{2}\gamma W^T X \}$

PDGEMM $\{ X = \frac{1}{2}\frac{1}{\gamma} X + QW \}$

fin mientras

5. $\{ V_E = LUP$, mediante la descomposición LU con pivotamiento parcial $\}$

PDGETRF

6. $\{ X = -\frac{1}{2} L^{-T} U^{-T} P X P^T U^{-1} L^{-1} \}$

PDLAPIV $\{X=XP^T\}$

PDTRSM $\{ X = XU^{-1} \}$

PDTRSM $\{ X = XL^{-1} \}$

PDLAPIV $\{X=PX\}$

PDTRSM $\{ X = U^{-T} X \}$

PDTRSM $\{ X = L^{-T} X \}$

PDSCAL $\{ X = -\frac{1}{2} X \}$

fin

5.3.5 Algoritmo para resolver las ecuaciones de Lyapunov acopladas

A continuación presentamos la paralelización del algoritmo *SILAE* para resolver ecuaciones de Lyapunov en tiempo continuo acopladas y que denominaremos *PSILAE*.

Algoritmo 24 *[PSILAE]*

1., $\{ X = Q$, $Y = P \}$

PDLACPY

PDLACPY

2. mientras $\|A + I_n\|_1 > tol$

2.1. $\{ V = A \}$

PDLACPY

2.2. $\{ A = LUP$, mediante la descomposición LU con pivotamiento parcial $\}$

PDGETRF

2.3. $\{ \gamma = |\det(A)|^{\frac{1}{n}} = \prod_{k=1}^{n} |u_{kk}|^{\frac{1}{n}} = \exp\left(\frac{1}{n} \sum_{k=1}^{n} \log|u_{kk}| \right) \}$

DGSUM2D $\{ \exp\left(\frac{1}{n} \sum_{k=1}^{n} \log|u_{kk}| \right) \}$

2.4a. $\{ W_X = XA^{-1}$, mediante eliminación progresiva y sustitución regresiva $\}$

PDLACPY $\{W_X = X\}$

PDLAPIV $\{W_X = W_X P^T\}$

PDTRSM $\{W_X = W_X U^{-1}\}$

PDTRSM $\{W_X = W_X L^{-1}\}$

2.4b. $\{W_Y = YA^{-1}$, mediante eliminación progresiva y sustitución regresiva $\}$

PDLACPY $\{W_Y = Y\}$

PDLAPIV $\{W_Y = W_Y P^T\}$

PDTRSM $\{W_Y = W_Y U^{-1}\}$

PDTRSM $\{W_Y = W_Y L^{-1}\}$

2.5a. $\{W_X = A^{-T} W_X\}$

PDLAPIV $\{W_X = PW_X\}$

PDTRSM $\{W_X = U^{-T} W_X\}$

PDTRSM $\{W_X = L^{-T} W_X\}$

2.5b. $\{W_Y = A^{-T} W_Y\}$

PDLAPIV $\{W_Y = PW_Y\}$

PDTRSM $\{W_Y = U^{-T} W_Y\}$

PDTRSM $\{W_Y = L^{-T} W_Y\}$

2.6a. $\{X = \frac{1}{2}\left(\frac{1}{\gamma} X + \gamma W_X\right)\}$

PDSCAL $\{X = \frac{1}{2\gamma} X\}$

PDAXPY $\{X = X + \frac{1}{2}\gamma W_X\}$

2.6b. $\{Y = \frac{1}{2}\left(\frac{1}{\gamma} Y + \gamma W_Y\right)\}$

PDSCAL $\{Y = \frac{1}{2\gamma} Y\}$

PDAXPY $\{Y = Y + \frac{1}{2}\gamma W_Y\}$

2.7. $\{A = \frac{1}{2}\left(\frac{1}{\gamma} A + \gamma A^{-1}\right)\}$

PDGETRI $\{V = A^{-1}\}$

PDSCAL $\{A = \frac{1}{2\gamma} A\}$

PDAXPY $\{A = A + \frac{1}{2}\gamma V\}$

fin mientras

3. $X = -\frac{1}{2} PXP^T$, $Y = -\frac{1}{2} PYP^T$

PDSCAL

PDSCAL

fin

5.3.6 Análisis teórico de los algoritmos

En este apartado analizamos el coste teórico de los algoritmos basados en la función signo matricial (apartados 5.3.3 a 5.3.5).

En la Tabla 2 presentamos los costes aritméticos y de comunicaciones aproximados para las rutinas de ScaLAPACK utilizadas en nuestros algoritmos. A partir de ellos obtendremos los costes asociados a los diversos algoritmos paralelos desarrollados utilizando el método de la función signo matricial.

En la Tabla 3 mostramos las rutinas más importantes utilizadas en las implementación de los diversos algoritmos paralelos desarrollados y el número de veces que utilizan estas rutinas. En esta tabla, "*iter*" representa el número de iteraciones necesario para que la correspondiente iteración converja.

Rutina	Coste aritmético Coste $\times \frac{n^3}{p}$	Coste de comunicaciones	
		Latencia $\times \sigma$	(Ancho de Banda)$^{-1} \times \frac{n^2}{\sqrt{p}} \tau$
PDGETRF	$\frac{2}{3}$	$(6 + \log_2 p)n$	$(3 + \frac{1}{4}\log_2 p)$
PDTRSM	1	n	$(1 + \frac{3}{4}\log_2 p)$
PDGETRI	$\frac{4}{3}$	$2n$	$(2 + \frac{3}{2}\log_2 p)$
PDGEMM	2	$(1 + \frac{1}{2}\log_2 p)\sqrt{p}$	$(1 + \frac{1}{2}\log_2 p)$
PDGEQPF	$\frac{4}{3}$	$3n \log p$	$\frac{4}{3}\log p$

Tabla 2. *Costes aritméticos y de comunicaciones para las rutinas paralelas del ScaLAPACK utilizadas en la implementación de los algoritmos.*

Algoritmo Paralelo	PDGETRF	PDTRSM	PDGETRI	PDGEMM	PDGEQPF
PSILE	iter	$4 \times$ iter	iter	–	–
PSILCE	iter	$2 \times$ iter	iter	iter	iter$^{(*)}$
PSILGE	2 + iter	$4 + 2 \times$ iter	–	$3 \times$ iter	–
PSILAE	iter	$8 \times$ iter	iter	–	–

Tabla 3. *Número de llamadas a las distintas rutinas del ScaLAPACK utilizadas en los algoritmos paralelos diseñados. La marca $^{(*)}$ indica que sólo contribuirá al coste cuando se cumpla la condición expresada en el algoritmo PSILCE ($k \leq \lfloor \log_2 \frac{n}{m} \rfloor$).*

Algoritmo Paralelo	Coste aritmético $\times \frac{n^3}{p}$
PSILE	$6 \times \text{iter}$
PSILCE	$\frac{22}{3} \times \text{iter}$
PSILGE	$\frac{16}{3} \times \text{iter}$
PSILAE	$10 \times \text{iter}$

Tabla 4. *Coste aritmético de los algoritmos paralelos desarrollados basados en la función signo matricial.*

A partir de las Tablas 4 y 5 podemos construir el coste del modelo teórico de los algoritmos paralelos

Algoritmo Paralelo	Latencia $\times \sigma$	(Ancho de Banda)$^{-1} \times \frac{n^2}{\sqrt{p}} \tau$
PSILE	$(12 + \log_2 p)n \times iter$	$(9 + \frac{19}{4} \log_2 p) \times iter$
PSILCE	$(1 + \frac{1}{2} \log_2 p)\sqrt{p} \times iter$ + $(10 + 4 \log_2 p)n \times iter$	$(1 + \frac{1}{2} \log_2 p) \times iter$ + $(7 + \frac{55}{12} \log_2 p) \times iter$
PSILGE	$3(1 + \frac{1}{2} \log_2 p)\sqrt{p} \times iter$ + $(8 + \log_2 p)(2 + iter)n$	$3(1 + \frac{1}{2} \log_2 p) \times iter$ + $(8 + \frac{7}{4} \log_2 p)(2 + iter)$
PSILAE	$(16 + \log_2 p)n \times iter$	$(13 + \frac{31}{4} \log_2 p) \times iter$

Tabla 5. *Coste de comunicaciones de los algoritmos paralelos desarrollados basados en la función signo matricial.*

5.4. Conclusiones

En este capítulo se han descrito diversos algoritmos paralelos para la resolución de las ecuaciones de Lyapunov y se ha realizado el análisis teórico de costes de los algoritmos más representativos. En particular se han desarrollado algoritmos paralelos basados en los métodos de Hammarling y la función signo matricial.

Basados en el método de Hammarling se han diseñado nuevos algoritmos paralelos para resolver las ecuaciones de Lyapunov en tiempo discreto para multiprocesadores con memoria compartida utilizando técnicas de frente de onda con grano de resolución fino (*PHWF*). Estos algoritmos resultan especialmente apropiados para multiprocesadores con procesadores superescalares. A fin de aprovechar las prestaciones adicionales que proporcionan los sistemas con unidades vectoriales se han diseñado algoritmos paralelos de grano medio (*PHWM*). Se han propuesto en ambos casos mejoras de estos algoritmos basados en la resolución por bloques de columnas (*PBHWF, PBHWM*) para aprovechar la localidad de las operaciones y optimizar los accesos a memoria. Otra mejora que se ha introducido en estos algoritmos es el solapamiento entre la resolución del fin de un bloque de columnas y el principio del siguiente (*PBSHWF, PBSHWM*), que permite reducir las pérdidas de eficiencia producidas por procesadores ociosos. Como el tamaño de las matrices del problema se reduce en cada etapa se ha propuesto el diseño de algoritmos que adapten el grano de resolución basados en la combinación de los algoritmos con granos de resolución fino y medio.

Así mismo, basados en el método de Hammarling, se han desarrollado nuevos algoritmos paralelos para sistemas multiprocesadores con memoria distribuida que utilizan paso de mensajes para las comunicaciones. En estos algoritmos se han aplicado y extendido las técnicas utilizadas para la resolución de sistemas triangulares lineales. Así, se han diseñado algoritmos paralelos frente de onda y cíclicos con grano de resolución fino (*PDHWF, PHCF, PHCF2*), medio (*PDHWM, PHCM, PHCMR*) y grueso (*PDHWB,PHCB*). Para reducir los tiempos de espera presentes en los algoritmos cíclicos se han propuesto mejoras basadas en la optimización del envío de los mensajes (*PHCFU, PHCFG*). Para aprovechar las ventajas de los algoritmos frente de onda y cíclicos, se han propuesto algoritmos basados en la combinación de las dos técnicas (*PHCWF, PHCWFU, PHCWFG, PHCWFN*). Además, siguiendo el razonamiento utilizado en memoria compartida, se han desarrollado algoritmos que adaptan dinámicamente del grano de resolución a medida que se va resolviendo el problema (*PADHWB, PAHCWFU*).

En la función signo matricial se han diseñado nuevos algoritmos paralelos para resolver la ecuación de Lyapunov en tiempo continuo (*PSILE*). También se han desarrollado algoritmos para obtener directamente el factor de Cholesky de esta ecuación (*PSILCE*), ya que en muchos problemas de control es más útil el factor de Cholesky de la solución que la solución en sí. Como generalización de los algoritmos anteriores hemos presentado algoritmos paralelos para resolver la ecuación de Lyapunov generalizada (*PSILGE*) basados en la iteración de Newton generalizada. Por último, se han diseñado algoritmos para la resolución de ecuaciones de Lyapunov acopladas (*PSILAE*). Estos algoritmos se han implementado utilizando la librería paralela ScaLAPACK.

Este capítulo se enlaza con el análisis experimental en el siguiente capítulo de las implementaciones de los algoritmos paralelos descritos, sobre diversas arquitecturas de computadores paralelos.

Capítulo 6

Análisis de los resultados experimentales

6.1 Introducción

En este capítulo se analizan los resultados experimentales obtenidos para las implementaciones de los algoritmos desarrollados en diversos computadores paralelos. Aunque se han realizado pruebas de todos los algoritmos propuestos, sólo mostraremos las versiones más elaboradas de cada uno de ellos y aquellos que, por sus características, muestren comportamientos que nos ayuden a un mejor conocimiento de la influencia de ciertas técnicas en los resultados obtenidos.

La sección 6.2 de este capítulo está dedicada al análisis de los algoritmos paralelos desarrollados utilizando el método de Hammarling. Así, en el apartado 6.2.1 se presentan los resultados sobre multiprocesadores con memoria compartida utilizando el paradigma de programación de paralelismo de control. En el apartado 6.2.2 analizamos los resultados obtenidos para los algoritmos de frente de onda y cíclicos implementados para multicomputadores y multiprocesadores que utilizan paso de mensajes. En la sección 6.3 presentamos los resultados obtenidos en los algoritmos implementados basados en el método de la función signo matricial.

El análisis experimental de los algoritmos incide fundamentalmente en las prestaciones. También se realiza un análisis de la precisión en el caso de los algoritmos basados en la función signo matricial; pero no en el caso de los algoritmos basados en el método de Hammarling, ya que se trata de métodos numéricamente estables.

Los algoritmos paralelos se evalúan conforme a los *items* presentados en el apartado 4.2.3 del capítulo 4 (tiempo de ejecución, eficiencia, etc.). Dependiendo del interés o el tipo de análisis que realicemos en cada caso utilizaremos el método de evaluación que más ilustre sus características. Los algoritmos paralelos se han comparado con el mejor algoritmo serie en lugar de con el algoritmo mismo paralelo utilizando 1 procesador. Cuando se evalúan las prestaciones de un algoritmo de esta última forma sólo se tiene en cuenta la paralelización intrínseca de éste, pero no el incremento de velocidad respecto del algoritmo serie. Esto da lugar, en muchas ocasiones, a falsas expectativas de eficacia para el algoritmo paralelo. También es bastante frecuente evaluar un algoritmo en función del aprovechamiento del computador. En estos casos se muestra la cantidad de operaciones en coma

flotante por segundo que consigue el algoritmo, midiendo el tiempo que tarda en ejecutarse y comparándolo con el coste teórico del problema.

En cuanto al tipo de datos utilizado para las pruebas, las características de las matrices utilizadas no influyen en el coste temporal de los algoritmos basados en el método de Hammarling, debido a que se trata de un método directo. Por ello, se han utilizado matrices generadas aleatoriamente, controlando únicamente las condiciones de solubilidad de las ecuaciones.

Como los algoritmos basados en el método de la función signo matricial son inherentemente iterativos y no numéricamente estables, la convergencia de los algoritmos será más o menos rápida dependiendo del tipo de matrices con las que se trabaje. Por tanto, en los experimentos hemos elegido varios tipos de matrices para analizar el tiempo de resolución de los algoritmos implementados.

El análisis de los algoritmos paralelos basados en el método de Hammarling para multiprocesadores con memoria compartida del apartado 6.2.1 se ha desarrollado en el Alliant FX/80, los algoritmos paralelos basados en este método para multiprocesadores con memoria distribuida y/o con paso de mensajes del apartado 6.2.2 se han evaluado en dos computadores muy distintos, en el PowerChallenge de Silicon Graphics (PCh) y en el Cray T3D (T3D). Finalmente, los algoritmos paralelos de la sección 6.3 basados en la función signo matricial se han desarrollado sobre el multicomputador IBM SP2 (SP2).

6.2 Método de Hammarling

En esta sección estudiaremos las prestaciones de los algoritmos paralelos basados en el método de Hammarling obtenidas en distintos computadores paralelos. En concreto, en el apartado 6.2.1 analizaremos los resultados obtenidos en multiprocesadores con memoria compartida y en el apartado 6.2.2 los resultados obtenidos en multiprocesadores con memoria distribuida (multicomputadores). En los apartados 6.2.2.1 analizaremos los algoritmos paralelos por bloques y frente de onda de antidiagonales y en el apartado 6.2.2.2 los algoritmos paralelos cíclicos y sus mejoras.

Los algoritmos paralelos evaluados en esta sección resuelven la ecuación reducida de Lyapunov en tiempo discreto suponiendo que la matriz de estados es triangular inferior (solo valores propios reales). En el caso de que existan valores reales y complejos las prestaciones de estos algoritmos no diferirán significativamente.

6.2.1 Multiprocesadores con memoria compartida

Los algoritmos paralelos de este apartado se han implementado en un multiprocesador con memoria compartida, el Alliant FX/80, utilizando el lenguaje de programación Fortran 77 con extensiones del lenguaje propias de esta máquina para indicar el tipo paralelismo a aplicar a cada parte de los programas. Todos los algoritmos se han compilado con las opciones de compilación *–Og*. Las opciones *–v* (vectorial) y *–c* (concurrente) se han utilizado para activar el uso de las unidades vectoriales y el paralelismo de los algoritmos, respectivamente.

Las matrices utilizadas para las pruebas se han generado de forma aleatoria y los resultados de los algoritmos paralelos se han comprobado con los resultados del algoritmo serie de Hammarling (*SH*). Los tamaños de los bloques de columnas (*tbcol*) utilizados en los algoritmos presentados en este apartado han sido de 8, 16, 32 y 64. Los tamaños de los vectores, *t*, en el caso de los algoritmos de

grano medio, varían según los mismos valores de *tbcol*. Los tamaños de las matrices en las pruebas realizadas, *n*, van desde 20 hasta 500.

En este apartado sólo se muestran las prestaciones obtenidas para los algoritmos paralelos que utilizan la resolución por bloques de columnas y solapamiento, pues es en ellos donde se obtienen los mejores resultados.

La Figura 35 nos muestra los tiempos de ejecución obtenidos para el algoritmo serie *SH* y para el algoritmo paralelo de grano fino *PBSHWF* compilado con la opción *–c* y ejecutado sobre 8 procesadores. En ambos casos se inhiben las optimizaciones vectoriales. Como puede observarse los tiempos de ejecución para el algoritmo *PBSHWF* no dependen en la práctica del tamaño de los bloques de columnas *tbcol*.

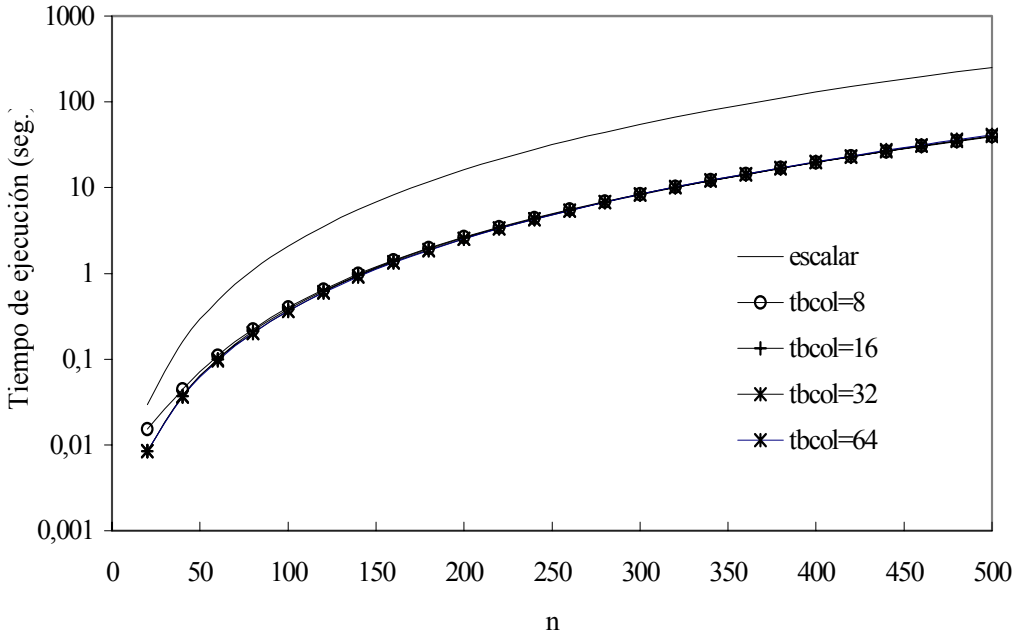


Figura 35. *Tiempos de ejecución obtenidos para el algoritmo serie de Hammarling SH (escalar) y para el algoritmo paralelo de grano fino PBSHWF utilizando 8 procesadores y diferentes tamaños de bloques de columnas (tbcol). No se utilizan las unidades vectoriales.*

La Figura 36 muestra la eficiencia del algoritmo paralelo *PBSHWF* utilizando 8 procesadores obtenida a partir de los resultados de tiempos de ejecución de la gráfica anterior. Como se puede apreciar, se consiguen eficiencias superiores a 0,8. Los resultados son similares para todos los tamaños de bloques columna. La eficiencia del algoritmo crece rápidamente, alcanzándose el 0,5 de eficiencia para n=50 y ésta se estabiliza alrededor del 0,8 a partir de n=300. Al seguir aumentando el tamaño del problema no existe una reducción significativa de la eficiencia. Este comportamiento es debido a un peor uso de la antememoria, más aún cuando, en el caso del Alliant, ésta es compartida por todos los procesadores.

Para valores de n pequeños podemos observar que la eficiencia del algoritmo *PBSHWF* es bastante reducida. Este comportamiento aparece en casi todos los algoritmos paralelos y se debe a que, en estos casos, el sobrecoste asociado a la sincronización de los procesadores o a la latencia de las comunicaciones tiene un valor muy alto en relación con el coste de resolución del problema.

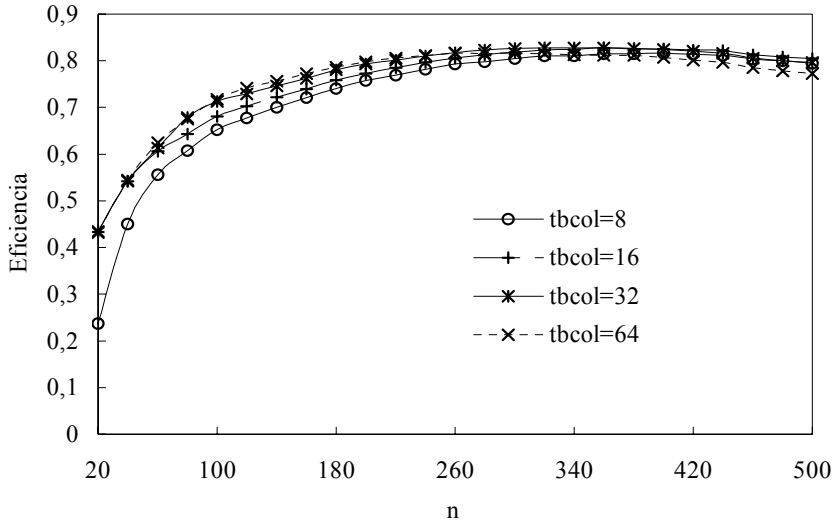


Figura 36. *Eficiencia obtenida para el algoritmo paralelo de grano fino PBSHWF utilizando 8 procesadores y diferentes tamaños de bloques de columnas (tbcol) respecto del algoritmo serie de Hammarling SH (escalar). No se utilizan las unidades vectoriales.*

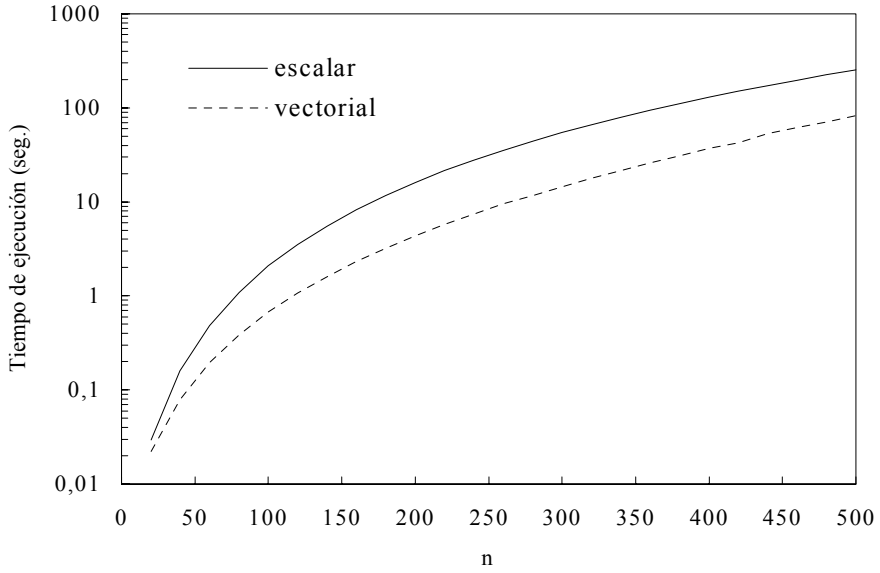


Figura 37. *Tiempos de ejecución obtenidos para el algoritmo SH sin utilizar las unidades vectoriales (escalar) y haciendo uso de ellas (vectorial).*

La vectorización es una técnica que, sobre procesadores vectoriales como los del Alliant FX/80, por si sola incrementa la velocidad de ejecución de un algoritmo numérico de forma muy significativa. Por ejemplo, el algoritmo serie *SH* con optimización vectorial consigue un incremento de velocidad que reduce notablemente el tiempo de ejecución (ver Figura 37) y alcanza una aceleración de más de 3,5 para algunos órdenes de *n*, como se puede ver en la Figura 38.

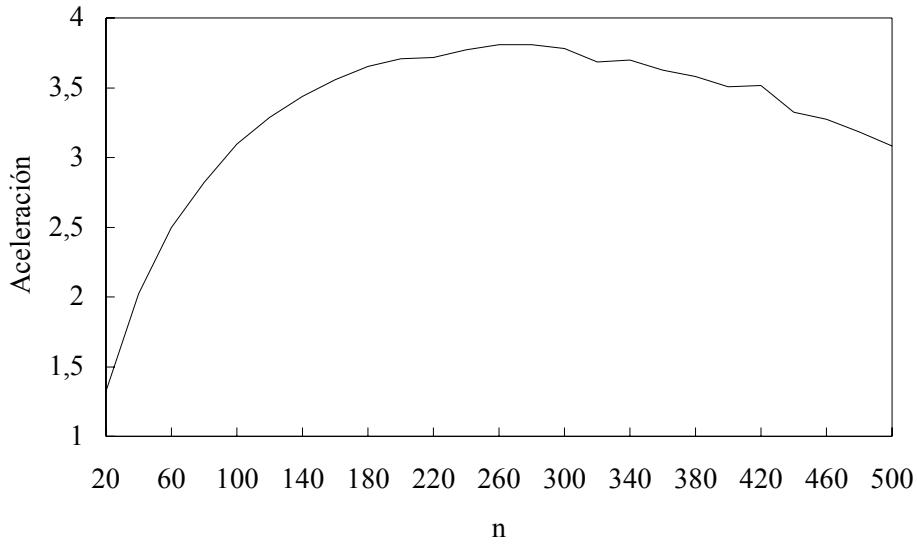


Figura 38. *Aceleración obtenida para el algoritmo secuencial SH con optimización vectorial (-v) comparado con el mismo algoritmo con estas optimizaciones inhibidas.*

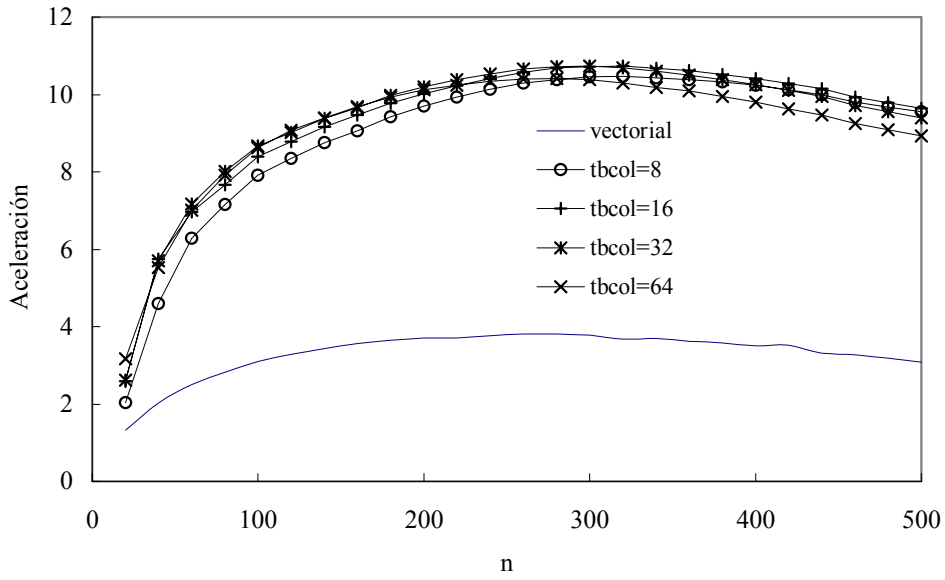


Figura 39. *Aceleración del algoritmo SH (vectorial) y el algoritmo paralelo PBSHWF para 8 procesadores y diferentes valores de tbcol, ambos con vectorización, comparados con el algoritmo SH sin utilizar la optimización vectorial.*

Puede observarse como la aceleración decrece para valores de *n* elevados. Las razones de ello están asociadas con el alto rendimiento que se obtiene de las unidades vectoriales cuando los datos del problema se encuentran casi todos contenidos en la antememoria. Cuando aumenta el tamaño del problema este rendimiento se reduce, sobre todo cuando no se han realizado modificaciones en el algoritmo para aprovechar la localidad de los datos. Téngase en cuenta que la vectorización del algoritmo *SH* se realiza de forma automática, dejando al compilador que tome la iniciativa respecto de qué parte del código tiene que vectorizar, sin intervención alguna del programador.

En la Figura 39 vemos como este comportamiento se repite para el algoritmo paralelo de grano fino *PBSHWF* cuando utilizamos las optimizaciones vectoriales. Se obtienen aceleraciones crecientes respecto del algoritmo *SH* sin utilizar las unidades vectoriales (escalar) hasta el tamaño *n*=300, pero a partir de ese tamaño la aceleración decrece a un ritmo similar a la del algoritmo *SH* con vectorización. Como consecuencia de ello, la eficiencia del algoritmo *PBSHWF* respecto del algoritmo *SH*, ambos con vectorización, alcanza un valor alrededor de 0,35 (ver Figura 40). Ésta incluso aumenta para valores de *n* elevados. Sin embargo, se obtiene una eficiencia que no supera el 40%, aunque bastante sostenida para toda la gama de ordenes de *n*.

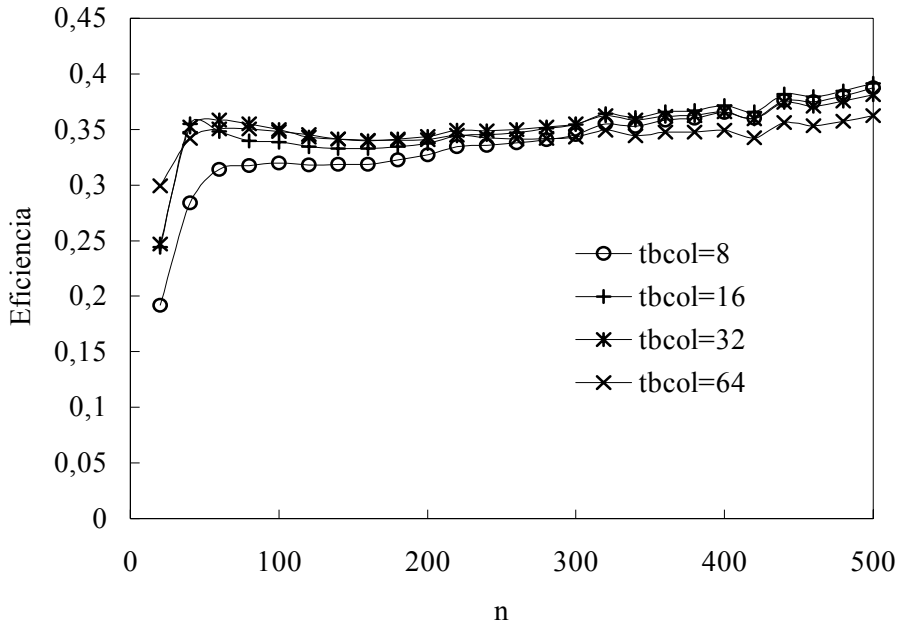


Figura 40. *Eficiencia para el algoritmo paralelo de grano fino PBSHWF para 8 procesadores y diferentes valores de tbcol comparado con el algoritmo serie SH. En ambos algoritmos se utilizan las optimizaciones vectoriales.*

Para obtener mejores prestaciones hay que utilizar un algoritmo orientado a vectores como el algoritmo paralelo por bloques de columnas y solapamiento, *PBSHWM*. En la Figura 41 se observa como el tiempo de ejecución obtenido para el algoritmo *PBSHWM*, sobre 8 procesadores es menor que el obtenido para el algoritmo *SH*, ambos con vectorización. El tamaño de los subvectores, *t*, utilizado en la resolución es igual al de los bloques de columnas, *tbcol*. Este algoritmo no presenta un comportamiento uniforme para los distintos tamaños de bloques columna como ocurría en el algoritmo paralelo *PBSHWF*. Además, en las curvas de tiempos de ejecución del algoritmo *PBSHWM* aparecen unos pequeños dientes de sierra. Este comportamiento es habitual en los programas diseñados para una resolución por vectores o por bloques ejecutados sobre computadores con unidades vectoriales. La forma de los dientes de sierra depende del tamaño del vector o bloque de resolución elegido y del tamaño de las unidades vectoriales.

En la Figura 41 observamos que los peores resultados se dan para tamaños de vector extremos (8 y 64), mientras que para los tamaños 16 y 32 se obtienen mejores resultados. Este comportamiento se debe a que cuando se aumenta el grano de resolución en un algoritmo se reduce el paralelismo que se puede obtener en la resolución del problema. Por otra parte, cuando se utilizan unidades vectoriales, se obtienen mayores prestaciones de éstas cuanto más se aproxima el grano de resolución al tamaño de la unidad vectorial (en el caso del Alliant éste es de 32 elementos). Esto provoca que cuanto menor sea

el tamaño de vector elegido, menor es el aprovechamiento de la unidad vectorial, pero mayor es el paralelismo y la localidad de los accesos a memoria. Lo contrario ocurrirá cuanto mayor sea el tamaño del vector si éste es múltiplo del número de los registros de la unidad vectorial. Como consecuencia de ello las mejores prestaciones se darán cuando exista un punto intermedio entre los dos factores expuestos.

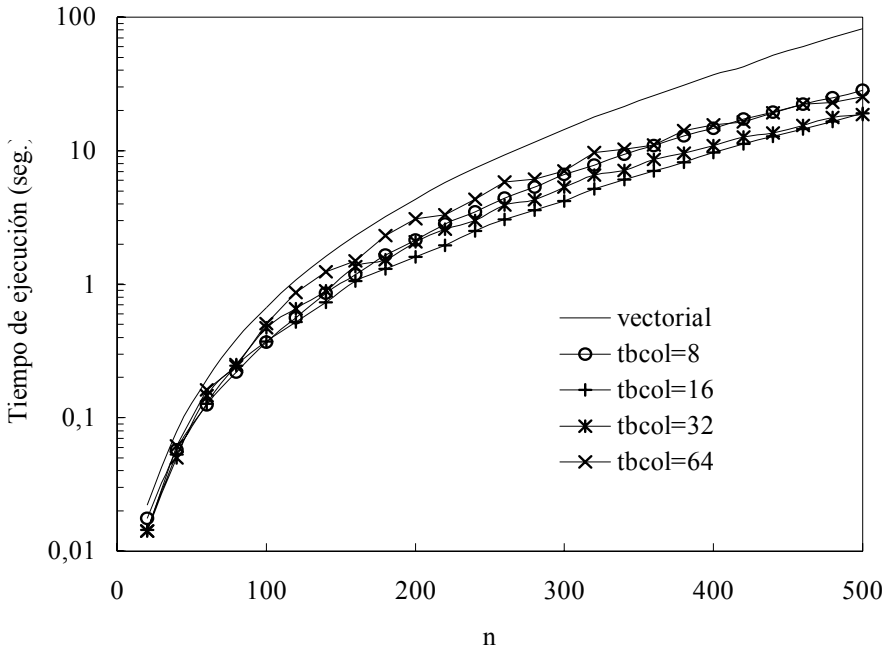


Figura 41. *Tiempos de ejecución obtenidos para el algoritmo paralelo de grano medio PBSHWM utilizando 8 procesadores y diferentes tamaños de bloques de columnas (tbcol). Se utilizan optimizaciones vectoriales.*

En la Figura 42 podemos observar como las mejores eficiencias para el algoritmo *PBSHWM* sobre 8 procesadores se dan para los tamaños bloques de columna 16 y 32. Las prestaciones del algoritmo son bajas para tamaños del problema, *n*, pequeños, incluso menores que para el algoritmo *PBSHWF*. Sin embargo, la eficiencia no se estabiliza en un valor rápidamente como en el algoritmo *PBSHWF*, sino se incrementa al crecer *n* para el rango de valores estudiado. En particular, se consiguen eficiencias de 0,56 para 8 procesadores y n=500.

La utilización de las rutinas del BLAS en los programas de grano medio no incrementa de forma sustancial la ganancia de éstos. La causa de ello es que únicamente podemos utilizar los niveles 1 y 2 de esta librería, es decir, operaciones del tipo escalar/vector y matriz/vector. Por otra parte, el aprovechamiento de estas rutinas se da para tamaños de vectores y matrices grandes, perdiéndose eficiencia cuando éstos son pequeños. Las prestaciones son similares a las vistas en la Figura 42, con la salvedad del incremento de prestaciones para tamaños de vector de 64 elementos. Las mejores prestaciones se siguen dando para tamaños de vector de 16 y 32.

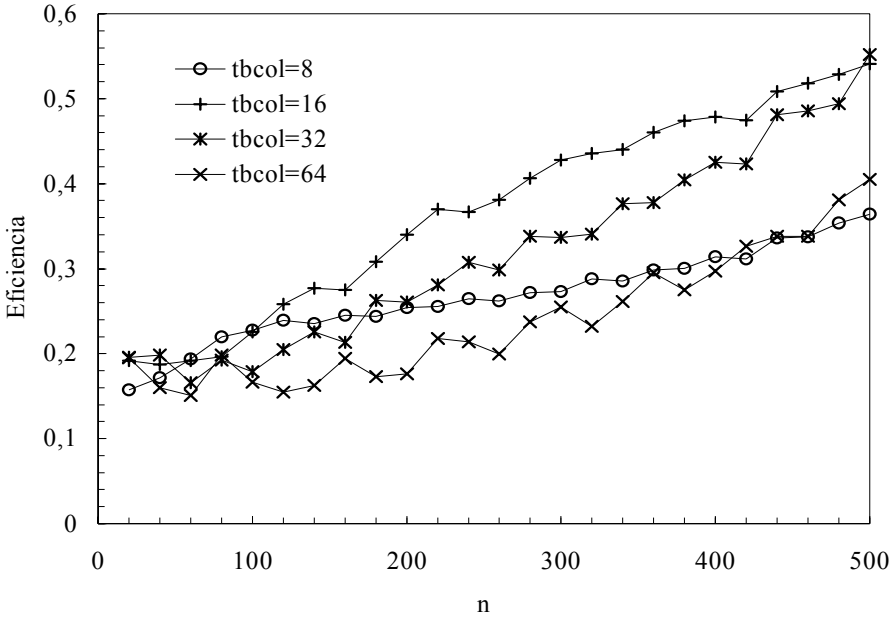


Figura 42. *Eficiencias obtenidas para el algoritmo para el algoritmo paralelo de grano medio PBSHWM utilizando 8 procesadores y diferentes tamaños de bloques de columnas (tbcol). Se utilizan las optimizaciones vectoriales*

Podemos observar, sin embargo, que para valores de *n* pequeños no obtenemos buenas prestaciones en el algoritmo *PBSHWM* y que éstas son incluso menores que para el algoritmo *PBSHWF,* ambos con vectorización. Esto es debido al grano de resolución mayor que estamos utilizando en el algoritmo *PBSHWM*, cuya consecuencia es esta pérdida de eficiencia para tamaños de problema pequeños. Estas pérdidas de eficiencia también se producen para tamaños de *n* grandes en las últimas etapas de resolución de la ecuación, donde las matrices con las que se trabaja son de pequeño tamaño. Al utilizar solapamiento se evitan estas pérdidas al final del cálculo de cada bloque de columnas.

Para obtener mejores prestaciones de nuestros programas para cualquier tamaño, proponemos un algoritmo paralelo que llamaremos *PBSHWC* y que utilizará la resolución de grano medio en una parte de la resolución del problema y la resolución de grano fino en otra. De esta forma, si el orden de las matrices de la ecuación de Lyapunov, *n,* es mayor que un valor límite n_l, ésta se resolverá utilizando el algoritmo *PBSHWM*. Cuando el tamaño de las matrices de la ecuación en una etapa del problema, *n'*, cumpla que $n' < n_l$, entonces se pasará a resolver ésta utilizando el algoritmo de grano fino *PBSHWF*. La elección de n_l se obtiene experimentalmente y coincide con el valor donde las prestaciones del algoritmo *PBSHWM* empiezan a resultar peores que las del algoritmo *PBSHWM* (ambos con vectorización).

Como podemos ver en la Figura 43, se obtienen mejores tiempos en el algoritmo *PBSHWC* para tamaños de *n* pequeños, obteniendo ligeras mejoras para tamaños de problema mayores. Esto es debido al poco peso del coste de las últimas etapas en el coste global del problema.

En la Figura 44 se observa una mejora de las prestaciones para el algoritmo *PBSHWC* respecto del algoritmo serie de Hammarling *SH*, obteniéndose eficiencias por encima de 0,38 para casi toda la gama de ordenes de *n* y para aquellos tamaños de bloques de columnas donde se obtienen los mejores resultados, 16 y 32. Para n= 500, se alcanza una eficiencia superior al 0,56.

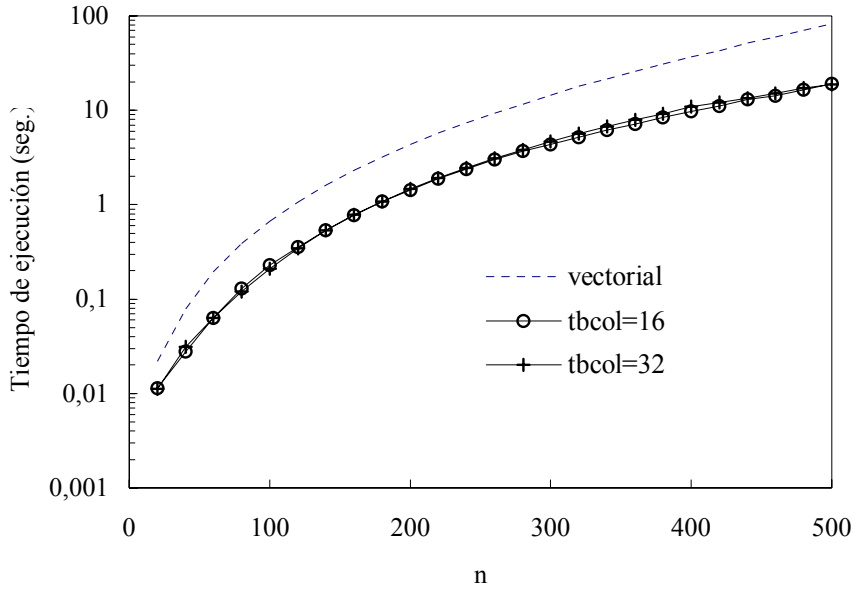


Figura 43. *Tiempos de ejecución obtenidos para el algoritmo serie de Hammarling SH y para el algoritmo paralelo de grano combinado PBSHWC utilizando 8 procesadores y bloques de columnas de tamaños 16 y 32. En ambos casos se utilizan las optimizaciones vectoriales.*

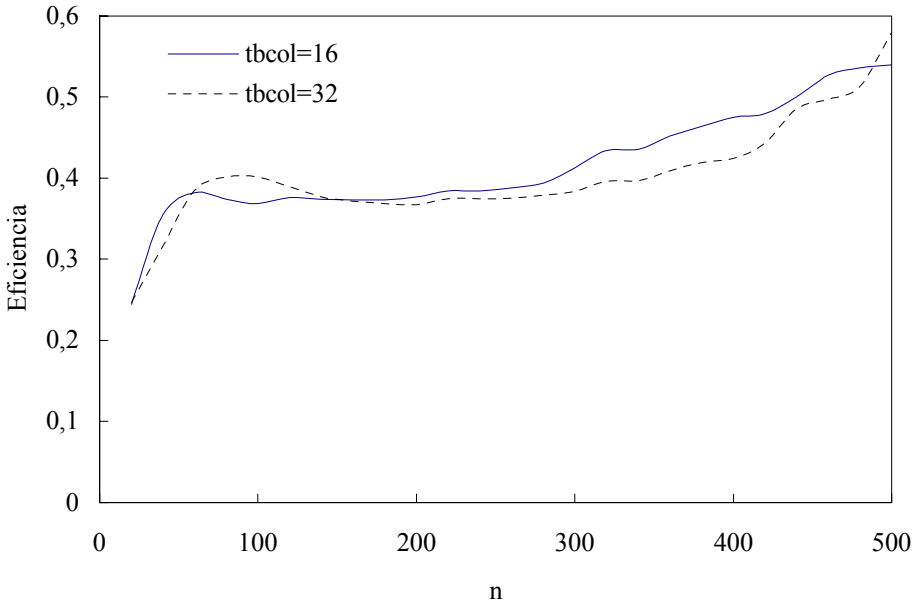


Figura 44. *Eficiencia obtenida para el algoritmo paralelo combinado PBSHWC respecto del algoritmo serie de Hammarling SH, utilizando 8 procesadores y bloques de columnas de tamaños 16 y 32. En ambos casos se utilizan las optimizaciones vectoriales.*

6.2.2 Multiprocesadores con memoria distribuida

Los algoritmos para la resolución de la ecuación de Lyapunov sobre multiprocesadores mediante el método de Hammarling se han implementado en C y se ha utilizado la librería PVM para las comunicaciones entre los procesadores. En el caso del PCh las comunicaciones se implementan sobre la memoria central a través de una red de tipo bus, mientras que en el T3D las comunicaciones se realizan mediante una red de interconexión fija de topología toroidal. Se han utilizado en los dos computadores optimizaciones de nivel O3 para todas las implementaciones, dado que este nivel es el que más optimiza el código. Las implementaciones paralelas se han comparado con un algoritmo secuencial orientado por bloques; en particular bloques de 16×32 para el PCh y bloques 2×1 para el T3D. Como se observa en la Figura 45, esta versión secuencial por bloques ofrece mejores prestaciones que la puramente secuencial, obteniéndose para tamaños de problema elevados (n>1000), mejoras superiores al 100% en el PCh y del 20% para el T3D. Estas mejoras son muy importantes para el caso del PCh, debido al gran tamaño de la antememoria de los procesadores.

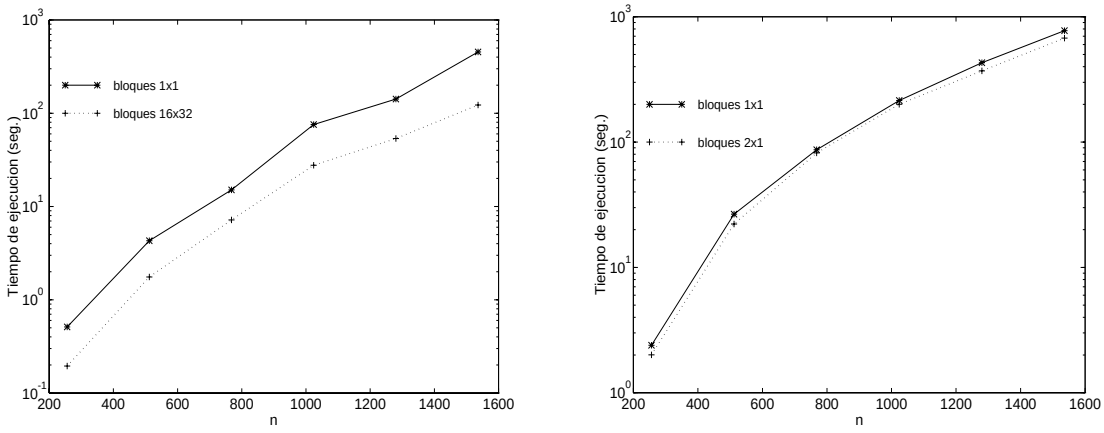


Figura 45. *Tiempos de ejecución para el algoritmo secuencial de Hammarling estándar (1×1) y sus versiones por bloques de 16×32 en el PCh (izquierda) y de 2×1 en el T3D (derecha).*

6.2.2.1 Algoritmos de frente de onda

En varios experimentos previos hemos podido comprobar que los resultados de los algoritmos *PDHWF* y *PDHWM* no resultan plenamente satisfactorios debido al pequeño grano de computación que utilizan. En consecuencia, se han evaluado los algoritmos de grano grueso *PDHWB*, donde los bloques son tamaño de $qp \times q$, siendo p el número de procesadores y q un entero positivo. Para el caso en que q=1, el comportamiento se corresponde con el del algoritmo *PDHWM*. Al elegir estos tamaños de bloque conseguimos que el resultado final se encuentre distribuido cíclicamente de forma natural entre todos los procesadores y optimizamos el tiempo dedicado a las comunicaciones durante el proceso de cálculo y la recogida de los resultados.

En las Figuras 46 y 47 se muestra el comportamiento del algoritmo *PDHWB* para distintos valores de q y n en el PCh utilizando 4 y 8 procesadores, respectivamente. Observamos que el algoritmo consigue muy buenas prestaciones a partir de tamaños de problemas de orden elevado, siendo baja para problemas de orden reducido. Esto está relacionado con la existencia de antememorias en este computador y el coste de la latencia de los mensajes. Cuando el tamaño del problema es grande, para el caso secuencial no caben los datos del problema en la antememoria por lo que las prestaciones decrecen. Es entonces cuando se obtiene un incremento de la eficiencia para los

algoritmos paralelos, gracias a la distribución de las matrices entre los procesadores. Otra circunstancia que limita las prestaciones de los algoritmos paralelos en el PCh es la existencia de una topología de bus para la red de comunicaciones, lo que supone un cuello de botella para la circulación simultánea de varios mensajes. Este efecto se verá acrecentado cuando mayor sea el número de procesadores.

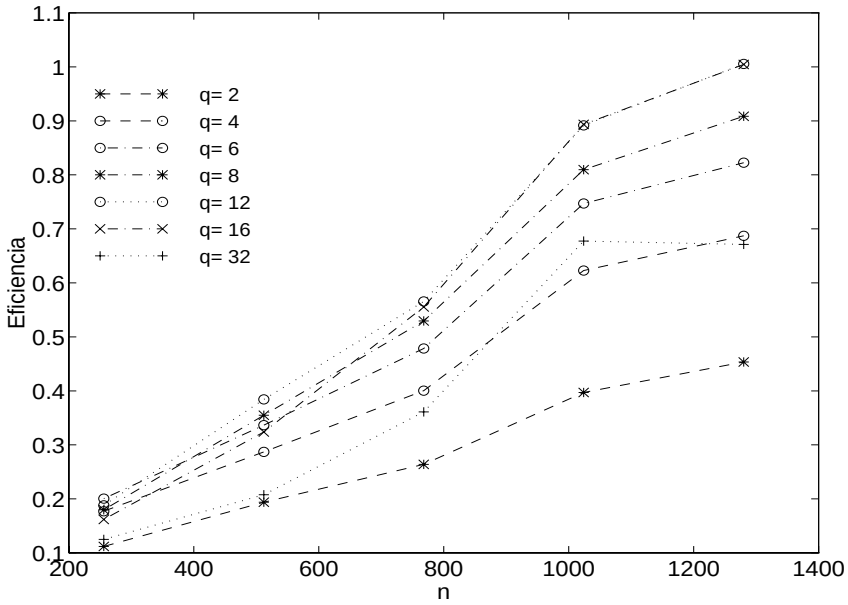


Figura 46. *Resultados de eficiencia en el PCh para el algoritmo paralelo PDHWB utilizando p= 4 procesadores y para bloques de tamaño qp× q, siendo q= 2, 4, 6, 8, 12, 16, 32.*

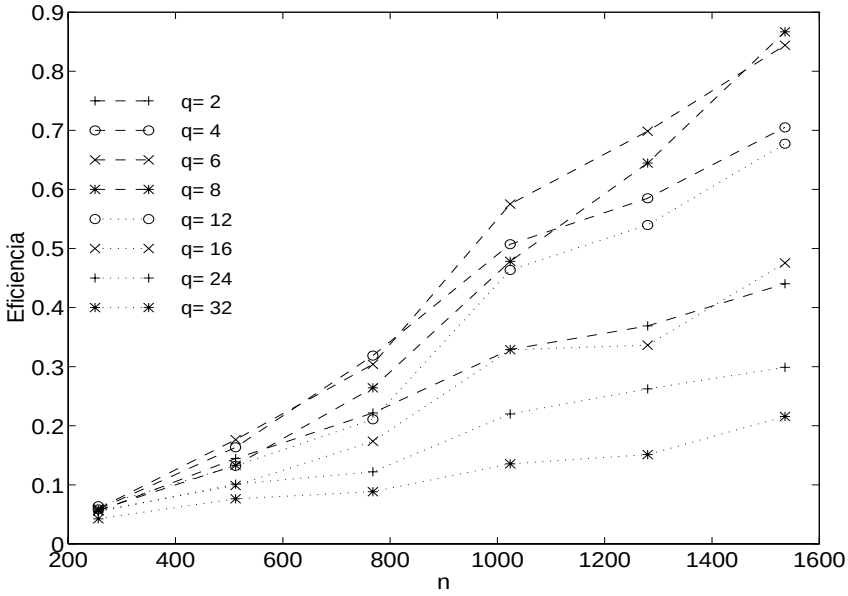


Figura 47. *Resultados de eficiencia en el PCh para el algoritmo paralelo PDHWB utilizando 8 procesadores y para bloques de tamaño qp× q, siendo q= 2, 4, 6, 8, 12, 16, 24 y 32.*

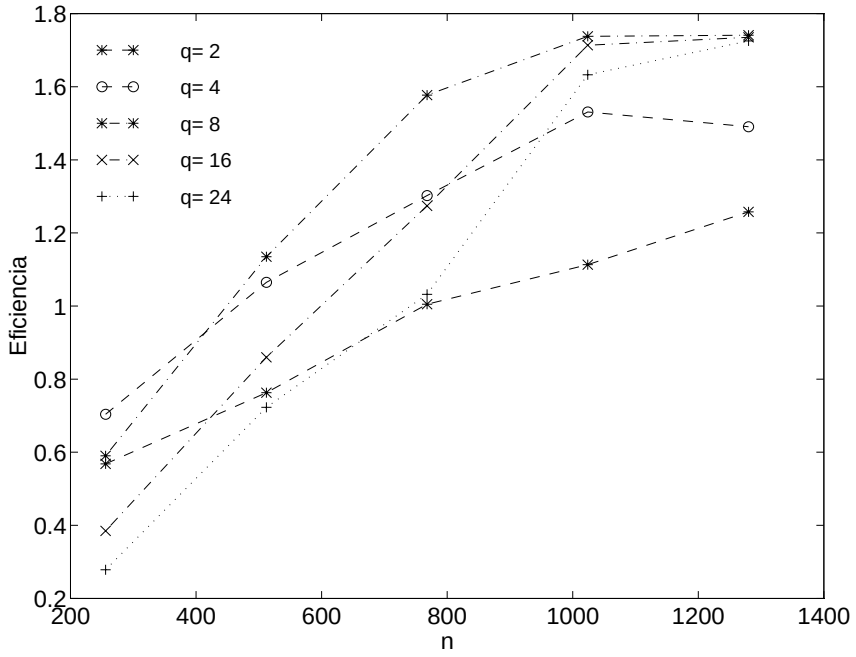


Figura 48. *Resultados de eficiencia en el T3D para el algoritmo paralelo PDHWB utilizando 4 procesadores y para bloques de tamaño qp× q, siendo q= 1, 2, 4, 8, 16, y 32.*

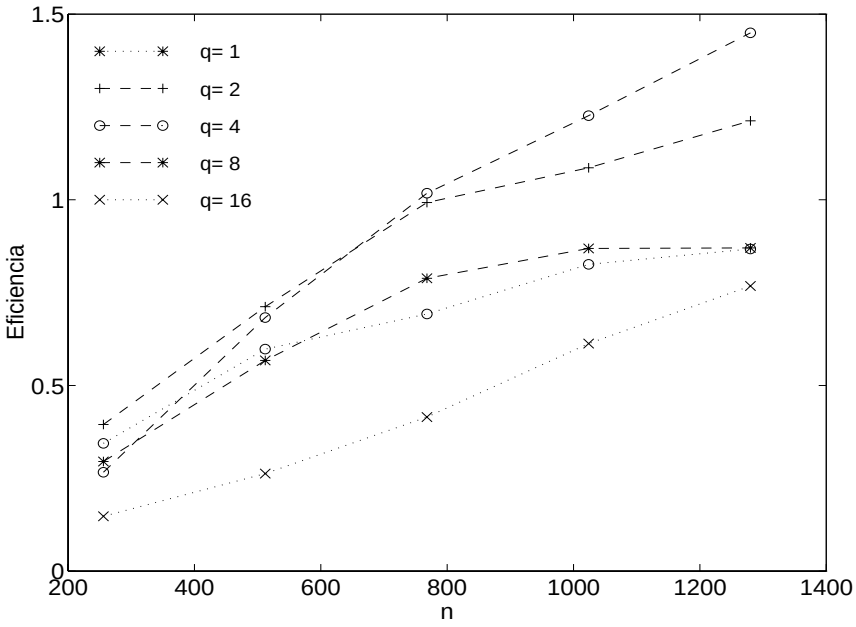


Figura 49. *Resultados de eficiencia en el T3D para el algoritmo paralelo PDHWB utilizando 8 procesadores y para bloques de tamaño qp× q, siendo q= 1, 2, 4, 8 y 16.*

En las Figuras 48 y 49 podemos ver los incrementos de velocidad obtenidos en este algoritmo para distintos valores de *q* y *n* en el T3D utilizando 4 y 8 procesadores, respectivamente. En este caso las prestaciones relativas obtenidas son mejores que para el PCh, debido a una antememoria mucho más pequeña. En este computador no se observa el incremento de prestaciones brusco que se aprecia

en el PCh cuando los datos dejan de caber todos en la antememoria. Además, el T3D posee una topología toroidal tridimensional, más adecuada para la transmisión simultánea de varios mensajes.

Por otra parte, en la Figura 50 se observa claramente que existe un pequeño rango de valores de q asociado a cada arquitectura multiprocesador, en los que se obtienen las mejores prestaciones para un tamaño de problema determinado. En esta figura vemos que para n=1024, este valor es de q=12 para 4 procesadores y de q=6 para 8 procesadores en el PCh. Como siempre que se toman bloques de tamaño fijo en problemas en los que el orden de éste se reduce a medida que se va resolviendo, el tamaño de bloque óptimo depende del orden del problema. Para el T3D podemos observar en la misma Figura 50 como los valores de q óptimos para n=1024 son q=8 para 4 procesadores y q=4 para 8 procesadores.

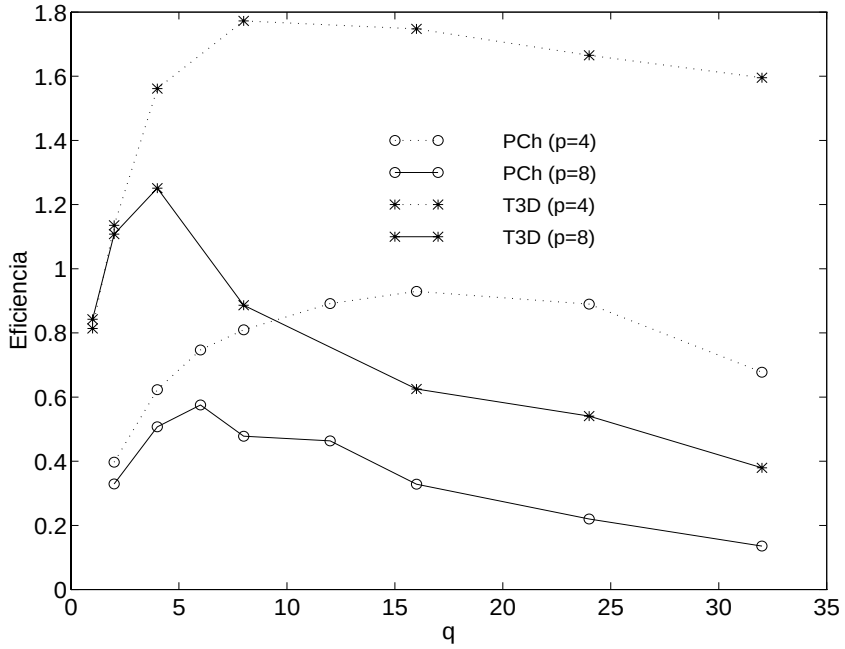


Figura 50. *Resultados de eficiencia en el PCH y el T3D para el algoritmo paralelo PDHWB con n=1024, utilizando 4y 8 procesadores y para diversos bloques de tamaño qp× q.*

Por otra parte, se aprecia como el valor óptimo de q es diferente para cada computador y éste se hace más crítico a medida que aumenta el número de procesadores. Este último comportamiento queda claramente reflejado en la Figura 51. En esta misma figura podemos ver como cuando aumenta el número de procesadores se reduce la eficiencia del algoritmo para un determinado tamaño del problema. Esto es debido fundamentalmente a la existencia de procesadores ociosos tanto en las primeras etapas como en las últimas de la resolución del problema. Este circunstancia se acentúa con el incremento del número de procesadores.

Tanto en el PCh como en el T3D hemos llegado a obtener eficiencias mayores que uno en algunos casos. Esto es debido a la práctica ausencia de esperas en nuestro algoritmo y la reducción de prestaciones del algoritmo secuencial debido al incremento del fallo de páginas en el manejo de la antememoria a partir de un cierto tamaño del problema.

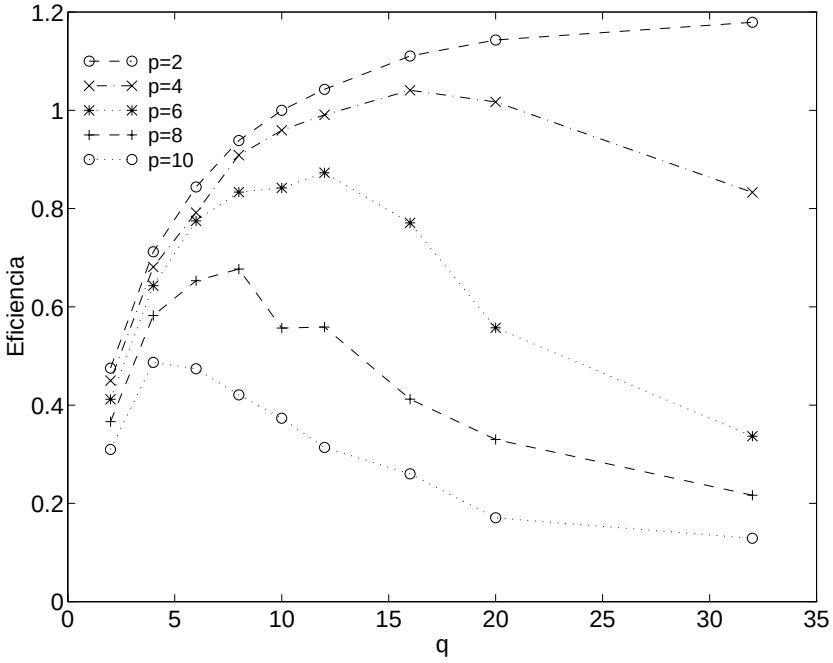


Figura 51. *Resultados de eficiencia en el PCh para el algoritmo paralelo PDHWB con n=1200, utilizando p= 2, 4, 6, 8 y 10 procesadores y para diversos bloques de tamaño qp×q.*

6.2.2.2 Algoritmos de frente de onda adaptativos

En este apartado presentamos los resultados obtenidos para los algoritmo paralelos de frente de onda adaptativos *PADHWB*. En la figura 52 se muestra la eficiencia obtenida para el algoritmo paralelo *PADHWB* utilizando 8 procesadores, para diferentes tamaños de problema y con diversos tamaños de bloques iniciales. El factor de reducción utilizado en estas implementaciones, y con el que se han obtenido los mejores resultados en nuestros tests experimentales, es de r=2. El tamaño de bloque más pequeño utilizado ha sido tomando para q= 2 ó 3, dependiendo éste del tamaño de bloque inicial elegido. El valor de la condición *condbloque* utilizado para la adaptación del tamaño de los bloques en tiempo de ejecución con el que se han obtenido los mejores resultados, también después de pruebas experimentales, ha sido de *condbloque*= 3p/2.

En la Figura 52 vemos el algoritmo *PADHWB* presenta para problemas de gran tamaño mejores prestaciones que el algoritmo *PDHWB*. Aunque este incremento es menor de lo que en principio se esperaba, la eficiencia obtenida supera el 0,9 para tamaños de problema elevados. Para problemas de orden pequeño se observa que el comportamiento de los dos algoritmos es muy similar. Por otra parte es importante observar que las prestaciones obtenidas para el algoritmo *PADHWB* son más independientes del tamaño del bloque de resolución elegido que para el algoritmo *PDHWB*. Así, es posible observar que las curvas de eficiencia para diferentes valores de tamaños de bloque están muy juntas a partir de un determinado tamaño de bloque.

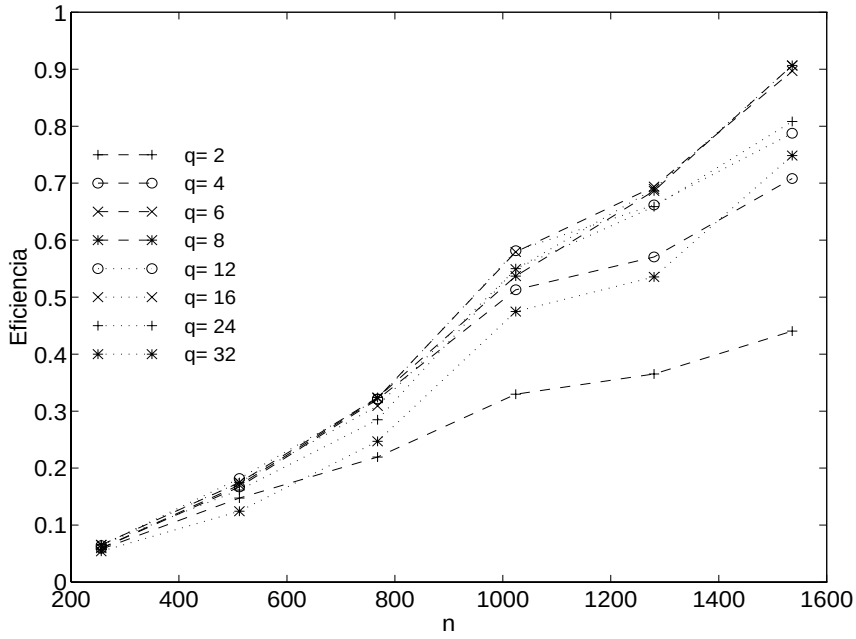


Figura 52. *Eficiencia obtenida para el algoritmo PADHWB en el PCh utilizando 8 procesadores para diferentes tamaños de problema y bloques de* $qp \times q$ *, donde q= 2, 4, 6, 8, 12, 16, 24 y 32.*

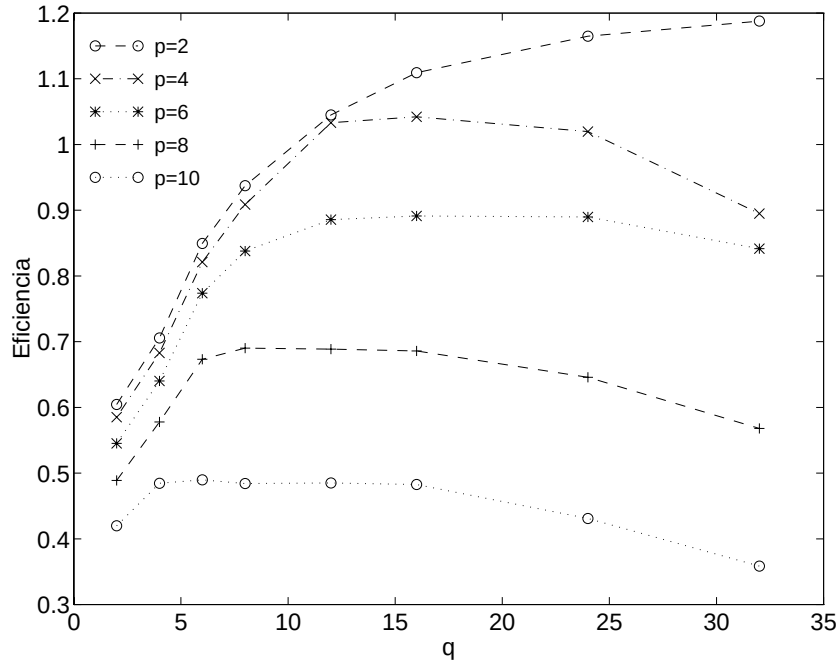


Figura 53. *Eficiencia obtenida para el algoritmo PADHWB en el PCh para n=1200, utilizando diferente número de procesadores y bloques de tamaño* $qp \times q$ *.*

El valor del tamaño de bloque para el que se obtienen las mejores prestaciones en el algoritmo *PDHWB* depende tanto del número de procesadores y como del orden del problema. Como se observa

en la Figura 51, el valor del tamaño de bloque influye más en las prestaciones del algoritmo *PDHWB* cuanto mayor es el número de procesadores.

En la Figura 53 se muestra como, para n= 1200 y con diferente número de procesadores, la eficiencia del algoritmo *PADHWB* se mantiene prácticamente constante en amplio rango de valores iniciales de q,. Por lo tanto, la elección del bloque de resolución afecta en estos algoritmos en menor medida a sus prestaciones que para el algoritmo *PDHWB*.

Al igual que para el algoritmo *PDHWB*, y por las mismas causas, obtenemos eficiencias mayores que uno en los algoritmos *PADHWB* para algunos tamaños del problema.

6.2.2.3 Algoritmos cíclicos

En este apartado presentamos los resultados obtenidos en los algoritmos cíclicos. Las matrices de la ecuación de Lyapunov se han distribuido de forma cíclica por filas y la matriz solución queda también distribuida cíclicamente entre los procesadores. En nuestros experimentos hemos utilizado un SGI Power Challenge (PCh).

De los algoritmos expuestos en el capítulo 5, aquellos que utilizan un grano pequeño de resolución y poca reutilización de los datos en la antememoria de los procesadores nos proporcionan prestaciones muy bajas comparadas con el algoritmo secuencial por bloques. En particular, para los algoritmos *PHCF* y *PHCF2* se obtienen eficiencias muy bajas ya que en ellos todos los procesadores se dedican a resolver una columna de la matriz solución cada vez y hay sólo un segmento circulando entre los procesadores. Estos algoritmos adolecen de los problemas ya conocidos en la resolución de sistemas triangulares [Li 89, Eisenstat 88] apareciendo esperas al aumentar el número de procesadores y el tamaño del problema.

La adaptación de las ideas de los algoritmos cíclicos modificados en sistemas triangulares a la resolución de la ecuación de Lyapunov en los algoritmos *PHCFU* y *PHCFG*, supone una mejora en los resultados experimentales. Sin embargo, debido a que se sigue utilizando un grano fino de resolución, el número de procesadores en el PCh es reducido (un máximo de 12 procesadores) y la desfavorable relación entre la velocidad de cálculo de los procesadores y el ancho de banda de las comunicaciones, siguen apareciendo esperas.

El incremento del grano de resolución del problema, como ya vimos en el análisis teórico de estos algoritmos, no resuelve el problema de las esperas. La única forma de subsanar estas esperas es incrementar en número de segmentos que circulan entre los procesadores y al mismo tiempo aprovechar en mayor medida los datos en la antememoria de los procesadores. Esto podemos conseguirlo con los algoritmos cíclicos de frente de onda.

En las Figuras 54 y 55 mostramos los resultados obtenidos para el algoritmo cíclico de frente de onda *PHCWF* en el PCh para diversos tamaños de problema en 4 y 8 procesadores, respectivamente, y donde r indica el número de columnas que se están resolviendo simultáneamente. En la implementación del algoritmo se hacen circular dos segmentos por cada columna que se está resolviendo, uno correspondiente a la etapa de resolución y otro a la de triangulación. El agrupamiento de estos dos mensajes en uno solo redunda en una reducción de las prestaciones del algoritmo. Para el caso en que r=1 el algoritmo *PHCWF* se correspondería con el algoritmo *PHCF2*.

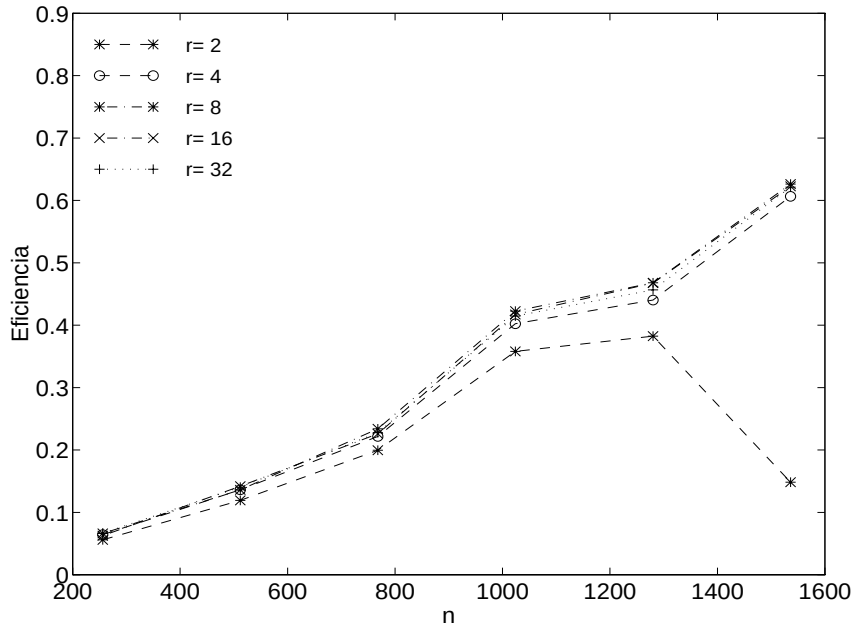


Figura 54. *Eficiencia obtenida para el algoritmo PHCWF en el PCh utilizando 4 procesadores para diferentes tamaños de problema y valores de r.*

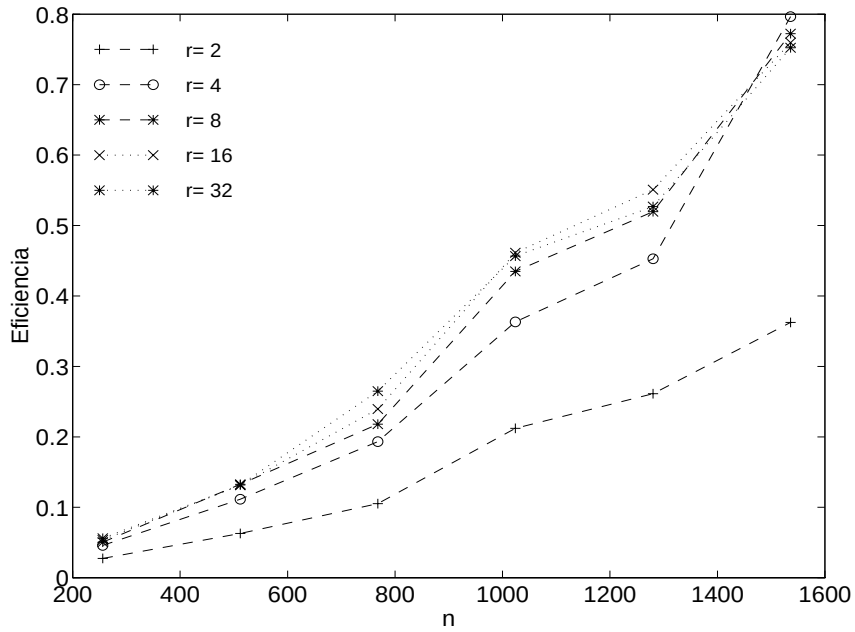


Figura 55. *Eficiencia obtenida para el algoritmo PHCWF en el PCh utilizando 8 procesadores para diferentes tamaños de problema y valores de r.*

En la Figura 54 se observa como, utilizando 4 procesadores, para r=2 la eficiencia es muy baja, e incluso desciende claramente para problemas de gran tamaño. Al incrementar el valor de r la

eficiencia aumenta de forma generalizada para todos los valores de *n*, estabilizándose a partir de *r*=8. Sin embargo la eficiencia no supera para problemas de orden elevado el 0,65.

Como vemos en la Figura 55, ejecutando el programa sobre 8 procesadores obtenemos un incremento de la eficiencia al aumentar el tamaño del problema y el valor de *r*. Ahora la eficiencia para valores de n elevados alcanza el 0,8. Este comportamiento no nos debe extrañar dado que es característico de este tipo de algoritmos, como podemos comprobar en [Heat 88] en el caso de la resolución de sistemas triangulares y que no ocurre en el caso de los algoritmos frente de onda. También observamos como los resultados son mejores para valores de *r* mayores cuando el tamaño del problema es pequeño (por ejemplo, *r*=32 para valores *n*=256 hasta 1024). A medida que el orden del problema aumenta el valor de *r* para el que se obtienen los mejores resultados es menor (por ejemplo, *r*=16 para *n*=1280 y *r*=4 para *n*=1536).

Las Figuras 56 y 57 muestran los resultados obtenidos en el algoritmo cíclico de frente de onda *PHCWFU* en el PCh para diversos tamaños de problema en 4 y 8 procesadores, respectivamente. En este algoritmo, a diferencia del algoritmo *PHCWF*, cada procesador calcula un subvector fila de *q* elementos pertenecientes a la misma fila antes de enviar el segmento que circula entre los procesadores por lo que el grano computacional es mayor. En nuestros experimentos *r* es múltiplo de *q*, en particular *q*=*r*/*p*, y para cada subvector que se calcula sólo se envía un segmento de tamaño 3*q*(*p*-1), agrupando por tanto las etapas de resolución y triangulación. El algoritmo *PHCWFU* para *q*=1 es el algoritmo *PHCWF*.

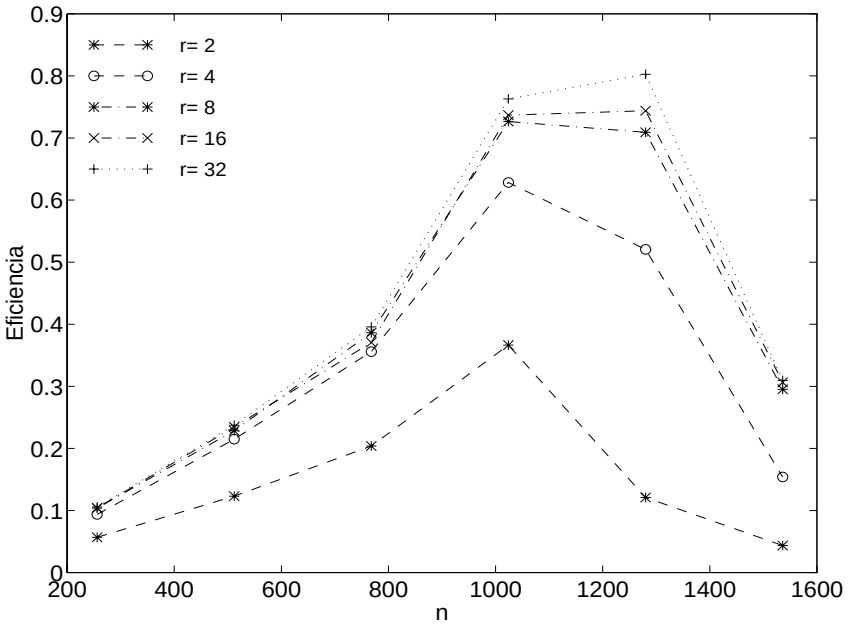


Figura 56. *Eficiencia obtenida para el algoritmo PHCWFU (con circulación de un mensaje de tamaño 3q(p-1)) en el PCh utilizando 4 procesadores para diferentes tamaños de problema y valores de r.*

Como se observa en la Figura 56, las eficiencias obtenidas para 4 procesadores son mucho mejores en el algoritmo *PHCWFU* que en el *PHCWF* para problemas de tamaño menor. Las prestaciones se incrementen a medida que aumentamos el valor de *r*, obteniéndose los mejores resultados para *r*=32. Esto supone un valor de *q*=8, siendo los resultados peores para valores de *r*

mayores. Sin embargo, para problemas de orden elevado, las prestaciones del algoritmo se reducen drásticamente.

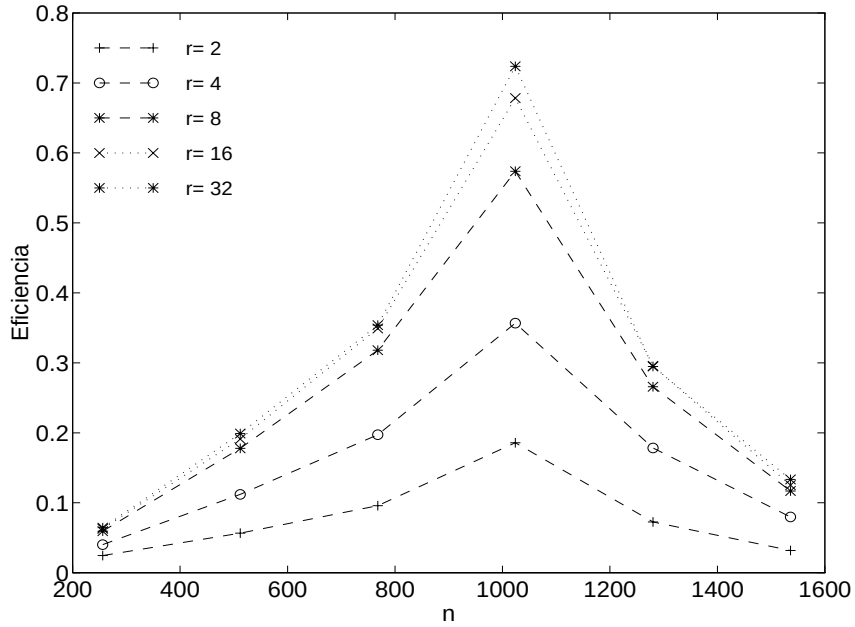


Figura 57. *Eficiencia obtenida para el algoritmo PHCWFU (con circulación de un mensaje de tamaño 3q(p-1)) en el PCh utilizando 8 procesadores para diferentes tamaños de problema y valores de r.*

Un comportamiento similar aparece cuando ejecutamos el algoritmo *PHCWFU* sobre 8 procesadores (ver Figura 57). En este caso la caída es mucho más acusada, llegándose ha obtener muy buenas prestaciones para n=1024 con una eficiencia de 0,73 y cayendo hasta una eficiencia del 0,13 para n=1536. Las mejores prestaciones se obtienen para r=32 (q=4).

En el siguiente experimento modificamos el algoritmo *PHCWFU* para que, en vez de enviar un único segmento por subvector fila calculado, se envíen dos segmentos, cada uno asociado a las etapas de resolución y triangulación. Estos subsegmentos serán de tamaño $q(p\text{-}1)$ para la etapa de resolución y de $2q(p\text{-}1)$ para la etapa de triangulación. En las Figuras 58 y 59 se muestran los resultados de eficiencia obtenidos en el PCh para 4 y 8 procesadores, respectivamente.

Como se observa en la Figura 58, las prestaciones son mejores en esta nueva versión del algoritmo *PHCWFU*. El algoritmo obtiene mejores eficiencias para problemas de orden menor y la degradación de las prestaciones que se presentaba en la versión anterior sigue apareciendo para valores de r extremos (para r=2 y 4, y para r=32). Sin embargo existen valores de r para los que el algoritmo se comporta incluso mejor que el algoritmo *PHCWF*, alcanzando eficiencias próximas a 1 para n=1536.

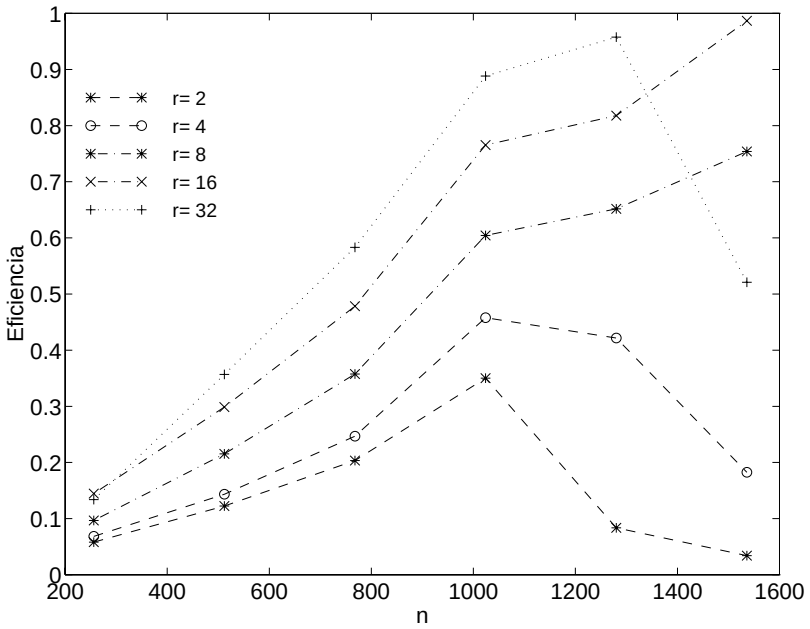


Figura 58. *Eficiencia obtenida para el algoritmo PHCWFU (con circulación de dos mensajes de tamaños q(p-1) y 2q(p-1)) en el PCh utilizando 4 procesadores para diferentes tamaños de problema y valores de r.*

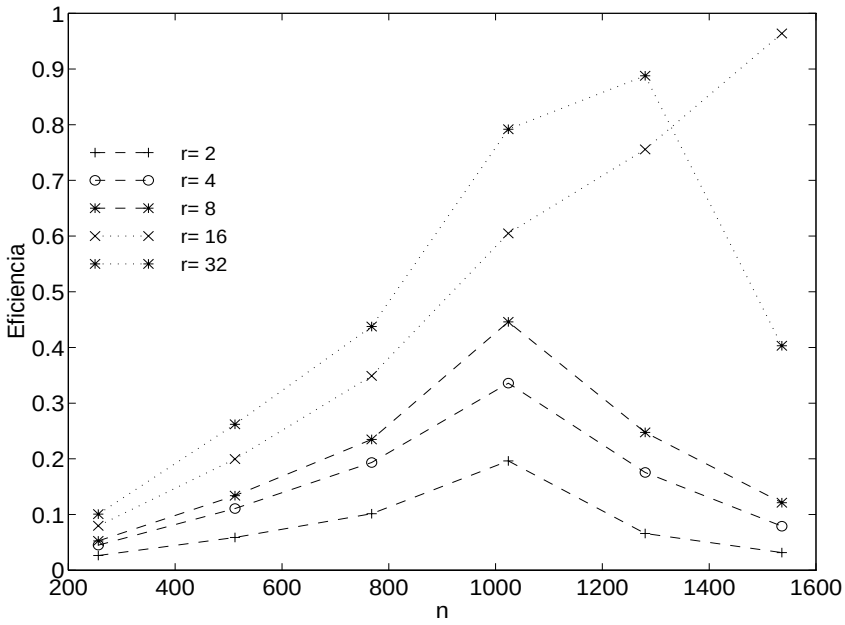


Figura 59. *Eficiencia obtenida para el algoritmo PHCWFU (con circulación de dos mensajes de tamaños q(p-1) y 2q(p-1)) en el PCh utilizando 8 procesadores para diferentes tamaños de problema y valores de r.*

En la Figura 59 observamos que el comportamiento del algoritmo *PHCWFU* en el PCh sobre 8 procesadores es muy parecido al comportamiento visto sobre 4 procesadores. En ambos casos se obtienen las mejores prestaciones utilizando r=32 hasta n=1280 y a partir de ese tamaño de problema las prestaciones se reducen considerablemente y sólo se incrementan para r=16.

6.2.3 Análisis de resultados

En esta sección se han presentado los resultados obtenidos en los algoritmos paralelos basados en el método de Hammarling [Hammarling 82, Hammarling 91]. En particular se han evaluado nuevos algoritmos paralelos de grano fino y medio para resolver las ecuaciones de Lyapunov en tiempo discreto en multiprocesadores con memoria compartida. Para obtener las máximas prestaciones posibles, se han aplicado técnicas de frente de onda, resolución por bloques de columnas y solapamiento entre la resolución del fin de un bloque de columnas y el principio del siguiente.

Los algoritmos de grano de resolución fino (*PBSHWF*) han mostrado ser muy adecuados para multiprocesadores con procesadores escalares y superescalares. La eficiencia del algoritmo *PBSHWF* crece muy rápidamente, manteniéndose alrededor de 0,8 para problemas de gran tamaño.

Como se puede observar en la Tabla 6, los algoritmos de grano fino (*PBSHWF*) no obtienen buenas eficiencias en multiprocesadores con unidades vectoriales. A fin de aprovechar las prestaciones adicionales que proporcionan los sistemas con unidades vectoriales se han diseñado algoritmos paralelos de grano medio por bloques de columnas (*PBSHWM*). En la Tabla 6 observamos que estos algoritmos no obtienen buenas prestaciones para tamaños de problema pequeños ya que el grano de resolución es menor, lo que redunda una reducción del paralelismo del problema. Sin embargo, para problemas de tamaño mayor, el algoritmo *PBSHWM* alcanza eficiencias mayores que el algoritmo *PBSHWF*.

Como el tamaño de las matrices del problema se reduce en cada etapa, y dado que los algoritmos de grano fino ofrecen mejores prestaciones para tamaños de problemas pequeños, se han implementado algoritmos que adaptan el grano de resolución basados en la combinación de los algoritmos con granos de resolución fino y medio (*PBSHWC*), consiguiéndose buenas eficiencias para un amplio rango de tamaños de ecuaciones.

Algoritmos	n=40	n=100	n=200	n=500
PBSHWF	0,35	0,35	0,34	0,38
PBSHWM	0,21	0,24	0,34	0,56
PBSHWC	0,38	0,4	0,37	0,58

Tabla 6. *Eficiencia de los algoritmos paralelos de grano fino, medio y combinado utilizando 8 procesadores respecto del algoritmo serie de Hammarling SH en el Alliant FX/80. Se utilizan las optimizaciones vectoriales. Se han considerado los mejores resultados obtenidos en cada algoritmo.*

Así mismo, basados en el método de Hammarling, se han evaluado nuevos algoritmos paralelos para sistemas multiprocesadores con memoria distribuida que utilizan paso de mensajes para las comunicaciones. En estos algoritmos se han aplicado y extendido las técnicas utilizadas para la resolución de sistemas triangulares lineales [Chamberlain 86, Eisensat 88, Heath 88, Li 88].

Se han implementado algoritmos paralelos frente de onda con granos de resolución fino (*PDHWF*), medio (*PDHWM*) y grueso (o por bloques) (*PDHWB*). Los mejores resultados se han obtenido para los algoritmos por bloques (*PDHWB*), siendo excelentes las eficiencias al aumentar el tamaño del problema. De echo se han alcanzado, para tamaños de problema elevados, eficiencias superiores a la unidad (*superspeedup*). Esto es debido a que los algoritmos frente de onda diseñados poseen tanto un excelente uso de la antememoria, como un alto solapamiento entre el cálculo aritmético y las comunicaciones. Además, se han evaluado algoritmos que adaptan dinámicamente del grano de resolución a medida que se va resolviendo el problema (*PADHWB*). Éste algoritmo proporciona, como muestra la Tabla 7, una mejora de las prestaciones respecto del algoritmo *PDHWB*.

Algoritmos	n=256	n=512	n=1024	n=1280	n=1536
PDHWB	0,05	0,18	0,57	0,70	0,86
PADHWB	0,06	0,19	0,58	0,74	0,92
PHCWF	0,06	0,13	0,47	0,55	0,80
PHCWFU	0,1	0,26	0,79	0,88	0,96

Tabla 7. *Eficiencia de los algoritmos paralelos de frente de onda y cíclicos (con frente de onda) utilizando 8 procesadores del SGI Power Challenge. Se han considerado los mejores resultados obtenidos en cada algoritmo*

Por otra parte, se han implementado algoritmos paralelos cíclicos con grano de resolución fino (*PHCF, PHCF2*), medio (*PHCM, PHCMR*) y grueso (*PHCB*). En este tipo de algoritmos la aparición de esperas degrada sus prestaciones. Para reducir estos tiempos de espera se han propuesto mejoras basadas en la optimización del envío de los mensajes (*PHCFU, PHCFG*), pero dada la naturaleza del método de Hammarling y las características de los computadores actuales, las prestaciones obtenidas son muy bajas. Para aprovechar las ventajas de los algoritmos frente de onda y cíclicos, se han propuesto nuevos algoritmos con diferente grano de resolución basados en la combinación de estas dos técnicas. Así, se han implementado y evaluado algoritmos cíclicos de frente de onda (*PHCWF, PHCWFU, PHCWFG, PHCWFN*) que mejoran considerablemente las prestaciones de los algoritmos cíclicos anteriores. En particular, para el algoritmo *PHCWFU* se obtienen mejores prestaciones que en los algoritmos de frente de onda al incrementar el número de procesadores y para cualquier tamaño de problema (ver Tabla 7).

6.3 Método de la función signo matricial

El método de la función signo matricial no puede considerarse como un algoritmo numéricamente estable para la resolución de la ecuación de Lyapunov, aunque en la práctica, como mostramos en esta sección, funciona como tal.

En el apartado 6.3.1 analizamos la fiabilidad numérica de los algoritmos basados en la función signo matricial y comparamos este método con los algoritmos directos de Bartels-Stewart y Hammarling. Estudiaremos varios ejemplos de ecuaciones, en función de las características de sus matrices de coeficientes y matriz de términos independientes.

En el apartado 6.3.2 presentamos los resultados experimentales de los algoritmos paralelos desarrollados para resolver la ecuación de Lyapunov generalizada, tanto para obtener la solución explícita de la ecuación como su factor de Cholesky.

Por último, en el apartado 6.3.3 estudiamos las prestaciones obtenidas en los algoritmos paralelos para resolver las ecuaciones de Lyapunov, estándar y generalizadas, acopladas que aparecen en la reducción de modelos. Evaluaremos este algoritmo mediante sistemas de simple entrada/salida y de múltiples entradas/salidas, y se considerará la obtención de las soluciones explícitas de estas ecuaciones como sus factores de Cholesky. En los apartados 6.3.2 y 6.3.3 también se realizará un análisis de la escalabilidad de los algoritmos.

6.3.1 Análisis de la fiabilidad numérica

En este apartado comparamos el método *SILGE*, basado en la función signo matricial para resolver las ecuaciones de Lyapunov generalizadas. Incluimos en la comparación el algoritmo de Bartels-Stewart (*BTST*) y, en aquellos ejemplos en los que la matriz de coeficientes sea estable y la matriz de términos independientes simétrica semidefinida positiva, incluimos así mismo el algoritmo *SILGCE* y el algoritmo de Hammarling (*HAMM*). Los códigos de los algoritmos *BTST* y *HAMM* están descritos en [Penzl 96].

Hemos empleado el esquema iterativo para la función signo matricial descrito en el capítulo 3 para el algoritmo *SILGE*, y para calcular el factor de Cholesky en el algoritmo *SILGCE*. En todos los esquemas iterativos se han utilizado los criterios de convergencia descritos en el capítulo 3, por ejemplo,

$$\|A_{k+1} + E\|_1 \leq 10 \cdot n \cdot \sqrt{\varepsilon} \cdot \|E\|_1,$$

más dos iteraciones adicionales después de satisfecho el criterio de convergencia.

En los resultados experimentales la exactitud de los algoritmos se ha estimado mediante el residuo normalizado

$$\frac{\|A^T XE + E^T XA + Q\|_1}{\|X\|_1}.$$

Por otra parte, los resultados numéricos se han contrastado con la separación de la ecuación de Lyapunov, $sep(A^T XE + E^T XA + Q)$, y el número de condición del problema, $\mathrm{cond}_2(A^T XE + E^T XA + Q)$, definidos en la sección 3.2. Estos valores se han estimado utilizando el algoritmo *BTST* de [Penzl 96].

Todos los experimentos se realizaron utilizando Fortran 77 y aritmética IEEE en doble precisión ($\varepsilon \approx 2.2 \times 10^{-16}$), en un ordenador SUN UltraSparc-167MHz. Se han utilizado las opciones de compilación apropiadas en los algoritmos para optimizar las prestaciones. También hemos hecho un uso intensivo de las librerías BLAS y del núcleos computacionales del LAPACK [Anderson 94].

A continuación presentamos los ejemplos utilizados para el análisis y comparación de la precisión numérica de los algoritmos utilizados.

Ejemplo 1. [Gardiner 92] Las matrices de coeficientes en este ejemplo se definen como

$$A = -((2^{-\tau} - 1)I_n + \text{diag}(1,2,\ldots,n) + U_n^T), \text{ y } E = I_n + 2^{-\tau}U_n,$$

donde U_n es una matriz $n \times n$ triangular inferior con todos los elementos sobre la diagonal y por debajo de ésta iguales a 1. La matriz Q se ha calculado de la forma

$$Q = -(A^T XE + E^T XA),$$

donde X tiene todas sus elementos a 1.

El número de condición $\text{cond}_2(A^T XE + E^T XA + Q)$ se controla mediante el parámetro τ. Cuando de incrementa el valor de τ, uno de los valores propios generalizados del haz de matrices $A - \lambda E$ se aproxima a cero y la ecuación de Lyapunov generalizada está peor condicionada. La Tabla 8 ilustra la separación y el correspondiente número de condición para matrices de dimensión n=100 y diferentes valores de τ. Como era de esperar, la tabla muestra un incremento constante en el mal condicionamiento del problema cuando se incrementa τ.

τ	$sep(A^T XE + E^T XA + Q)$	$1/\text{cond}_2(A^T XE + E^T XA + Q)$
10	4.9×10^{-4}	4.2×10^{-8}
20	4.3×10^{-7}	3.7×10^{-11}
30	4.2×10^{-10}	3.6×10^{-14}
40	4.1×10^{-13}	3.6×10^{-17}

Tabla 8. *Separación y número de condición para el Ejemplo 1 (n=100).*

La tabla 9 ilustra los residuos normalizados obtenidos para los algoritmos basados en el método Bartels-Stewart (*BTST*) y la función signo matricial (*SILGE* y *SILGCE*) para el Ejemplo 1, n=100, y valores crecientes de τ. En este caso no es posible utilizar los algoritmo *SILGE* y *HAMM* puesto que A no es una matriz estable.

τ	*BTST*	*SILGE*
10	$3{,}1 \times 10^{-12}$	$1{,}1 \times 10^{-10}$ (19)
20	$6{,}3 \times 10^{-12}$	$5{,}4 \times 10^{-8}$ (27)
30	$1{,}3 \times 10^{-12}$	$5{,}8 \times 10^{-5}$ (34)
40	$7{,}7 \times 10^{-13}$	$2{,}6 \times 10^{-2}$ (41)

Tabla 9. *Residuos normalizados y número de iteraciones de la función signo matricial (entre paréntesis) para el Ejemplo 1 (n=100).*

El algoritmo basado en la función signo matricial *SILGE* obtiene residuos normalizados que están de acuerdo con el número de condición del problema. Por otra parte, el algoritmo basado en el método de Bartels-Stewart presenta una exactitud muy alta. En realidad, los resultados son mucho mejores que los que se podrían esperar del condicionamiento de los problemas correspondientes, tal y como han sido estimados por $\text{cond}_2(A^T XE + E^T XA + Q)$.

Este comportamiento puede ser explicado del siguiente modo: en la primera etapa del algoritmo de Bartels-Stewart, el haz $A - \lambda E$ se reduce a la forma real de Schur generalizada mediante el algoritmo QZ. Debido a que todos los valores propios del haz de matrices $A - \lambda E$ son reales, las dos matrices reducidas son triangulares. Entonces, la ecuación Lyapunov triangular se resuelve mediante sustitución regresiva. La primera etapa revela un pequeño valor propio del haz $A - \lambda E$ que aparece como un bloque 1×1 en la parte superior de la diagonal de la forma real de Schur generalizada. Aunque la etapa de sustitución regresiva está mál condicionada debido a la existencia de este pequeño elemento, la ecuación de Lyapunov triangular se resuelve con una exactitud muy alta. Así, parece que este proceso de sustitución regresivo comparte la alta exactitud de la solución de los sistemas lineales triangulares [Higham 96 (capítulo 8), Wilkinson 63].

Ejemplo 2. Consideremos la siguiente variación de la matriz *A* del Ejemplo 1.

$$A = -((2^{-\tau} - 1)I_n + \text{diag}(n, n-1, \ldots, 1) + U_n^T).$$

La Tabla 10 muestra los residuos normalizados para la variante del Ejemplo 2. En la tabla podemos observar como una simple reordenación de los elementos de la diagonal de *A* produce diferencias en los resultados para los algoritmos basados en la función signo matricial. Se obtienen resultados similares para el método de Bartels-Stewart y los basados en la función signo matricial aunque $sep(A^T XE + E^T XA + Q)$ y $\text{cond}_2(A^T XE + E^T XA + Q)$ son los mismos que en el Ejemplo 1.

τ	*BTST*	*SILGE*
10	$3{,}1 \times 10^{-12}$	$1{,}1 \times 10^{-12}$ (19)
20	$6{,}3 \times 10^{-12}$	$5{,}4 \times 10^{-12}$ (27)
30	$1{,}3 \times 10^{-12}$	$5{,}8 \times 10^{-13}$ (34)
40	$7{,}7 \times 10^{-12}$	$2{,}6 \times 10^{-12}$ (41)

Tabla 10. *Residuos normalizados y número de iteraciones de la función signo matricial (entre paréntesis) para el Ejemplo 2 (n=100).*

Ejemplo 3. [Penzl 96] Las matrices de coeficientes en este ejemplo se definen para $n = 3q$ del siguiente modo

$$A = V_n \text{diag}(A_1, \ldots, A_q) W_n,\ A_i = \begin{pmatrix} \sigma_i & 0 & 0 \\ 0 & \tau_i & \tau_i \\ 0 & -\tau_i & \tau_i \end{pmatrix}, \text{ y } E = V_n W_n,$$

donde W_n es una matriz $n \times n$ triangular inferior con todos los elementos en y bajo la diagonal iguales a 1, y V_n es una matriz $n \times n$ con elementos a 1 en y sobre la antidiagonal, y todos los demás elementos nulos. La matriz semidefinida positiva de términos independientes *Q* se define como

$$Q = C^T C,\ C = [1, 2, \ldots, n].$$

En este caso el haz de matrices $A - \lambda E$ posee valores propios reales y complejos, σ_i y $\tau_i \pm \imath\tau_i$, respectivamente, donde $\imath = \sqrt{-1}$.

τ	$sep(A^T XE + E^T XA + Q)$	$1/\mathrm{cond}_2(A^T XE + E^T XA + Q)$
1,0	$6{,}0 \times 10^{-1}$	$1{,}8 \times 10^{-9}$
1,2	$6{,}2 \times 10^{-2}$	$1{,}4 \times 10^{-11}$
1,4	$1{,}0 \times 10^{-1}$	$2{,}0 \times 10^{-13}$
1,6	$1{,}2 \times 10^{-1}$	$3{,}6 \times 10^{-15}$
1,8	$1{,}3 \times 10^{-1}$	$9{,}9 \times 10^{-17}$

Tabla 11. *Separación y correspondiente número de condición para el Ejemplo 3 (n=100).*

La Tabla 11 nos muestra la separación y el correspondiente número de condición para el Ejemplo 3, para matrices de dimensión *n*=100 y $\sigma_i = \tau_i = \tau^i$. Como muestra la tabla, la separación se altera muy ligeramente cuando se incrementa τ. Sin embargo, el problema está peor condicionado debido al incremento de $\|A\|_F$.

τ	*BTST*	*SILGE*
1,0	$2{,}5 \times 10^{-11}$	$5{,}9 \times 10^{-12}$ (6)
1,2	$9{,}2 \times 10^{-9}$	$1{,}7 \times 10^{-9}$ (8)
1,4	$1{,}7 \times 10^{-6}$	$3{,}1 \times 10^{-7}$ (9)
1,6	$7{,}0 \times 10^{-5}$	$2{,}8 \times 10^{-5}$ (9)
1,8	$3{,}9 \times 10^{-3}$	$6{,}4 \times 10^{-4}$ (10)

τ	*HAMM*	*SILGCE*
1,0	$3{,}4 \times 10^{-11}$	$2{,}9 \times 10^{-12}$ (6)
1,2	$1{,}2 \times 10^{-8}$	$5{,}0 \times 10^{-9}$ (8)
1,4	$9{,}6 \times 10^{-7}$	$6{,}9 \times 10^{-7}$ (9)
1,6	$2{,}8 \times 10^{-5}$	$5{,}7 \times 10^{-5}$ (9)
1,8	$2{,}9 \times 10^{-3}$	$8{,}1 \times 10^{-4}$ (10)

Tabla 12. *Residuos normalizados y número de iteraciones de la función signo matricial (entre paréntesis) para el Ejemplo 3 (n=99).*

En la Tabla 12 podemos observar que los métodos basados en la función signo matricial, *SILGE* y *SILGCE*, obtienen una precisión muy similar a la de los algoritmos *BTST* y *HAMM*.

Ejemplo 4. También hemos generado las matrices de coeficientes *A* y *E* con valores aleatorios utilizando una distribución uniforme y valores propios generalizados aleatorios, uniformemente distribuidos en [-1,0). La matriz *X* tenía todos los valores a 1 y la matriz *Q* fue generada como en el Ejemplo 3. Los resultados numéricos mostraron una exactitud muy similar para todos los algoritmos.

6.3.2 Ecuación de Lyapunov generalizada

En este apartado comparamos las prestaciones de los algoritmos que resuelven las ecuaciones de Lyapunov generalizadas. En nuestros ejemplos, la solución calculada se ha obtenido con la exactitud que puede esperarse del condicionamiento del problema descrito en el apartado anterior.

Todos los experimentos se han realizado utilizando Fortran 77 y aritmética IEEE de doble precisión en un IBM SP2 (ver capítulo 4). En nuestros tests sobre un nodo se obtienen 200 Mflops para el producto de matrices (rutina DGEMM). Utilizamos el BLAS optimizado para esta máquina, y las librerías LAPACK, BLACS y ScaLAPACK [Anderson 94, Blackford 97] para asegurar la portabilidad de los algoritmos.

Las matrices de coeficientes en nuestros experimentos se definen del siguiente modo

$$A = V_n \text{diag}(\alpha_1, \ldots, \alpha_n) W_n \text{ , y } E = V_n W_n ,$$

donde los valores escalares $\alpha_i, 1 \le i \le n$, están uniformemente distribuidos en el intervalo [-10,0), W_n es una matriz $n \times n$ triangular inferior con todos los elementos en y sobre la diagonal iguales a 1, y V_n es una matriz $n \times n$ con elementos a 1 en y sobre la antidiagonal, y todos los demás elementos nulos. La matriz de términos independientes Q se define como el producto $Q = C^T C$, donde C es una matriz $r \times n$ generada aleatoriamente. Recordemos que el criterio de convergencia utilizado en nuestros algoritmos no involucra a las matrices C o Q.

El tiempo de ejecución por iteración en los algoritmos para resolver la ecuación de Lyapunov basados en la función signo matricial no depende de las características o estructura de sus matrices. Por tanto, este sencillo ejemplo nos permite analizar el grado de paralelismo de nuestros algoritmos.

El tiempo de ejecución de los algoritmos basados en la función signo matricial depende del número de iteraciones requeridas para alcanzar la convergencia. En nuestros experimentos hemos realizado 10 iteraciones de estos algoritmos, de tal forma que los costes teóricos de los métodos directos y nuestros métodos iterativos sean similares.

Como se muestra en las Figuras 60 y 61, en la práctica, los algoritmos basados en la función signo matricial obtienen una mayores prestaciones debido una implementación muy eficiente de sus núcleos computacionales. En nuestros experimentos los algoritmos secuenciales basados en la función signo matricial han mostrado mejores prestaciones que los métodos directos incluso cuando los primeros requieren 20 iteraciones para converger.

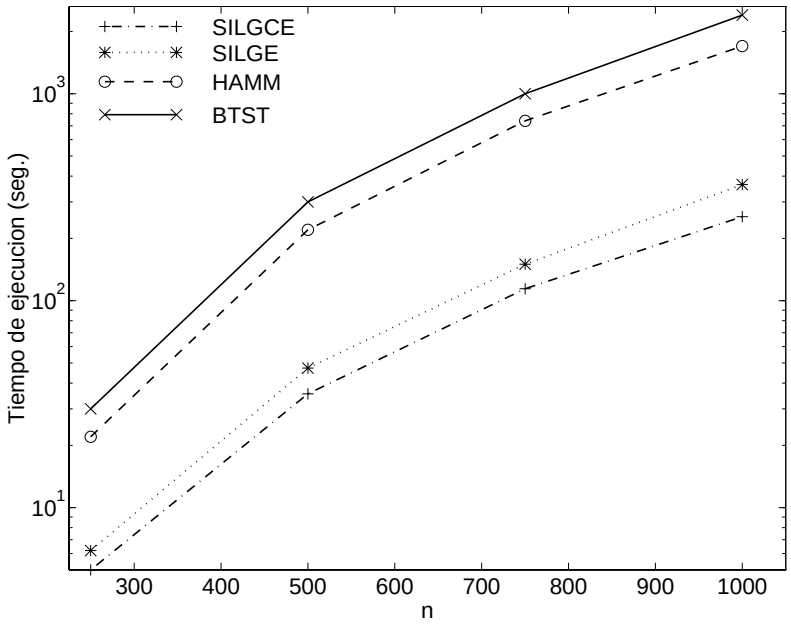


Figura 60. *Tiempo de ejecución para los algoritmos secuenciales para resolver la ecuación de Lyapunov generalizada con r=1 en un procesador del IBM SP2.*

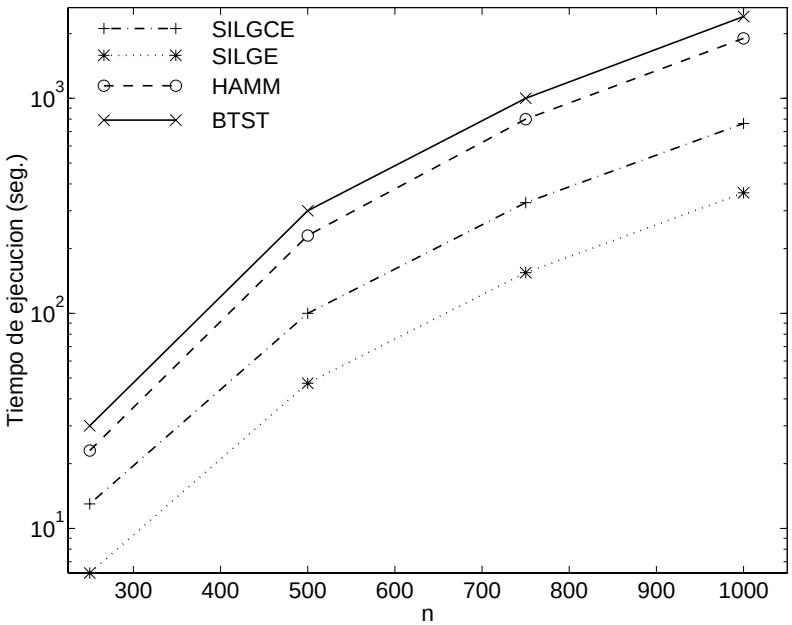


Figura 61. *Tiempo de ejecución para los algoritmos secuenciales para resolver la ecuación de Lyapunov generalizada con r=n en un procesador del IBM SP2.*

En las Figuras 62 a 65, mostramos los tiempos de ejecución de los algoritmos *PSILGE* y *PSILGCE* en 1, 2×2, 3×3 y 4×4 procesadores, con r=1, $n/2$ y n.

En el caso semidefinido, cuando el rango de Q es bajo ($r << n$), resolver la ecuación de Lyapunov para obtener el factor de Cholesky es más eficiente que resolver la ecuación para obtener la solución explícita X. Muchas aplicaciones requieren precisamente este factor de Cholesky. Por otra parte, el alto sobrecoste del algoritmo *PSILGCE* cuando *r=n* puede ser sólo justificado por una posible ganancia de precisión numérica.

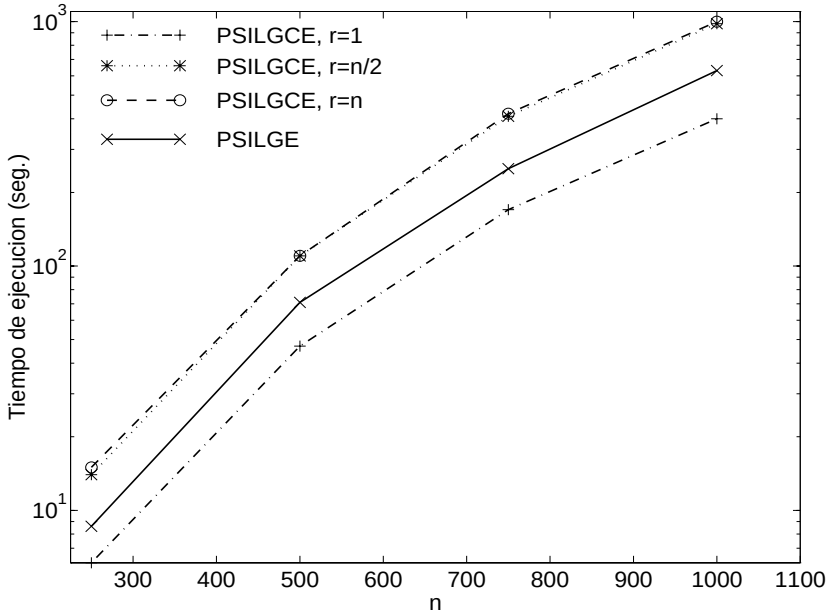


Figura 62. *Tiempo de ejecución de los algoritmos serie para resolver la ecuación de Lyapunov generalizada con r=1, r=n/2 y r=n en 1 procesador del SP2.*

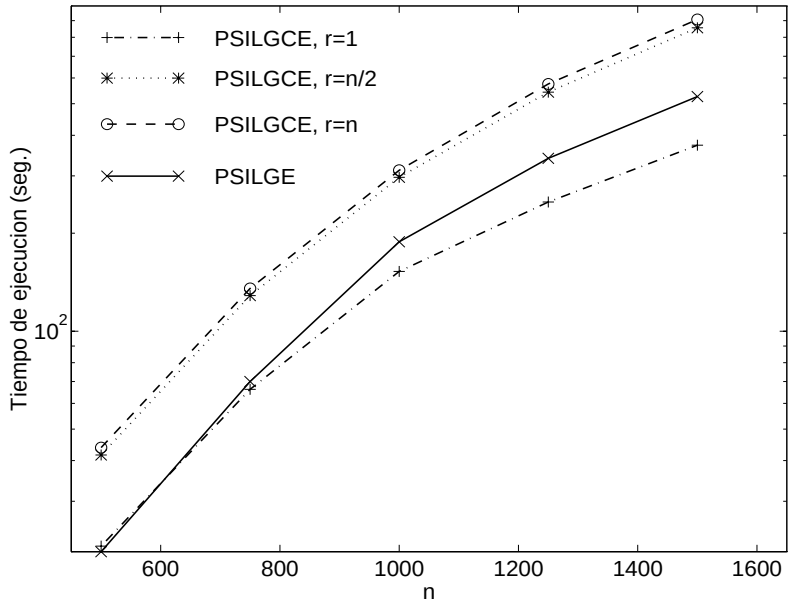


Figura 63. *Tiempo de ejecución de los algoritmos paralelos para resolver la ecuación de Lyapunov generalizada con r=1, r=n/2 y r=n en 2× 2 procesadores del SP2.*

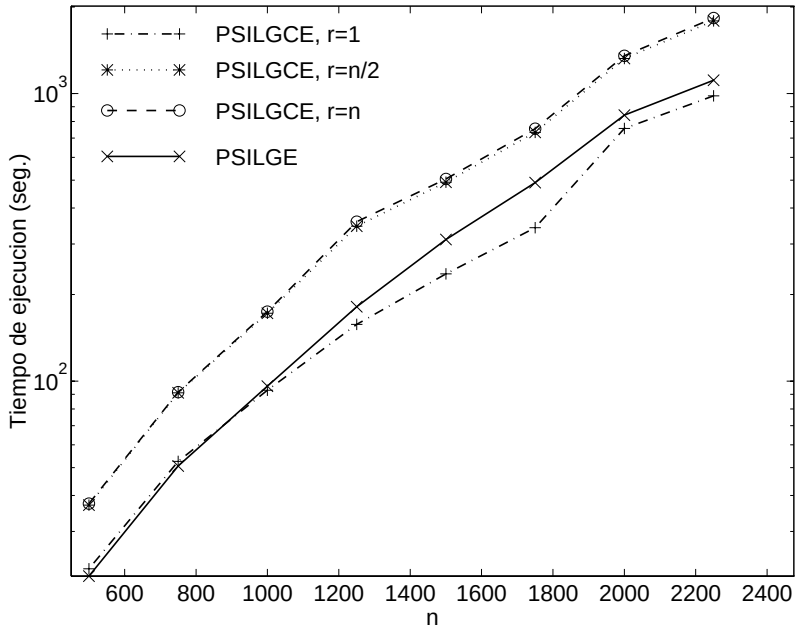


Figura 64. *Tiempo de ejecución de los algoritmos paralelos para resolver la ecuación de Lyapunov generalizada con r=1, r=n/2 y r=n en 3× 3 procesadores del SP2.*

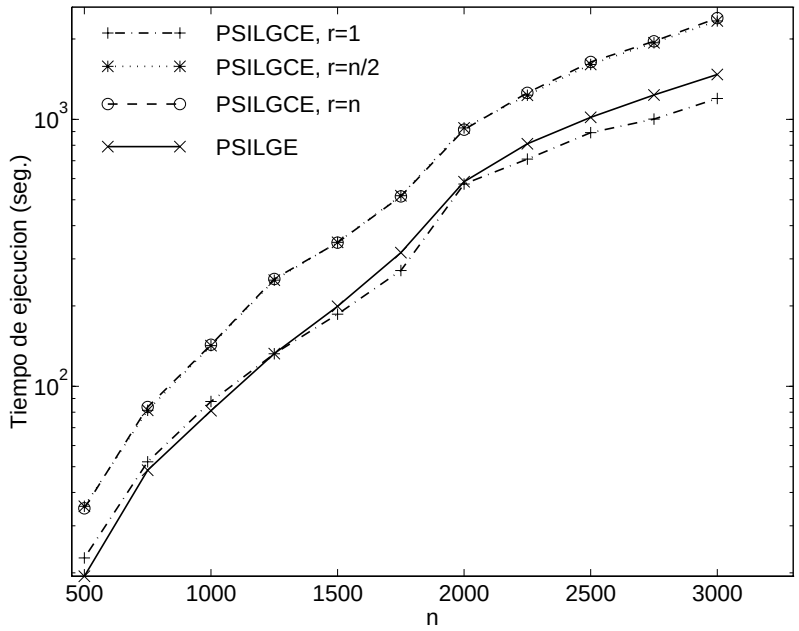


Figura 65. *Tiempo de ejecución de los algoritmos paralelos para resolver la ecuación de Lyapunov generalizada con r=1, r=n/2 y r=n en 4× 4 procesadores del SP2.*

La Tabla 13 muestra la aceleración obtenida en $\sqrt{p} \times \sqrt{p}$ procesadores para los algoritmos paralelos desarrollados respecto del algoritmo serie. La tabla muestra aceleraciones aceptables para 4 y 9 procesadores, mientras que ésta desciende considerablemente para 16 debido al pequeño tamaño de los problemas evaluados.

n	*PSILGE*			*PSILGCE (r=1)*			*PSILGCE (r=n)*		
	p=4	*p=9*	*p=16*	*p=4*	*p=9*	*p=16*	*p=4*	*p=9*	*p=16*
500	3,3	3,3	3,6	2,5	2,9	3,1	2,1	2,1	2,0
750	3,5	4,9	5,1	3,1	4,5	5,0	2,5	3,2	3,2
1000	3,3	6,5	7,7	3,2	5,7	6,9	2,6	4,3	4,5

Tabla 13. *Aceleración para los algoritmos PSILGE y PSILGCE en* $\sqrt{p} \times \sqrt{p}$ *procesadores del SP2.*

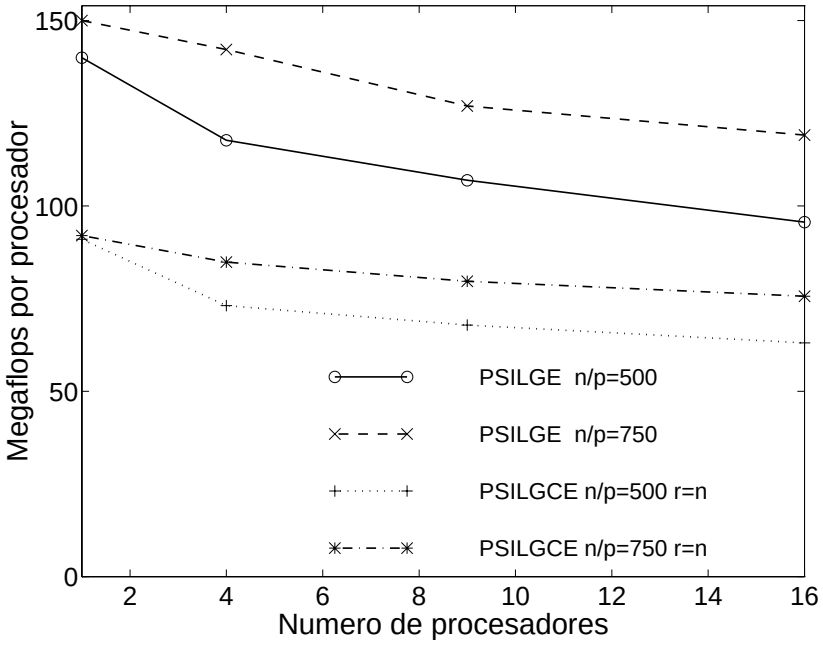


Figura 66. *Prestaciones en Mflops por procesador, utilizando isocarga de datos en los procesadores, en los algoritmos para resolver la ecuación de Lyapunov generalizada con r=n en el SP2.*

Aunque no es posible obtener los tiempos de ejecución de los algoritmos serie, y por tanto la aceleración, para problemas grandes debido a las limitaciones de memoria por procesador, si podemos analizar la escalabilidad de los algoritmos paralelos. Fijamos el tamaño del problema de modo que los requerimientos de memoria por procesador sean de *n/p*=500 y 750. Entonces obtendremos el rendimiento en Mflops por procesador dividiendo el coste teórico (en flops) del algoritmo por el tiempo de ejecución. La Figura 66 nos muestra la escalabilidad de los algoritmos. En ella podemos ver como ésta se degrada sólo ligeramente al aumentar el número de procesadores. Estos resultados son

concordantes con la degradación de los bloques básicos que utilizamos para diseñar nuestros algoritmos (factorización LU, sistemas triangulares lineales, etc.).

6.3.3 Ecuaciones de Lyapunov generalizadas acopladas

En este apartado analizaremos las prestaciones de los algoritmos para resolver las resolver las ecuaciones de Lyapunov estándar acopladas

$$AX + XA^T + BB^T = 0,$$
$$A^T Y + YA + C^T C = 0,$$

y las ecuaciones de Lyapunov generalizadas acopladas

$$AXE^T + EXA^T + BB^T = 0,$$
$$A^T YE + E^T YA + C^T C = 0.$$

En ambos casos $B \in \mathsf{R}^{n \times m}$ es la matriz de controles (entradas) y $C \in \mathsf{R}^{r \times n}$ es la matriz de salidas. En el caso estándar (generalizado) el estado del sistema viene definido por la matriz $A \in \mathsf{R}^{n \times n}$ (el par de matrices (A,E), con $A, E \in \mathsf{R}^{n \times n}$).

Hemos evaluado para nuestros experimentos tanto problemas de simple-entrada simple-salida (SISO), donde $m=r=1$, como problemas de múltiple-entrada múltiple-salida (MIMO), con $m=r=n$. El haz de matrices (A,E) se ha generado con entradas aleatorias, estando los valores propios generalizados uniformemente distribuidos en el intervalo [-10,0). Las matrices independientes B y C se han generado con entradas aleatorias.

En nuestro primer experimento comparamos las prestaciones obtenidas por los algoritmos secuenciales que resuelven las ecuaciones de Lyapunov, estándar y generalizadas, acopladas basados en los métodos directos de Bartels-Stewart [Bartels 72, Penzl 97] (denominados *BTST* para el caso estándar y *BTSTGE* para el caso generalizado) y Hammarling [Hammarling 82, Penzl 97] (denominados *HAMM* para el caso estándar y *HAMMGE* para el caso generalizado), y los algoritmos basados en la iteración de Newton para la función signo matricial propuestos en el capítulo 3. Para el diseño de los programas se han utilizado las librerías BLAS (*essl*) y LAPACK proporcionadas por IBM para el SP2.

En las Figuras 67 y 68 se muestran los tiempos de ejecución obtenidos sobre el SP2 para los algoritmos secuenciales que resuelven las ecuaciones de Lyapunov estándar acopladas asociadas a sistemas SISO y MIMO, respectivamente. Por otra parte, en las Figuras 69 y 70 se ilustran los tiempos de ejecución obtenidos para los algoritmos secuenciales que resuelven las ecuaciones de Lyapunov generalizadas acopladas asociadas a sistemas SISO y MIMO, respectivamente.

Observamos que los métodos directos son mucho más lentos que los algoritmos basados en la iteración de Newton para la función signo matricial. En particular, se han obtenido mejoras con respecto a los algoritmos directos en un factor de 10. El número de iteraciones utilizado en la iteración de Newton en todos los experimentos está comprendido entre 10 y 15. Este número de iteraciones es suficiente para alcanzar la convergencia en la mayoría de los casos.

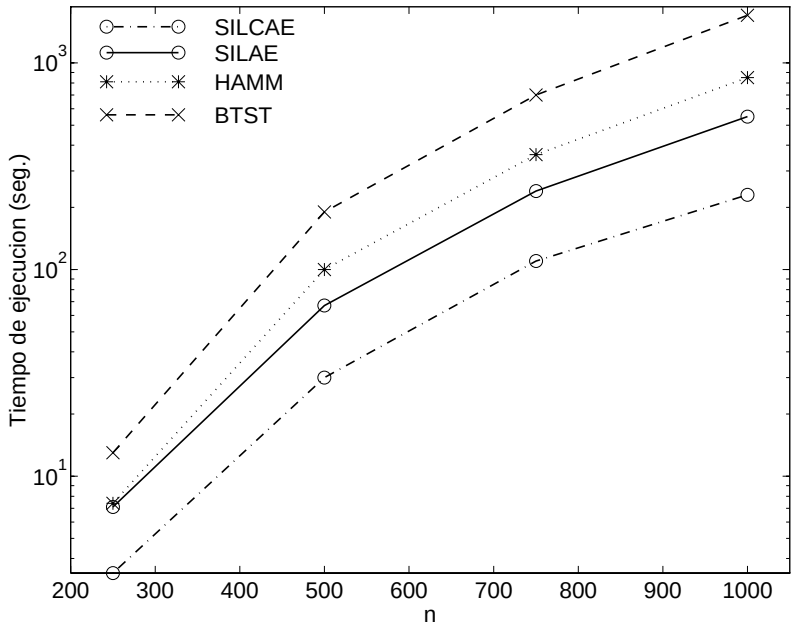


Figura 67. *Tiempo de ejecución de los algoritmos serie para resolver las ecuaciones de Lyapunov estándar acopladas para un sistema SISO.*

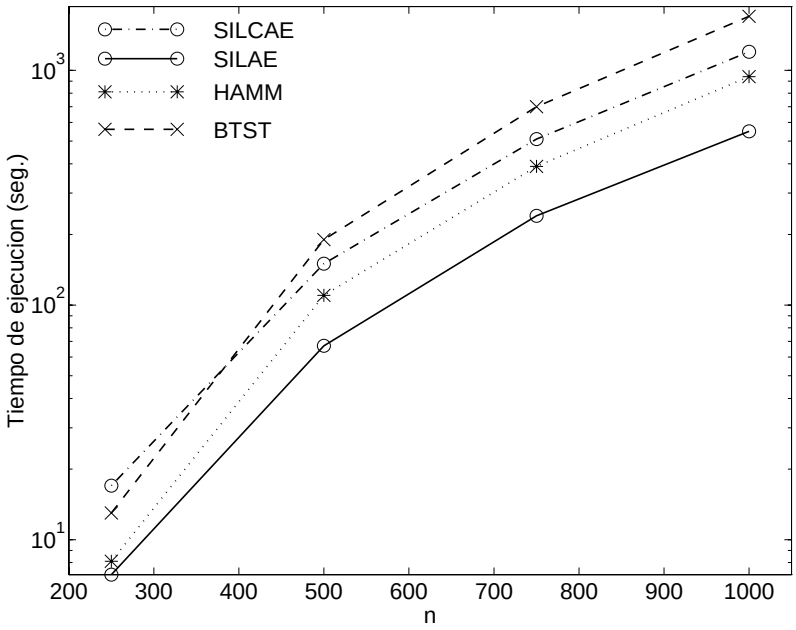


Figura 68. *Tiempo de ejecución de los algoritmos serie para resolver las ecuaciones de Lyapunov estándar acopladas para un sistema MIMO.*

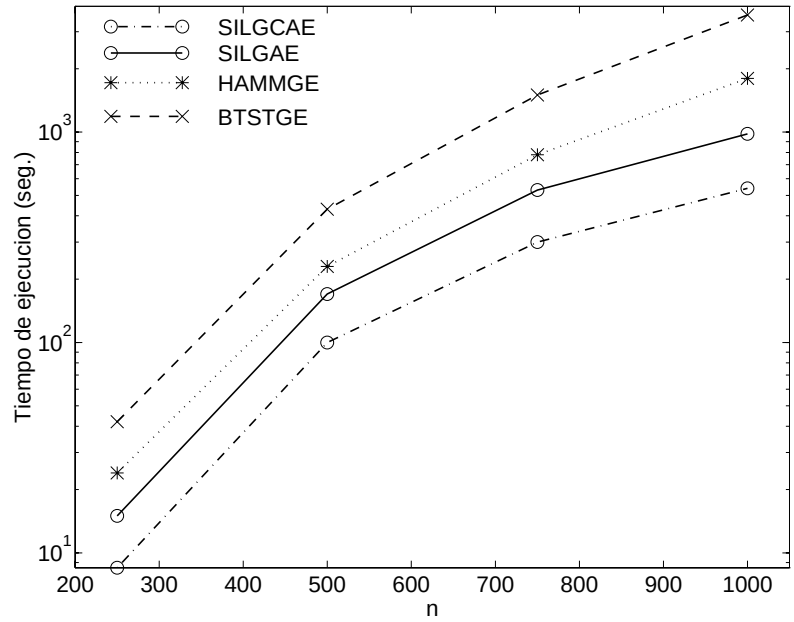


Figura 69. *Tiempo de ejecución de los algoritmos serie para resolver las ecuaciones de Lyapunov generalizadas acopladas para un sistema SISO.*

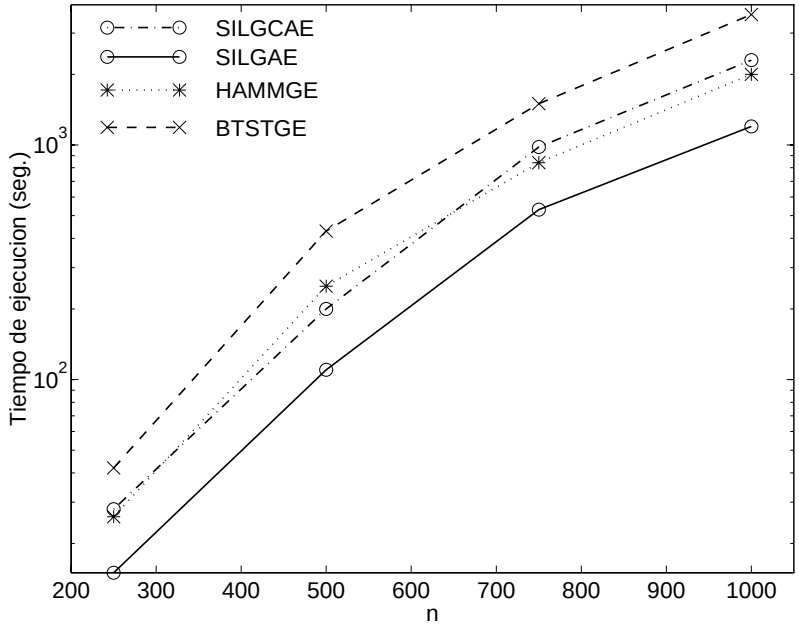


Figura 70. *Tiempo de ejecución de los algoritmos serie para resolver las ecuaciones de Lyapunov generalizadas acopladas para un sistema MIMO.*

Si analizamos los resultados obtenidos en las los sistemas SISO y los obtenidos en los sistemas MIMO, vemos como los algoritmos para obtener el factor de Cholesky, *SILCAE* y *SILGCAE*,

son más eficientes para los sistemas SISO. La eficiencia de estos algoritmos depende en gran medida del número de entradas y salidas del sistema, creciendo rápidamente su coste a medida que aumentan los valores de *m* y *r*. Los sistemas de tipo SISO aparecen frecuentemente en numerosos esquemas de discretización para sistemas de parámetros distribuidos [Rosen 95].

En el siguiente experimento analizamos las prestaciones obtenidas para los algoritmos paralelos basados en la función signo matricial. No hemos incluido en este análisis los métodos directos debido a que en la versión actual del ScaLAPACK no existen los núcleos matriciales apropiados para diseñarlos (por ejemplo, el algoritmo QZ).

En las Figuras 71 y 72 presentamos los tiempos de ejecución obtenidos para los algoritmos serie y los paralelos en una malla de $\sqrt{p} \times \sqrt{p}$ procesadores, con $\sqrt{p}$ = 2, 3, 4 procesadores. En este experimento hemos fijado la carga de datos por procesador tal que *n/p*= 750. Estas figuras muestran que para los sistemas SISO los algoritmos PSILCAE y PSILGCAE son más eficientes que los PSILAE y PSILGAE, respectivamente. Sin embargo los algoritmos que obtienen el factor de Cholesky pierden su eficiencia cuando en número de entradas y salidas es elevado.

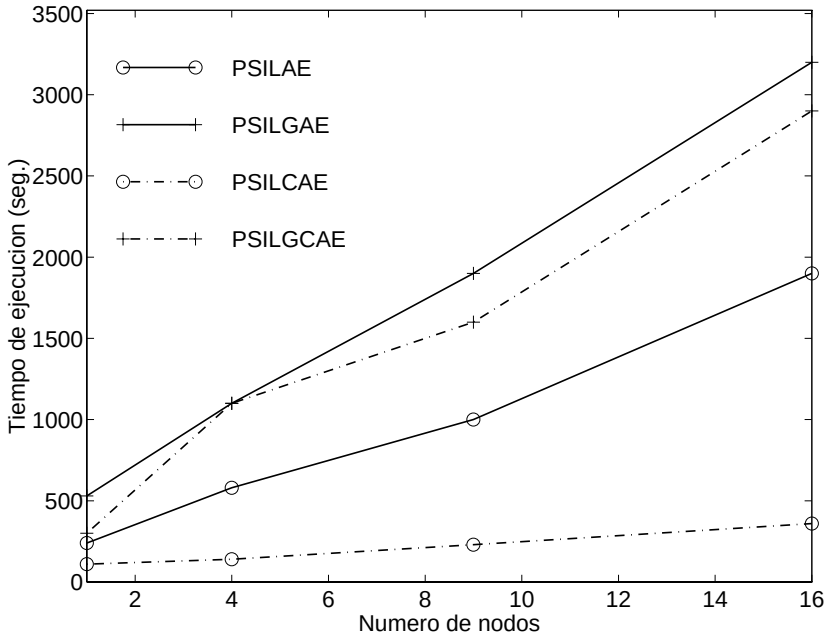


Figura 71. *Tiempo de ejecución de los algoritmos paralelos para resolver las ecuaciones de Lyapunov, estándar y generalizadas, acopladas para un sistema SISO. La carga de datos por procesador se ha fijado en n/p= 750.*

En nuestro siguiente experimento analizamos el grado de paralelismo y la escalabilidad de los algoritmos paralelos basados en la función signo matricial. Para ello, fijamos la relación de carga de datos por procesador en *n/p*= 750 (por ejemplo, para *p*= 4 tendremos un problema con *n*=3000) y calculamos los Megaflops por nodo obtenidos en los algoritmos.

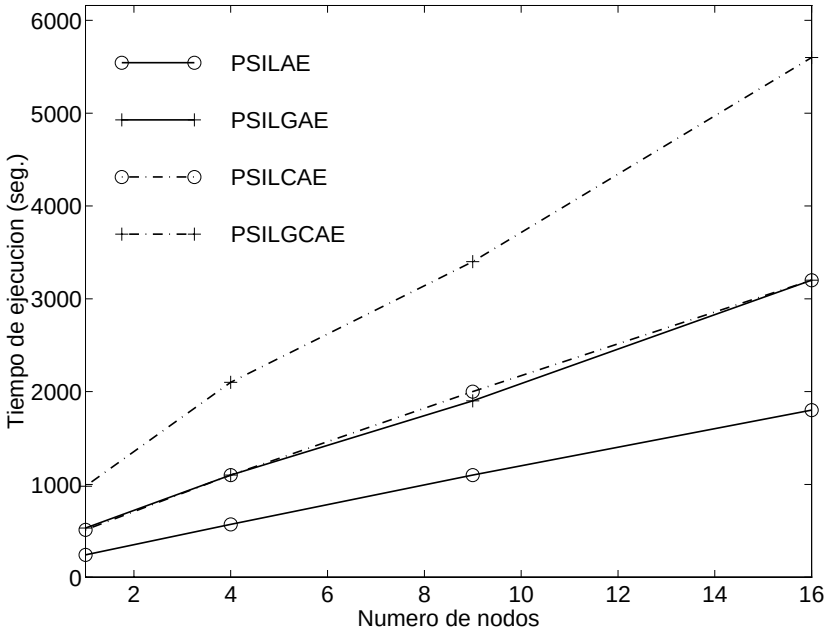


Figura 72. *Tiempo de ejecución de los algoritmos paralelos para resolver las ecuaciones de Lyapunov, estándar y generalizadas, acopladas para un sistema MIMO. La carga de datos por procesador se ha fijado en n/p= 750.*

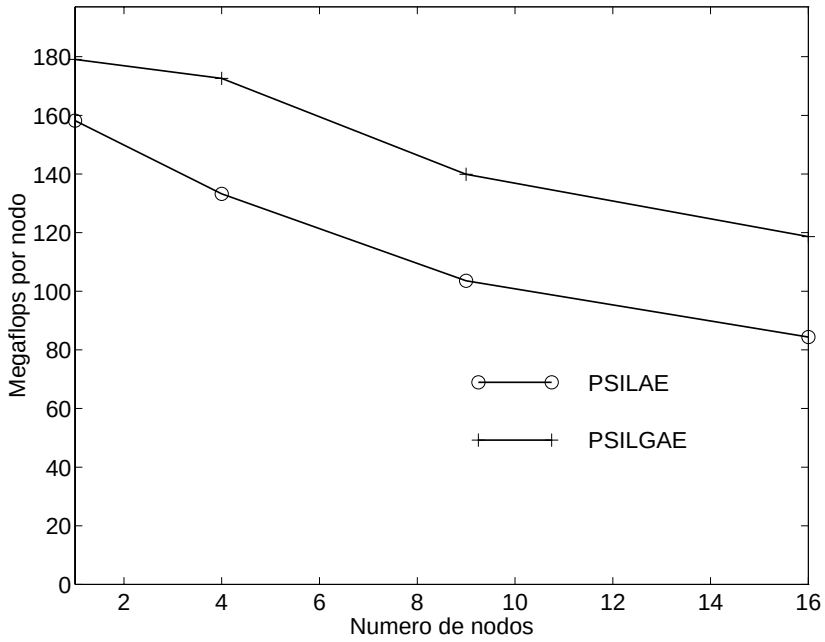


Figura 73. *Megaflops por nodo de los algoritmos paralelos para resolver las ecuaciones de Lyapunov, estándar y generalizadas, acopladas para un sistema MIMO, obteniendo la solución explícita. La carga de datos por procesador se ha fijado en n/p= 750.*

En la Figura 73 mostramos los resultados obtenidos para los algoritmos *PSILAE* y *PSILGAE*. Esta figura nos permite mostrar la escalabilidad de nuestros algoritmos y medir el sobrecoste de comunicaciones de los algoritmos paralelos.

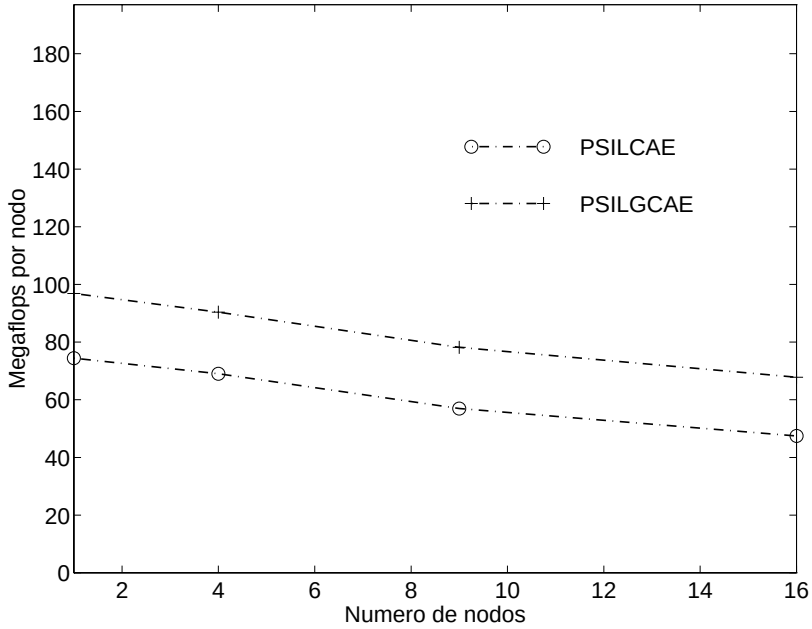


Figura 74. *Megaflops por nodo de los algoritmos paralelos para resolver las ecuaciones de Lyapunov, estándar y generalizadas, acopladas para un sistema MIMO, obteniendo el factor de Cholesky de la solución. La carga de datos por procesador se ha fijado en n/p= 750.*

En la Figura 74 se muestran los resultados obtenidos para los algoritmos *PSILCAE* y *PSILGCAE*. Observamos que las prestaciones de estos algoritmos paralelos son menores que en el caso anterior. Sin embargo, es menor el sobrecoste de comunicaciones de los algoritmos paralelos por lo que se aprecia un incremento en la escalabilidad de estos algoritmos.

6.3.4 Análisis de resultados

Se han evaluado algoritmos secuenciales por bloques para resolver, mediante la función signo matricial, tanto la ecuación de Lyapunov estándar como generalizada en tiempo continuo. Éstos han resultado ser muchos más eficientes que los métodos directos de Bartels-Stewart y Hammarling, como podemos ver en la Tabla 14 para el caso generalizado con la matriz de términos independientes independientes de rango igual (r=n) o menor (r=1) al de la matriz de estados. Además, pese a no estar probada la estabilidad numérica del método de la función signo matricial [Petkov 91, Byers 97], los resultados experimentales muestran resultados similares, en cuanto a precisión, a los obtenidos para los métodos directos numéricamente estables [Benner 97].

Algoritmos	*r=n*	*r*=1
BTST	300	300
HAMM	230	220
SILGE	47	47,3
SILGCE	100	35

Tabla 14. *Tiempo de ejecución (en segundos) de los algoritmos serie para resolver la ecuación de Lyapunov generalizada para n=500 y con r=n y r=1.*

Basados en la función signo matricial se han diseñado nuevos algoritmos paralelos para resolver la ecuación de Lyapunov en tiempo continuo (*PSILE*) y para obtener directamente el factor de Cholesky de esta ecuación (*PSILCE*), ya que en muchos problemas de control es más útil el factor de Cholesky de la solución que la solución en sí. Como generalización de los algoritmos anteriores se han presentado algoritmos paralelos para resolver la ecuación de Lyapunov generalizada (*PSILGE, PSILGCE*) basados en la iteración de Newton generalizada.

En los algoritmos paralelos *PSILCE* y *PSILGCE* se optimiza el uso de la memoria para el caso en que la matriz de términos independientes es de rango menor al de la matriz de estados, obteniéndose en este caso menores tiempos de ejecución (ver Tabla 15 para el caso generalizado) que para los algoritmos *PSILE* y *PSILGE*, respectivamente.

Algoritmos	*n*=1000	*n*=2000
PSILGE	96	841,5
PSILGCE (r=1)	92,9	756,2
PSILGCE (r=n/2)	172,2	1324,9
PSILGCE (r=n)	174,5	1353,3

Tabla 15. *Tiempo de ejecución (en segundos) de los algoritmos serie para resolver la ecuación de Lyapunov generalizada con r=1, r=n/2 y r=n en 3 × 3 procesadores del SP2.*

Además, se han implementado y evaluado algoritmos serie (*SILAE, SILCAE, SILGAE, SILGCAE*) y paralelos (*PSILAE, PSILCAE, PSILGAE, PSILGCAE*) específicos para la resolución de ecuaciones de Lyapunov, estándar y generalizadas, acopladas. En estos algoritmos se consigue una reducción importante del coste de cálculo al realizarse una sola iteración sobre la matriz de coeficientes.

Como se observa en las Tablas 16 y 17, los algoritmos serie basados en la función signo matricial siempre han obtenido mejores prestaciones que los métodos de Bartels-Stewart y Hammarlimg, tanto en el caso estándar como en el generalizado, para sistemas SISO (*r*=1). Este buen resultado se repite en los sistemas MIMO (*r=n*) para los algoritmos SILAE y SILGAE. Sin embargo, las prestaciones de los algoritmos *SILCAE* y *SILGCAE* son comparables a las conseguidas con el método de Hammarling.

Algoritmos	*r=n*	*r*=1
BTST	190	190
HAMM	110	100
SILAE	67	67
SILCAE	150	30

Tabla 16. *Tiempo de ejecución (en segundos) de los algoritmos serie para resolver las ecuaciones de Lyapunov estándar acopladas para n=500 y con r=1 y r=n.*

Algoritmos	*r=n*	*r*=1
BTSTGE	430	430
HAMMGE	250	230
SILGAE	110	170
SILGCAE	200	100

Tabla 17. *Tiempo de ejecución (en segundos) de los algoritmos serie para resolver la ecuaciones de Lyapunov generalizadas acopladas para n=500 y con r=1 y r=n.*

Por último la Tabla 18 ilustra la eficiencia por procesador obtenida en los algoritmos *PSILAE, PSILCAE, PSILGAE* y *PSILGCAE* para distintas mallas de $\sqrt{p} \times \sqrt{p}$ procesadores, donde podemos comprobar los buenos resultados de escalabilidad para estos algoritmos.

Algoritmos	*p*=4	*p*=9	*p*=16
PSILAE	0,84	0,65	0,53
PSILGAE	0,96	0,78	0,66
PSILCAE	0,92	0,76	0,64
PSILGCAE	0,93	0,80	0,70

Tabla 18. *Eficiencia de los algoritmos paralelos para resolver las ecuaciones de Lyapunov, estándar y generalizadas, acopladas respecto del algoritmo serie en diversas mallas de* $\sqrt{p} \times \sqrt{p}$ *procesadores. La carga de datos por procesador se ha fijado en n/p= 750*

Todos estos algoritmos paralelos se han implementado utilizando la librería paralela ScaLAPACK y una distribución de datos 2D. Esta librería facilita el desarrollo de algoritmos paralelos, permitiendo obtener buenas prestaciones al aumentar el tamaño del problema, buena escalabilidad al aumentar el número de procesadores y mantener la estabilidad numérica para los experimentos realizados.

6.4 Conclusiones

En este capítulo se han presentado los resultados experimentales obtenidos para los diferentes algoritmos serie, por bloques y paralelos que se han desarrollado, tanto en lo referente a sus prestaciones como, cuando ha procedido, a la precisión de éstos. Se han desarrollado algoritmos basados dos métodos completamente distintos para resolver las ecuaciones de Lyapunov, el método de Hammarling y la función signo matricial.

En el caso del método de Hammarling hemos evaluado implementaciones de algoritmos paralelos tanto para memoria compartida como para memoria distribuida que resuelven la ecuación reducida de Lyapunov en tiempo discreto.

Para multiprocesadores con memoria compartida se ha constatado que los nuevos algoritmos obtienen las máximas prestaciones para sistemas con procesadores escalares y vectoriales utilizando técnicas de ejecución por frente de onda, resolución por bloques de columnas y solapamiento de resolución entre bloques columna.

En el caso de memoria distribuida se han evaluado algoritmos con distribuciones frente de onda y cíclicas, así como algunas de las mejoras propuestas basadas en la combinación de las dos técnicas anteriores con muy buenos resultados. Los mejores resultados para todos lo tamaños de problema se han obtenido para los algoritmos cíclicos con frente de onda *PHCWFU* .

Se han evaluado los algoritmos secuenciales por bloques y paralelos basados en el método de la función signo matricial para resolver las ecuación de Lyapunov, tanto estándar como generalizada, en tiempo continuo. Tanto en los algoritmos en los que se calcula la solución de la ecuación como en los que se calcula el factor de Cholesky de ésta se han obtenido en general mejores prestaciones que para los métodos directos de Bartels-Stewart y Hammarling, con resultados de precisión similares a éstos últimos.

En los algoritmos implementados para la resolución de las ecuaciones de Lyapunov acopladas se ha comprobado la reducción del coste de cálculo al realizarse una sola iteración sobre la matriz de coeficientes y la idoneidad de los algoritmos que obtienen directamente el factor de Cholesky para sistemas SISO.

Por último, hemos visto como todos los algoritmos basados en la función signo matricial implementados muestran un alto grado de escalabilidad. Las causas de ello son la propia naturaleza de los algoritmos diseñados, la utilización de la librería ScaLAPACK, la distribución de los datos y la arquitectura del computador utilizado en nuestros experimentos.

En el análisis experimental se han utilizado diferentes tipos de computadores lo que permite que las conclusiones tengan un carácter más general. Los algoritmos secuenciales y por bloques se han evaluado sobre el procesador con capacidad vectorial del Alliant FX/80 y los procesadores superescalares DEC Alfa AXP 6220 del Cray T3D, MIPS R10000 del SGI Power Challenge e IBM RS/6000 del IBM SP2. En cuanto a los algoritmos paralelos, para el método de Hammarling sobre memoria compartida se ha utilizado el Alliant FX/80 y para memoria distribuida se han utilizado el

SGI Power Challenge y el Cray T3D, utilizando las librerías de paso de mensajes PVM 3.0 y MPI. Los algoritmos paralelos basados en la función signo matricial se han evaluado en el IBM SP2, utilizando la librería paralela ScaLAPACK.

Capítulo 7

Conclusiones y líneas futuras

7.1 Conclusiones

El objetivo de esta tesis se enmarca dentro del estudio de la paralelización de los algoritmos utilizados para la reducción de modelos en el espacio de estados de SLC invariantes en el tiempo. Este objetivo se ha materializado en el diseño y desarrollo de nuevos algoritmos secuenciales y paralelos para la resolución de las ecuaciones de Lyapunov, estándar y generalizadas, que aparecen como piedra angular de los métodos más importantes utilizados para obtener el modelo reducido de un SDL invariante en el tiempo. Al mismo tiempo se ha estudiado el comportamiento de estos algoritmos en algunos de los computadores de altas prestaciones actuales más potentes y se han iniciado nuevas líneas de investigación en este campo.

A continuación se resumen los aspectos más importantes del problema de reducción de modelos mediante truncamiento en el espacio de estados y se describen las conclusiones más relevantes acerca de las diversas aproximaciones para la resolución paralela de las ecuaciones de Lyapunov presentadas en esta memoria.

7.1.1 Redución de modelos: truncamiento en el espacio de estados

La reducción de modelos supone un compromiso entre el orden del modelo y la adecuación de éste a las características de nuestro sistema. Como apunta Moore [Moore 81] no existe ningún método de reducción que sirva para todos los sistemas, dado que muchas de las características del sistema dependen en gran medida de la aplicación.

En este trabajo hemos descrito algunas de las técnicas utilizadas en la reducción de modelos mediante truncamiento en el espacio de estados y nos hemos centrado en dos de las más importantes y representativas para su estudio detallado. Estas técnicas no se utilizan de forma exclusiva, sino que a menudo se combinan dos o más técnicas de reducción de modelos para conseguir un modelo reducido.

Los métodos de reducción basados en realizaciones balanceadas [Moore 81, Laub 87, Tombs 87] y la forma real de Schur [Safonov 89] son de especial interés por estar basados en transformaciones ortogonales, lo que les otorga una importante estabilidad numérica. Por otra parte el modelo reducido que generan estos métodos mantiene casi inalteradas muchas de las propiedades del sistema original.

La resolución de la ecuaciones de Lyapunov juega un papel clave en los métodos de reducción que se han tratado con mayor detalle. Estas ecuaciones deben resolverse para obtener las matrices Gramian o sus factores de Cholesky. Además, las ecuaciones que hay que resolver están acopladas, es decir, poseen una matriz, la de coeficientes, que las relaciona directamente. En los SDL en lazo cerrado el balanceado del sistema implica la resolución de dos ecuaciones algebraicas de Riccati acopladas [Jonkheere 83]. Uno de los métodos más fiables para resolver estas ecuaciones es el método de Newton (o iteración de Kleinmann) [Kleinmann 68, Benner 95] que requiere la resolución de una ecuación de Lyapunov en cada paso.

Por todo ello será importante el desarrollo de algoritmos que resuelvan estas ecuaciones con la mayor exactitud y en el menor tiempo posible. El presente trabajo se ha dedicado a alcanzar estos objetivos mediante la utilización de los métodos numéricos más adecuados en cada caso y el uso de las técnicas de programación paralela y distribuida más actuales.

7.1.2 Métodos para la resolución de las ecuaciones de Lyapunov

La resolución de las ecuaciones de Lyapunov se aborda frecuentemente mediante los métodos de Bartels-Stewart [Bartels 72] y Hammarling [Hammarling 82, Hammarling 91]. Estos son métodos con una primera parte iterativa y una segunda parte directa, siendo ambos numéricamente estables. Éstos requieren de una primera etapa iterativa muy costosa en la que la matriz de coeficientes se factoriza en la forma real de Schur mediante el algoritmo iterativo QR [Francis 61]. A continuación se obtiene la solución mediante una etapa directa compuesta por descomposiciones LU y resolución de sistemas triangulares lineales.

El método de Hammarling es especialmente interesante pues proporciona directamente el factor de Cholesky de la solución de la ecuación de Lyapunov. En numerosas aplicaciones de control es este factor de Cholesky el resultado que se necesita, mientras que resolver la ecuación de Lyapunov para la solución explícita X y obtener a partir de esta el factor de Cholesky puede producir una pérdida de precisión en los resultados. Además, el algoritmo de Hammarling trabaja directamente sobre el factor C o B (matrices de entradas o de control), por lo que no es necesario formar explícitamente el producto $C^T C$ o $B^T B$, evitando de este modo una nueva posible pérdida en la precisión de los resultados.

El método de la función signo matricial fue introducido por [Roberts 80] para resolver numéricamente la ecuación algebraica de Riccati y está basado en la iteración de Newton. No es un método numéricamente estable, pero en la práctica la precisión obtenida es similar a la de aquellos. Recientemente se han desarrollado algoritmos que aplican este método a la ecuación de Lyapunov [Benner 97].

Una de las principales características de este método es que requiere operaciones como productos de matrices, resolución de sistemas triangulares e inversión de matrices. Estas operaciones están muy optimizadas en gran número de librerías computacionales, tanto secuenciales como paralelas, lo que permite un alto rendimiento a la hora de la implementación del código. Por otra parte, el método de la función signo permite una fácil generalización para la resolución de ecuaciones de Lyapunov generalizadas o su modificación para la obtención del factor de Cholesky de la solución.

7.1.3 Computadores de altas prestaciones

La programación de los multiprocesadores con memoria compartida no resulta muy diferente de la programación secuencial. Si se quiere obtener el máximo rendimiento del computador hay que diseñar el código para optimizar el uso de los recursos del computador (registros, unidades de cálculo, antememoria, etc.) de forma muy similar a como se haría en un computador monoprocesador, y aplicar las optimizaciones y directivas de compilación más adecuadas en cada caso.

En cambio la programación de los multicomputadores, si se quiere aprovechar al máximo los recursos de la máquina, debe realizarse bajo un entorno de paso de mensajes, presentando este modelo una programación mucho más compleja.

Uno de los parámetros principales que determina las prestaciones de los algoritmos paralelos sobre multicomputadores es la distribución de los datos. Tres han sido las distribuciones propuestas, estas son, distribución por bloques de columnas (o filas), distribución cíclica por bloques de columnas (o filas) y distribución toroidal 2-D.

La distribución por bloques de filas (columnas) es válida cuando se aplica algoritmos de frente de onda, aunque ello obligue a una redistribución de los datos durante la resolución del problema. La distribución cíclica por bloques de filas (columnas) resulta especialmente apropiada para problemas de dimensión moderada. La principal ventaja de la distribución toroidal 2-D es su escalabilidad, que la hace adecuada para resolver problemas de gran dimensión con un elevado número de procesadores.

La combinación de una distribución de datos apropiada, y un uso coherente de las subrutinas de comunicación, facilita en gran medida el desarrollo de algoritmos paralelos. En nuestro trabajo hemos utilizado las librerías de comunicaciones PVM y MPI.

El estudio de la arquitectura y de las prestaciones de la máquina paralela sobre la cual se desean desarrollar los algoritmos es una de las tareas básicas en la programación. Así, se han estudiado las características más importantes de las máquinas utilizadas durante la realización de esta tesis.

7.1.4 Algoritmos paralelos basados en el método de Hammarling

En este trabajo se han diseñado e implementado diversos algoritmos paralelos para la resolución de las ecuaciones de Lyapunov y se ha realizado el análisis teórico de costes de los más representativos.

Basados en el método de Hammarling [Hammarling 82, Hammarling 91] se han diseñado nuevos algoritmos paralelos de grano fino y medio para resolver las ecuaciones de Lyapunov en tiempo discreto en multiprocesadores con memoria compartida. Para obtener las máximas prestaciones posibles, se han aplicado técnicas de frente de onda, resolución por bloques de columnas y solapamiento entre la resolución del fin de un bloque de columnas y el principio del siguiente.

Los algoritmos de grano de resolución fino (*PBSHWF*) han mostrado ser muy adecuados para multiprocesadores con procesadores escalares y superescalares. La eficiencia del algoritmo *PBSHWF* crece muy rápidamente, manteniéndose alrededor de 0,8 para problemas de gran tamaño.

Los algoritmos de grano fino (*PBSHWF*) no obtienen buenas eficiencias en multiprocesadores con unidades vectoriales. A fin de aprovechar las prestaciones adicionales que proporcionan los sistemas con unidades vectoriales se han diseñado algoritmos paralelos de grano medio por bloques de columnas (*PBSHWM*). Estos algoritmos no obtienen buenas prestaciones para tamaños de problema pequeños ya que el grano de resolución es menor, lo que redunda una reducción del paralelismo del problema. En nuestros experimentos el algoritmo *PBSHWF* obtiene, para problemas de orden n=100, una eficiencia de 0,35 mientras que en el algoritmo *PBSHWM* la eficiencia es de 0,24. Sin embargo, para problemas de tamaño mayor, la eficiencia del algoritmo *PBSHWF* no aumenta, pero si la del algoritmo *PBSHWM*, alcanzando eficiencias de 0,56.

Como el tamaño de las matrices del problema se reduce en cada etapa, y dado que los algoritmos de grano fino ofrecen mejores prestaciones para tamaños de problemas pequeños, se han implementado algoritmos que adaptan el grano de resolución basados en la combinación de los algoritmos con granos de resolución fino y medio (*PBSHWC*), consiguiéndose buenas eficiencias para un amplio rango de tamaños de ecuaciones.

Así mismo, basados en el método de Hammarling, se han desarrollado nuevos algoritmos paralelos para sistemas multiprocesadores con memoria distribuida que utilizan paso de mensajes para las comunicaciones. En estos algoritmos se han aplicado y extendido las técnicas utilizadas para la resolución de sistemas triangulares lineales [Chamberlain 86, Eisensat 88, Heath 88, Li 88].

Se han diseñado algoritmos paralelos frente de onda con granos de resolución fino (*PDHWF*), medio (*PDHWM*) y grueso (o por bloques) (*PDHWB*). Los mejores resultados se han obtenido para los algoritmos por bloques (*PDHWB*), siendo excelentes las eficiencias al aumentar el tamaño del problema. De echo se han alcanzado, para tamaños de problema elevados, eficiencias superiores a la unidad (*superspeedup*). Esto es debido a que los algoritmos frente de onda diseñados poseen tanto un excelente uso de la antememoria, como un alto solapamiento entre el cálculo aritmético y las comunicaciones.

Además, siguiendo el razonamiento utilizado en memoria compartida, se han desarrollado algoritmos que adaptan dinámicamente del grano de resolución a medida que se va resolviendo el problema (*PADHWB*). Esta técnica da como resultado una mejora de las prestaciones y una independencia en la elección del bloque de resolución del tamaño del problema y del número de procesadores. La elección del bloque de resolución es crítica cuando aumenta el número de procesadores en el algoritmo *PDHWB*, obteniéndose las mejores prestaciones sólo para un determinado tamaño de bloque. Con el algoritmo *PDHWB* conseguimos las mejores prestaciones del algoritmo para un conjunto de tamaños de bloque de resolución mayor.

Por otra parte, se han diseñado algoritmos paralelos cíclicos con grano de resolución fino (*PHCF, PHCF2*), medio (*PHCM, PHCMR*) y grueso (*PHCB*). En este tipo de algoritmos la aparición de esperas degrada las prestaciones de los algoritmos. Para reducir estos tiempos de espera se han propuesto mejoras basadas en la optimización del envío de los mensajes (*PHCFU, PHCFG*), pero dada la naturaleza del método de Hammarling y las características de los computadores actuales, ya comentadas en este trabajo, las prestaciones obtenidas son muy bajas.

Para aprovechar las ventajas de los algoritmos frente de onda y cíclicos, se han propuesto nuevos algoritmos con diferente grano de resolución basados en la combinación de estas dos técnicas. Así, se han diseñado algoritmos cíclicos de frente de onda (*PHCWF, PHCWFU, PHCWFG, PHCWFN*) que mejoran considerablemente las prestaciones de los algoritmos cíclicos anteriores. En particular, para el algoritmo *PHCWFU* se obtienen mejores prestaciones que en los algoritmos de frente de onda al incrementar el número de procesadores.

Además, siguiendo el razonamiento utilizado en memoria compartida, se han propuesto algoritmos que adaptan dinámicamente del grano de resolución a medida que se va resolviendo el problema (*PAHCWFU*).

7.1.5 Algoritmos paralelos basados en la función signo matricial

Se han evaluado algoritmos secuenciales por bloques para resolver, mediante la función signo matricial, tanto la ecuación de Lyapunov estándar como generalizada en tiempo continuo. Éstos han resultado ser muchos más eficientes que los algoritmos directos de Bartels-Stewart y Hammarling. Además, pese a no estar probada la estabilidad numérica del método de la función signo matricial [Petkov 91, Byers 97], los resultados experimentales muestran resultados en cuanto a precisión similares a los obtenidos para los métodos directos numéricamente estables [Benner 97].

Basados en la función signo matricial se han diseñado nuevos algoritmos paralelos para resolver la ecuación de Lyapunov en tiempo continuo (*PSILE*).

También se han desarrollado algoritmos para obtener directamente el factor de Cholesky de esta ecuación (*PSILCE*), ya que en muchos problemas de control es más útil el factor de Cholesky de la solución que la solución en sí.

Como generalización de los algoritmos anteriores se han presentado algoritmos paralelos para resolver la ecuación de Lyapunov generalizada (*PSILGE, PSILGCE*) basados en la iteración de Newton generalizada.

En los algoritmos *PSILCE* y *PSILGCE* se optimiza el uso de la memoria para el caso en que la matriz de términos independientes es de rango menor al de la matriz de estados, obteniéndose en estos casos menores tiempos de ejecución que para los algoritmos *PSILE* y *PSILGE*, respectivamente.

Además, se han diseñado algoritmos serie (*SILAE, SILCAE, SILGAE, SILGCAE*) y paralelos (*PSILAE, PSILCAE, PSILGAE, PSILGCAE*) específicos para la resolución de ecuaciones de Lyapunov, estándar y generalizadas, acopladas. En estos algoritmos se consigue una reducción importante del coste de cálculo al realizarse una sola iteración sobre la matriz de coeficientes.

En los algoritmos serie basados en la función signo matricial siempre se han obtenido mejores prestaciones que para los algoritmos de Bartels-Stewart y Hammarlimg, tanto en el caso estándar como en el generalizado, para sistemas SISO. Este buen resultado se repite en los sistemas MIMO para los algoritmos SILAE y SILGAE. Sin embargo, las prestaciones de los algoritmos *SILCAE* y *SILGCAE* son comparables a los conseguidos con el algoritmo de Hammarling.

Todos estos algoritmos paralelos se han implementado utilizando la librería paralela ScaLAPACK y una distribución de datos 2D. Esta librería facilita el desarrollo de algoritmos paralelos. Las características más importantes obtenidas han sido sus buenas prestaciones al aumentar el tamaño del problema y la escalabilidad al aumentar el número de procesadores, manteniendo al mismo tiempo estabilidad numérica para los experimentos realizados.

7.2 Publicaciones y trabajos en el marco de la Tesis

Durante el desarrollo de esta tesis se han obtenido un conjunto de resultados que han sido difundidos en publicaciones y congresos de carácter nacional e internacional.

En concreto, han sido publicados o están pendientes de ser publicados los siguientes artículos en revistas de carácter internacional:

- "Solving discrete-time Lyapunov equations for the Cholesky factor on a shared memory multiprocessor", José M. Claver, Vicente Hernández, Enrique S. Quintana. Parallel Processing Letters, Vol. 6, No. 3, pp. 365-376, 1996.

- "Parallel wavefront algorithms solving Lyapunov equations for the Cholesky factor on message passing multiprocessors", José M. Claver, Vicente Hernández. The Journal of Supercomputing (aceptado y pendiente de publicación), 1998.

- "Parallel distributed solvers for large stable generalized Lyapunov equations", Peter Benner, José M. Claver, Enrique S. Quintana. Parallel Processing Letters (aceptado y pendiente de publicación), 1998.

- "Parallel adaptive wavefront algorithms solving Lyapunov equations for the Cholesky factor on message passing multiprocessors", José M. Claver, Vicente Hernández. Concurrency: PRACTICE AND EXPERIENCE (en revisión).

Además, próximamente aparecerá publicado en actas de congreso el artículo titulado:

- "Efficient solution of coupled Lyapunov equations via matrix sign function iteration", Peter Benner, José M. Claver, Enrique S. Quintana. 3rd Portuguese Conference on Automatic Control - Controlo 98, Coimbra (Portugal).

Por último se han realizado las siguientes comunicaciones en congresos internacionales:

- "Fine and medium grain parallel algoritms for the Lyapunov equations on a shared memory multiprocessor", José M. Claver. 3rd International Conference on Numerical Methods and Aplications, Sofia (Bulgaria), 1994.

- "A parallel approach to the balanced realization problem in control linear systems", José M. Claver, Vicente Hernández, Enrique S. Quintana. GEPPCOM Workshop II, Peñíscola, 1995.

Finalmente, este trabajo de investigación ha generado un total de 4 informes técnicos en el departamento de Informática de la Universitat Jaume I de Castellón.

7.3 Líneas futuras de investigación

En problemas de control asociados a sistemas lineales de muy gran dimensión suelen aparecer ecuaciones de Lyapunov con matrices de coeficientes dispersas (gran número de elementos nulos). Los métodos de resolución habituales (Bartels-Stewart, Hammarling, función signo matricial, etc.) no son válidos para estas ecuaciones pues obtienen una matriz resultado densa que resulta imposible almacenar debido a la propia dimensión del problema. Para paliar este problema se han propuesto algoritmos iterativos para resolver este tipo de ecuaciones que preservan la dispersión del problema en [Hodel 88; Hodel 92, Penzl 98].

En los métodos para la obtención de realizaciones balanceadas de sistemas lineales simétricos de simple entrada/salida [Fernando 83, Fortuna 88] y desarrollados con posterioridad para sistemas de múltiple entrada/salida, aparecen nuevas ecuaciones matriciales asociadas a estas técnicas. Éstas se

denominan ecuaciones matriciales de Lyapunov cruzadas [Fernando 85], si están asociadas a sistemas de control en lazo abierto, y ecuaciones matriciales de Riccati cruzadas [Fortuna 88], si están asociadas a sistemas de control en lazo cerrado. Para el caso de la realización balanceada en lazo abierto, de una de estas ecuaciones se puede obtener la misma información que antes se obtenía de las dos ecuaciones de Lyapunov asociadas a los Gramianos de controlabilidad y observabilidad.

El método de la función signo matricial puede aplicarse a la resolución de la ecuación de Lyapunov en tiempo discreto mediante la transformación de Cayley, que convierte esta ecuación en una ecuación de Lyapunov en tiempo discreto. Sin embargo, el uso de esta transformación requiere el cálculo de la inversa de la matriz de coeficientes, que en algunos casos puede introducir grandes errores numéricos. Por ello es conveniente utilizar métodos iterativos específicos para la resolución de este tipo de ecuaciones como, por ejemplo, los basados en el cálculo de la función disco matricial [Benner 96].

La aplicación de los algoritmos secuenciales y paralelos a problemas de reducción de modelos pasa por el desarrollo de núcleos computacionales numéricos para resolver las etapas posteriores al cálculo de las matrices de Gramian. En concreto, en el método de Laub [Laub 87] es necesario obtener la descomposición en valores singulares de un producto de matrices. En [Heath 86, Fernando 87] se han propuesto algoritmos para el cálculo de este tipo de descomposiciones sin la realización explícita del producto. La paralelización de estos algoritmos es un tema abierto que se ha estudiado recientemente, por ejemplo, en [Bai 94, Mollar 96].

En diversa bibliografía pueden encontrarse descripciones completas de aplicaciones de control reales donde es necesario proceder a una reducción de modelos [Fortuna 92, Rosen 95]. Estas aplicaciones corresponden a problemas de orden demasiado pequeño para que resulte interesante el empleo de algoritmos paralelos eficientes. En cuanto a problemas reales de mayor dimensión, hemos podido constatar que resulta difícil que las empresas, que disponen de los datos del problema (el modelo de la aplicación), estén dispuestas a compartir los mismos y a hacerlos públicos.

7.4 Agradecimientos

La realización de esta tesis se ha enmarcado y ha disfrutado de las siguientes ayudas para la investigación provenientes de los siguientes proyectos de investigación:

- Proyecto MI-25.041/92. Algoritmos paralelos en problemas de control de sistemas y procesamiento de la señal, Univ. Jaume I y la Fundació Caixa Castelló, de 1/1/92 al 30/3/93. Investigador principal: Gloria Martínez Vidal.

- Proyecto A-35-IN/93. Algorísmica i Programació Paral·lela, Universitat Jaume I y la Fundació Bancaixa, de 1/7/93 a 30/7/94. Investigador principal: Angel P. del Pobil i Ferré y José Manuel Claver Iborra.

- Project de Creation d'un Reseau de Centres Scientifiques pour l'Aide au Developement Industriel par la Formation, RECITE Programme, de 1/1/93 a 31/12/94. Investigador principal: Vicente Hernández García.

- Proyecto ESPRIT No. 9072 - GEPPCOM: General Purpose Parallel Computing, Comunidad Europea (Programa ESPRIT III), de 1/10/94 a 1/10/96. Investigador principal: Vicente Hernández García.

- Proyecto CICYT No. TIC 96-1062-C03-03: Algoritmos paralelos para el cálculo de valores propios y la resolución de ecuaciones matriciales en problemas de ingeniería, CICYT, de 1/7/96 a 30/6/99. Investigador principal: Vicente Hernández García.

- Red Temática Brite/Euram III RTD NICONET: Network for development and evaluation of numerically reliable software in control engineering and its implementation in production technologies, Comunidad Europea, 1/1/98 a 1/12/99. Investigador principal: Sabine Van Huffel.

Así mismo, durante este tiempo se ha realizado una estancia de 2 meses en el Centro Europeo para la Investigación y la Formación en Cálculo Científico de Toulouse (Francia) de enero a febrero de 1995. Bolsa de viaje concedida por la Consellería de Educación i Ciencia de la Generalitat Valenciana dentro del programa RECITE.

Bibliografía

[Ahues 97] M. Ahues, F. Tisseur. *A new deflation criterion for the QR algorithm*. LAPACK working note LAWN-122. 1997.

[Alliant 87] Alliant Computer System Corp. *FXI Series architecture manual*. Alliant Computer System Corp. Documentation, Littleton, EEUU, 1987.

[Alliant 88] Alliant Computer System Corp. *FXI Series common library*. Alliant Computer System Corp. Documentation, Littleton, EEUU, 1988.

[Alliant 89] Alliant Computer System Corp. *FX/Fortran language manual*. Alliant Computer System Corp. Documentation, Littleton, EEUU, 1989.

[Anderson 92] E. Anderson et al. *LAPACK users' guide.* SIAM, 1992.

[Aoki 78] M. Aoki. *Some aproximation methods for estimation and control of large scale systems*. IEEE Trans. on Automatic Control, Vol. 23, Nº 2, pp. 173-182. 1978.

[Bai-89] Z. Bai, J. Demmel. *On a block implementation of Hessenberg Multishift QR iteration*. International Journal of High Speed Computing, Vol. 1, pp. 97-112. 1989.

[Bai 92] Z. Bai, J. Demmel. *Design of a parallel nonsymmetric eigenroutine toolbox, part I.* Computer Science Division Report UCB//CSD-92-718, University of California at Berkeley, 1992.

[Barnett 75] S. Barnett. *Matrices in control theory*. Van Nostrand Reinhold, Londres, Reino Unido, 1975.

[Bartels 72] R. Bartels, G. W. Stewart. *Algorithm 432: The solution of the matrix equation AX-XB=C.* Communications of the ACM, 15, pp. 820-826,1972.

[Beavers 74] A. Beavers, E. Denman. *A new solution method for quadratic matrix equations*. Math. Biosci., 20, pp. 135-143, 1974.

[Benner 94] P. Benner, R. Byers. *Step size control for Newton's method applied to algebraic Riccati equations.* 5th SIAM Conference on Applied Linear Algebra, 1994.

[Benner 97] P. Benner, E. S. Quintana-Ortí. *Solving stable generalized Lyapunov equations with the matrix sign function.* Technischen Universität Chemnitz, Alemania, Technical Report SFB393/97-23, 1997.

[Benner 98] P. Benner, J. M. Claver, E. S. Quintana-Ortí. *Parallel distributed solvers for large stable generalized Lyapunov equations*. Parallel Processing Letters (aceptado y pendiente de publicación), 1998.

[Benner 98] P. Benner, J. M. Claver, E. S. Quintana-Ortí. *Efficient solution of coupled Lyapunov equations via matrix sign funtion iteration*. 3rd Poutuguese Conference on Automatic Control - Controlo 98 (aceptado y pendiente de publicación), 1998.

[Berry 86] M. Berry, & A. Sameh. *Multiprocessor Jacobi algorithms for dense symmetric eingenvalue and singular value decompositions*, ACM-Trans. on Math. Soft., 1986.

[Brent 85] Brent, R. & Luk, F.: *The solution of singular value and symmetric eigenvalue problem*, SIAM J. Sci. Stat. Comp., Vol. 8, No. 2, March 1987.

[Bischof 87] C. H. Bischof, C. Van Loan. *The WY representation of products of Householder matrices*. SIAM J. on Sci. & Stat. Comp., 8, pp. 2-13, 1987.

[Blackford 97] L. S. Blackford *et al.*. *ScaLAPACK users' guide*. SIAM, Phil., PA, 1997.

[Byers 83] R. Byers. *Hamiltonian and symplectic algorithms for the algebraic Riccati equation.* Ph.D. Thesis, Dept. of Computer Science, Cornell University, CorneU, EEUU, 1983.

[Byers 86] R. Byers. *Numerical stability and stability in matrix sign function based algorithms*. En Computational and Combinatorial Methods in Systems Theory (Eds. C.I. Byrnes, A. Lindquist), pp. 185-199, Elsevier, North Holland, 1986.

[Byers 87] R. Byers. *Solving the algebraic Riccati equation with the matrix sign function*. Lin. Algebra & Its Appl., 85, pp. 267-279, 1987.

[Byers 97] R. Byers, C. He, V. Mehrmann. *The matrix sign function and the computation of invariant subspaces.* SIAM J. on Matrix Analysis & Appl., Vol. 18(3), pp. 615-632, 1997.

[Chamberlain 86] R. Chamberlain. *An algorithm for LU factorization with partial pivoting on a Hypercube.* Technical Report CCS 86/1, Dept. of Science and Technology, Chr. Michelson Institute, 1986.

[Claver 94] J. M. Claver, V. Hernández. *Algoritmos paralelos de grano fino y medio para resolver la ecuación de Lyapunov en un multiprocesador con memoria compartida*, Departament d'Informàtica, Universitat Jaume I, Technical Report DI 08-11/94, 1994

[Claver 96] J. M. Claver, V. Hernández, E. S. Quintana. *Solving discrete-time Lyapunov equations for the Cholesky factor on shared memory multiprocessors*, Parallel Processing Letters, Vol 6, No 3, pp. 365-376, 1996.

[Claver 97a] J. M. Claver, V. Hernández. *Algoritmos frente de onda para el cálculo de del factor de Cholesky de la ecuación de Lyapunov en multiprocesadores con paso de mensajes*, Departament d'Informàtica, Universitat Jaume I, Technical Report DI 01-04/97, 1997

[Claver 97b] J. M. Claver, V. Hernández. *Parallel adaptive wavefront algorithms solving Lyapunov equations for the Cholesky factor on message passing multiprocesors*. Concurrency: Practice and experience, John Wiley and Sons (pendiente de revisión).

[Claver 98] J. M. Claver, V. Hernández. *Parallel wavefront algorithms solving Lyapunov equations for the Cholesky factor on message passing multiprocesors*, The Journal of Supercomputing, Kluwer Academic Publishers, (aceptado y pdte. de publicación), 1998.

[Cray 93a] Cray Research Inc. *PVM and HENCE programmers manual*. (Cray Research Software Documentation, SR-2501, Mendota Heights, EEUU, 1993).

[Cray 93b] Cray Research Inc. *Cray MPP Fortran reference manual*. (Cray Research Software Documentation, SR-2504, Mendota Heights, EEUU, 1993).

[Cray 93c] Cray Research Inc. *Cray T3D system architecture overview manual.* (Cray Research Software Documentation, HR-04033, Mendota Heights, EEUU, 1993).

[Dally 87] W. J. Dally, C. L. Seitz. *Deadlock-free message routing in multiprocessor interconnection networks.* IEEE Trans. on Computers, 36, pp. 547-553, 1987.

[Dally 92] W. J. Dally. *Virtual-channel flow control*. IEEE Trans. on Parallel and Distributed Systems, 3, pp. 194-205, 1992.

[Dongarra 79] J. J. Dongarra et al. *LINPACK user's guide*. SIAM, 1979.

[Dongarra 87a] J. J. Dongarra et al. *A set of level 3 BLAS basic linear algebra subprograms.* Tech. Report ANL-MCS-TM-SS, Dept. of Computer Science, Argonne National Laboratory, 1988.

[Dongarra 87b] J. J. Dongarra, S. Hammarling, D. C. Sorensen. *Block reduction of matrices to condensad form for eigenvalue computations*. Tech. Report ANL-MCS-TM88, Dept. of Computer Science, Argonne National Laboratory, 1987.

[Dongarra 88] J. J. Dongarra et al. *An extended set of Fortran basic linear algebra subpro*grams. ACM Trans. on Mathematical Software, 14, pp. 1-17, 1988.

[Dongarra 91] J. J. Dongarra, R. van de Gein, R. Whaley. *Two dimensional basic linear algebra communication subprograms*. Computer Science Dept. Technical Report CS91-138, University of Tennessee, 1991 (Lapack Working Note #37).

[Dongarra 93] J. J. Dongarra, D. Walker. *The design of linear algebra libraries for high performance computers*. Computer Science Dept. Technical Report CS-93-188, University of Tennessee, 1993 (Lapack Working Note #58).

[Dongarra 95] J. J. Dongarra, R. C. Whaley. *A user's guide to the BLACS v1.0*, . Computer Science Dept. Technical Report CS-95-281, University of Tennessee, 1995 (Lapack Working Note #94).

[Dowd 93] K. Dowd. *High performance computing*. O'Reilly and Ass., Sebastopol, EEUU, 1993.

[Eberlein-87] P. J. Eberlein. *On the Schur decomposition of a matrix for parallel computation*. IEEE Trans. on Computers. C-36, pp. 167-174, 1987.

[Eisenstat 88] S. C. Eisenstat, M. T. Heath, C. S. Henkel, C.H. Romine. *Modified cyclic algorithms for solving triangular systems on distributed-memory multiprocessors*. SIAM J. on Sci. & Stat. Comp., Vol. 9, Nº 3, 1988.

[Fernando 85] K. V. Fernando, H. Nicholson. *On the Cross-Gramian symmetric MIMO systems*. IEEE Trans. on Circ. & Syst., Vol. 32, Nº 5, pp. 487-489, 1975

[Fernando 88] K. V. Fernando, S. J. Hammarling,. *A product induced singular value decomposition (PSVD) for two matrices and balanced realization*, in B.N. Datta, C.R. Johnson, M.A. Kaashoek, R.J. Plemmons, and E.D. Sontag, Eds., Linear Algebra in Signals, Systems, and Control (SIAM, PA, Philadelphia), pp. 128-140, 1988

[Flynn 66] M. J. Flynn. *Very high-speed computing systems*. Proceedings IEEE, 54, pp. 19011909, 1966.

[Fortuna 88] L. Fortuna, A. Gallo, C. Guglielmino, G. Nunneri. *On the solution of a non-linear matrix equation for MIMO symmetric realizations*. Syst. Contr. Lett., Vol. 11, pp. 79-82, 1988.

[Fortuna 92] L Fortuna, G. Nunnari, A. Gallo. *Model order reduction techniques with applications in electrical engineering*. (*Springer-Verlag*, 1992).

[Francis 61] J. G. F. Francis. *The QR transformation: a unitary analogue to the LR transformation, parts I and II*. Computing Journal, 4, pp. 265-272, 1961.

[Gajic 95] Z. Gajic, M. Qureshi. *Lyapunov matrix equation in system stability and control*. Academic Press, Math. in Sci. and Eng. Series, San Diego, CA, 1995.

[Gantmacher 66] F. R. Gantmacher. *Théorie des* matrices. (Tome 1, Ed. Dunod, París, Francia, 1966).

[Gardiner 86] J. D. Gardiner, A. J. Laub. *A generalization of the matrix sign function solution for the algebraic Riccati equations.* Int. Journal of Control, 44, pp. 823-832, 1986.

[Gardiner 88] J. D. Gardiner, A. J. Laub. *Matrix sign function implementations on a Hypercube multiprocessor.* Proceedings of the 27th Conference on Decision and Control, Austin, EEUU, 1988.

[Gardiner 96] J. D. Gardiner. *A stabilized matrix sign function algorithm for solving algebraic Riccati equations.* SIAM J. on Sci. & Stat. Comp., (aceptado y pdte. de publicación), 1996.

[Geist 88] G. A. Geist, R. C. Ward, G. J. Davis, R. E. Funderlic. *Finding eigenvalues and eigenvectors of unsymmetric matrices using a Hypercube multiprocessor*. Proceedings of the 3th Conference on Hypercube Concurrent Computers and Appl. (Ed. G. Fox), pp. 1577-1582, 1988.

[Geist 94] G. A. Geist et al. P Vill: *Parallel Virtual Machine. A user's guide and tutorial for networked parallel computing.* The MIT Press. Cambridge, EEUU, 1994.

[Glover 84] K. Glover. *All optimal Hankel-norm approximations of linear multivariable systems and their L^∞-error bounds*, International Journal of Control, Vol. 39, Nº 6, pp. 1115-1193, 1984.

[Golub 79] G. H. Golub, S. Nash, C. F. Van Loan. *A Hessenberg-Schur method for the problem AX + XB = C.* IEEE Trans. on Automatic Control, AC-24, pp. 909-913, 1979.

[Golub 89] G. H. Golub, C. F. Van Loan. *Matrix computations.* The Johns Hopkins University Press, Baltimore, EEUU, 1989.

[Gropp 94] W. Gropp, E. Lusk, A. Skjellum. *Using MPI. Portable parallel programming with the message-passing interface*. (The MIT Press, Cambridge, EEUU, 1994).

[Hammarling 82] S. J. Hammarling. *Numerical solution of the stable, non-negative definite Lyapunov equation*. IMA J. of Numerical Analysis, 2, pp. 303-323, 1982.

[Hammarling 82b] S. J. Hammarling. *Newton's method for solving the algebraic Riccati equation*. NPL Report DITC 12/82, National Physical Laboratory, Division of Information Technology and Computing, 1982.

[Hammarling 91] S. J. Hammarling. *Numerical solution of the discrete-time, convergente non-negative definite Lyapunov equation*. Systems & Control Letters, 17, pp. 137-139, 1991.

[Heath 86] M. T. Heath, A. J. Laub, C. C. Paige, R. C. Ward. *Computing the singular value decomposition of a product of two matrices*, SIAM J. on Sci. & Stat. Comp., Vol. 7, No. 4, 1986.

[Heath 88] M. T. Hea,th, C. H. Romine. *Parallel solution of triangular systems on distributed memory multiprocessors*. SIAM J. on Sci. & Stat. Comp., Vol. 9, pp. 558-588, 1988.

[Helmke 94] U. Helmke, J. Moore. Optimization and dynamical systems. Springer Verlag, London, 1994.

[Henrici 74] P. Henrici. *Applied and computational complex analysis.* Vol. 1, John Wiley sons, New York, EEUU, 1974.

[Henry 94] G. Henry, R. van de Geijn. *Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem: myths and reality*. Lapack Working note #79, 1994.

[Henry 96] G. Henry, R. van de Geijn. *Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem: myths and reality*. SIAM J. on Sci. & Stat. Comp., Vol 17, pp.870-833, 1996.

[Henry 97] G. Henry, D. Watkins, J. Dongarra. A *parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures*. Lapack Working note #121, 1997.

[Higham 84] N. J. Higham. *Computing the polar decomposition – with applications*. NAR No. 94, Dept. of Mathematics, Univ. of Manchester, 1984.

[Hoare 78] C.A.R. Hoare. *Communicating sequential processes.* CACM, 21, pp. 666-667, 1978.

[Hockney 90] R.W. Hockney, C.R. Jesshope. *Parallel computers*. (Adam Hilger Ltd., Bristol, EEUU, 1990).

[Hodel 92] A. S. Hodel, K. Polla. *Parallel solution of the large Lyapunov equations*. *SIAM J. Matrix Anal. & Appl.*, Vol. 18, No 4, pp. 1189-1203, Oct. 1992.

[Hwang 84] K. Hwang, F. A. Briggs. *Computer architecture and parallel processing.* (Mc Graw-Hill Book Company, New York, EEUU, 1984).

[Jonckheere 83] E. A. Jonckheere, L. M. Silverman. *A new set of invariants for linear systems - Applications to reduced-order compensator design*. IEEE Trans. on Automatic Control, Vol. 28, pp.953-964, 1983.

[Kabamba 85] P. T. Kabamba. *Balanced gains and their significance for L^2 model reduction*. IEEE Trans. on Automatic Control,Vol. 30, Nº 7, pp.690-693, 1985.

[Kågström 89] B. Kågström, P. Poromaa. *Distributed block algorithns for the triangular Sylvester equation with condition estimator*. Hypercube and Distributed Computers (Elsevier Science Publishers B.V., North-Holland) pp. 233-248, 1989.

[Kågström 90] B. Kågström, P. Poromaa. Distributed *& shared block algorithms for the triangular Sylvester equation with Sep^{-1} estimators.* Report UMINF 174-90, Inst. of Information Processing, Univ. of Umeá, 1990.

[Kailath 80] T. Kailath. *Linear systems.* Prentice-Hall Int., Englewood Cliffs, NJ, 1980.

[Kenney 89] C. Kenney, A. J. Laub. *On scaling Newtons method for polar decomposition and the matrix sign function.* Report No. SCL 89-11, Dept. of Electrical and Computer Engineering, University of California, 1989.

[Kenney 90a] C. Kenney, A. J. Laub, M. Wette. *Error bounds for Newton refinement of solutions to algebraic Riccati equations.* Math. Control and Signal systems, 3, pp. 211-224, 1990.

[Kenney 91b] C. Kenney, A. J. Laub. *Polar decomposition and matrix sign function condition estimates.* SIAM J. on Sci. & Stat. Comp., 12, pp. 488-504, 1991.

[Kenney 93] C. Kenney, A. J. Laub, P. M. Papadopoulos. A *Newton-squaring algorithm for computing the negative invariant subspace of a matrix*. IEEE Trans. on Automatic Control, AC-38, pp. 1284-1289, 1993.

[Kleinmann 68] D. L. Kleinmann. *On an iterative technique for Riccati equation computations*. IEEE Trans. on Automatic Control, AC-13, pp. 114-115, 1968.

[Kokotovic 76] P. V. Kokotovic, P. Sannuti. *Singular perturbations method for reducing the model order in optimal control design*. IEEE Trans. on Automatic Control, Vol. 31, Nº 4, pp. 123-132, 1976.

[Kokotovik 86] P. V. Kokotovik, R. E. O'Malley, P. Sannuti. *Singular perturbation and order reduction in control theory – an overview*. Automatica, 12, pp. 123-132, 1986.

[Kovarik 70] Z. Kovarik. *Some iterative methods for improving orthonormality*. SIAM J. on Numerical Analysis, 7, pp. 386-389, 1970.

[Kumar 94] V. Kumar. *Introduction to parallel computing*. (The Benjamin/Cumming Publishing COMDany, Inc., Redwood City, EEUU, 1994).

[Larin 93] V. B. Larin, F. A. Aliev. *Construction of square root factor for solution of the Lyapunov matrix equation*. Sys. Control Letters, Vol. 20, pp. 109-112, 1993.

[Laub 79] A. J. Laub. *A Schur method for solving algebraic Riccati equations*. IEEE Trans. on Automatic Control, AC-24, pp. 913-921, 1979.

[Laub 80] A. J. Laub. *Computation of balancing transformations*. Proceedings of the Joint Automate Control Conference, S. Francisco, Vol. II, 1980.

[Laub 87] A, J. Laub, M. T. Heath. Paige, R. C. Ward. *Computation of system balancing transformations and other applications of simultaneous diagonalization algorithms,* IEEE Trans. on Automatic Control, AC-32, pp. 115-122, 1987.

[Lawson 79] C. L. Lawson et al. *Basic linear algebra subprograms for Fortran usage.* ACM Trans. on Mathematical Software, 5, pp. 308-323, 1979.

[Li 88]G. Li, T. Coleman. A *Parallel triangular solver for a distributed-memory multiprocessor.* SIAM J. on Sci. & Stat. Comp., Vol. 9, pp. 485-502, 1989.

[Li 89]G. Li, T. Coleman. *A new method for solving triangular systems on distributed memory message-passing multiprocessor*. SIAM J. on Sci. & Stat. Comp., Vol.10, pp. 382-396, 1989.

[Leipnik 71] R. B. Leipnik. *Rapidly convergent recursive solution of quadratic operator equations*. Numerische Mathematik, 17, pp. 1-16, 1971.

[Marqués 91] M. Marqués, V. Hernández. *Parallel algorithms for the quasi-triangular Stein matrix equation on a shared memory multiprocessor*. Technical Report, DSIC11/36/1991, Universidad Politécnica de Valencia, 1991.

[Marqués 92] M. Marqués, V. Hernández. *Parallel algorithms for the quasi-triangular generalized Sylvester matrix equation on a shared-memory multiprocessor*. Procedings of the 2nd IFAC Workshop on Algorithms and Architectures for Real-Time Control (Eds. P. J. Fleming, W. H. Kwon), Pergamon Press, pp. 23-28, 1992.

[Marqués 93] M. Marqués, R. van de Geijn, V. Hernández. *Parallel algorithms for the triangular Sylvester equation*. Third SIAM Conference on Linear Algebra in Signals, Systems and Control, Seattle, 1993.

[Mehrmann 91] V. Mehrmann. *The autonomous linear quadratic control problem*. (SpringerVerlag Berlin, Alemania, 1991).

[Moler 87] C. Moler, J. N. Little, S. Bangert. *PC-Matlab users* guide. The Math Works Inc., Sherborn, EEUU, 1987.

[Mollar 96] M. Mollar, V. Hernández, *Computing the singular values of the product of two matrices in distributed memory multiprocesors*. Proc. of the 4th Euromicro Workshop on Par. and Dist. Computing, 1996.

[Moore 81] B. C. Moore. Principal component analysis in linear systems: Controlability, observability and model reduction. IEEE Trans. on Automatic Control, Vol. 26, Nº 1, pp. 100-105, 1981

[MPI 94] Message Passing Interface Forum. *Document for a standard Message-Passing Interface*. Technical Report Nº CS-93-214, University of Tennessee, 1994.

[Mullis 76] C. T. Mullis, R. A. Roberts. Synthesis of minimum roundoff noise fixed point digital filters. IEEE Trans. Circ. & Syst., Vol. 23, 1976.

[O'Leary 86] D. P. O'Leary, G. W. Stewart. *Data-flow algorithms for parallel matrix computations*. Communications of the ACM, Vol. 28, Nº 8, pp. 840-853, 1986.

[Paige 81] C. C. Paige. *Properties of numerical algorithms related to computing controllability*. IEEE Trans. on Automatic Control, AC-26, pp. 130-138, 1981.

[Pandey 90] P. Pandey, C. Kenney, A. J. Laub. *A parallel algorithm for the matrix sign function*. Int. J. of High Speed Computing, 2, pp. 181-191, 1990.

[Papadopoulos 9l] P.M. Papadopoulos et al. *Control-structure interaction for space station solar dynamic power module*. Proceedings of the 30th Conference on Decision and Control, Brighton, Reino Unido, 1991.

[Penzl 96] T. Penzl. *Numerical solution of generalized Lyapunov equations*. Technical Report SFB393/96-02, Fak. f. Mathematik, TU Chemnitz-Zwickau, 09107 Chemnitz, FRG, 1996.

[Penzl 98] T. Penzl. *A cyclic low rank Smith method for large sparse Lyapunov equations with applications in model reduction and optimal control*. Technical Report SFB393/98-6, Faculty of Mathematics, Technical University of Chemnitz-Zwickau, Germany, 1998.

[Pernebo 82] L. Pernebo, L. M. Silverman. *Model reduction via balanced state space representations*. IEEE Trans. on Automatic Control, Vol. 27, Nº 2, 1982.

[Perrot 92] R. H. Perrot. *Software for paralel computers*. Chapman & Hall, 1992.

[Petkov 87] P. H. Petkov, N. D. Christov, M. M. Konstantinov. *On the numerical properties of the Schur approach for solving the matrix Riccati* equation. Systems and Control Letters. 9, pp. 197-201, 1987.

[Petkov 91] P. Hr. Petkov, N. D. Christov, M. M. Konstantinov. *Computational methods for linear control systems,* Prentice-Hall International Ltd., Reino Unido, 1991.

[Ralha 93] R. M. S. Ralha. *Parallel one-side Householder transformations for eigenvalues computation*. Euromicro Workshop on Parallel and Distributed Processing, Gran Canaria, pp. 218-221, 1993.

[Ralha 95] R. M. S. Ralha. *Parallel QR algorithm for the complete eigensystem of symmetric matrices*. Euromicro Workshop on Parallel and Distributed Processing, San Remo, Italia, pp. 480-485, 1995

[Roberts 80] J. Roberts. *Linear model reduction and solution of the algebraic Riccati equation by the use of the sign function*. Int. Journal of Control, 32, pp. 677-687, 1980.

[Romine 86] C. H. Romine, J. M. Ortega. *Parallel solution of triangular systems of equations*. Technical Report RM-86-05, Dept. of Applied Mathematics, University of Virginia, 1986.

[Rosen 95] I. Rosen, C. Wang. *A multi-level technique for the aproximate solution of operator Lyapunov and algebraic Riccati equations*. SIAM J. Numer Anal., Vol. 39, pp. 514-541, 1995.

[Safonov 88] M. G. Safonov, R. Y. Chiang. Model order reduction for robust control: A Schur relative error method. Int. J. Adapt. Cont. Sign. Proc. Vol 2, pp. 259-272, 1988.

[Safonov 89] M. G. Safonov, R. Y. Chiang. *A Schur method for balanced-truncation model reduction*. IEEE Trans. on Automatic Control, Vol. 34, Nº 7, pp. 729-733, 1989.

[Silicon 94] Silicon Graphics Computer Systems. *PowerChallenge Technical Report.* Silicon Graphics Computer Systems Documentation, EEUU, 1994.

[Sima 94] V. Sima *Algorithms for linear-quadratic optimization*. (Marcel Dekker Inc., New York, 1994).

[Skelton 88] R. E. Skelton. *Dynamic systems control*, John Wiley, New York, 1988

[Stewart 73] G. W. Stewart. *Introduction to matrix computations*. Academic Press, New York, EEUU, 1973.

[Stewart 85] G. W. Stewart. *A Jacobi-like algorithm for computing the Schur decomposition of a non-hermitian matrix*. SIAM J. Sci. Statis. Comput., Vol. 6, pp. 853-864, 1985.

[Stewart 87] G. W. Stewart. *A parallel implementation of the QR algorithm*. Parallel Computing, 5, pp. 187-196, 1987.

[Stewart 90] G. W. Stewart, J. Sun. *Matrix perturbation theory*. Academic Press Inc., San Diego, EEUU, 1990.

[Sunderan 94] V. S. Sunderan, G. A. Geist, J. Dongarra, R. Manchek. *The PVM concurrent system: Evolution, experience, and trends*. Parallel Computing, Vol. 20, pp. 531-545, 1994.

[Tisseur 96] F. Tisseur. *Backward stability of the QR algorithm*. Technical Report 139, Equipe d'Analyse Numerique, Université Jean Monnet de Saint-Etienne, Cedex 02, France, 1996.

[Tombs 87] M. S. Tombs, I. Pstlehwaite. *Truncated balanced realization of a stable non-minimal state space system*. Int. Journal of Control, Vol. 46, pp. 1319-1330, 1987

[van de Geijn 89] R. A. van de Geijn, D. G. Hudson. *An eflicient parallel implementation of the non-symmetric QR algorithm*. Proceedings of the 4th Conference on Hypercube Concurrent Computers and Appl., 1989.

[Van Dooren 79] P. M. Van Dooren. *The computation of Kronecker's canonical form of a singular pencil*. Linear Algebra & Its Appl., 27, pp. 103-141, 1979.

[Van Dooren 91] P. M. Van Dooren. *Structured linear algebra problems in digital signal processing.* NATO ASI Series (Eds. G.H. Golub, P.M. Van Dooren), Vol. F70, SpringerVerlag, Berlin, Alemania, 1991.

[Van Dooren 95] P. M. Van Dooren. *Numerical linear algebra for signal systems and control*. Draft notes for the Graduate School in Systems and Control,. Louvain-la-Neuve, Bélgica, 1995.

[Varga 90] A. Varga. *A note on Hammarling's algorithm for the discrete Lyapunov equation*. Systems & Control Latters, North-Holland, Nº 15, pp. 273-275, 1990.

[Varga 91] A. Varga. *Efficient minimal realization procedure based on balancing*. IMACS Symposium MCTS, Casablanca, pp. 42-47, 1991.

[Varga 94] A. Varga. *Numerical methods and software tools for model reduction*, Proc. 1st MATHMOD Conf., Vienna, pp. 226-230, 1994.

[Varga 95] A. Varga. *On stabilization methods of descriptor systems*. Sys. Control Letters, Vol. 24, pp. 133-138, 1995.

[Watkins 94] D. S. Watkins. *Shifting strategies for the parallel QR algorithm.* SIAM J. on Scientific Computing, 15, pp. 953-958, 1994.

[Wilkinson 65] J. H. Wilkinson. *The algebraic eigenvalue problem*. Clarendon Press: Oxford, 1965.

[Wilson 79] D. A. Wilson, R. N. Mishra. *Optimal reduction of multivariable systems*. International Journal of Control, Vol. 29, Nº 2, pp. 267-278, 1979.

[Wolfram 88] S. Wolfram. *Mathematica: a system for doing mathematics by computer*. Addison Wessley, New York, 1988.

[Zhou 96] K. Zhou, J. Doyle, K Glover. Robust and optimal control. Prentice Hall, Upper Saddle River, NJ, 1996.