Algoritmos paralelos segmentados para los problemas de Mínimos Cuadrados Recursivos (*RLS*) y de Detección por Cancelación Ordenada y Sucesiva de Interferencia (*OSIC*)

Tesis Doctoral
para optar al grado de
Doctor en Informática

Septiembre 2007

Autor: Francisco José Martínez Zaldívar
Director: Dr. A.M. Vidal Maciá

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Quiero dedicar este trabajo y todo el esfuerzo que ha conllevado a las dos personas que llenan mi vida, por quienes vale la pena seguir luchando cada día: mi hijo Carlos y mi mujer Julia.

Espero, Carlos, que me sigas enseñando cosas cada día. Tengo mucho que aprender de ti.

Espero, Julia, que me perdones. Jamás podré recompensarte el esfuerzo que todo esto ha supuesto para ti. Siempre te agradeceré tu paciencia, tu cariño, tu perseverancia y tu "¿acabas ya?" Ahora las cosas deben cambiar.

Espero tener tiempo para toda la familia que, de una u otra forma, ha vivido mis ausencias pacientemente. Gracias a todos.

Espero, Antonio, que te sientas mínimamente satisfecho con el trabajo que hemos realizado. Gracias por tu ayuda durante todos estos años.

Espero que aquéllos cuyos nombres no están en estas líneas, os encontréis en ellas con mi más sincero agradecimiento

Espero dejar de tener miedo a aprender, miedo a recorrer "el sendero de los perros hecho por los mismos perros"

Espero dejar de esperar.

Paco

Resumen

Dentro del marco de los sistemas de comunicaciones de banda ancha, podemos encontrar canales modelados como sistemas MIMO (Multiple Input Multiple Output) en el que se utilizan varias antenas en el transmisor (entradas) y varias antenas en el receptor (salidas), o bien sistemas de un solo canal que puede ser modelado como los anteriores (sistemas multi-portadora o multicanal con interferencia entre ellas, sistemas multi-usuario con una o varias antenas por terminal móvil y sistemas de comunicaciones ópticas sobre fibra multimodo). Estos sistemas pretenden alcanzar valores de capacidad de transmisión relativa al ancho de banda muy superiores al de un único canal SISO (Single Input Single Output).

Hoy en día, existe, desde un punto de vista de implementación del sistema, una gran actividad investigadora dedicada al desarrollo de algoritmos de codificación, ecualización y *detección*, muchos de ellos de gran complejidad, que ayuden a aproximarse a las capacidades prometidas.

En el aspecto relativo a la *detección*, las soluciones actuales se pueden clasificar en tres tipos: soluciones subóptimas, ML (*Maximum Likelihood*) o cuasi-ML e iterativas. En estas últimas, se hace uso explícito de técnicas de control de errores empleando intercambio de información *soft o indecisa* entre el detector y el decodificador; en las soluciones ML o cuasi-ML se lleva a cabo una búsqueda en árbol que puede ser optimizada llegando a alcanzar complejidades polinómicas en cierto margen de relación señal-ruido; por último dentro de las soluciones subóptimas destacan las técnicas de forzado de ceros, error cuadrático medio y cancelación sucesiva de interferencias SIC (*Succesive Interference Cancellation*), ésta última con una versión ordenada —OSIC—. Las soluciones subóptimas, aunque no

llegan al rendimiento de las ML o cuasi-ML son capaces de proporcionar la solución en tiempo polinómico de manera determinista.

En la presente tesis doctoral, hemos implementado un método basado en la literatura para la solución del problema OSIC; adicionalmente hemos desarrollado e implementado un método novedoso para este mismo problema con mejores prestaciones, lo cual consideramos una de las aportaciones más interesantes de este trabajo. Las implementaciones han sido paralelizadas, evaluadas y comparadas.

Ambos métodos OSIC están basados en sendos métodos que resuelven el problema de mínimos cuadrados recursivos (RLS —*Recursive Least Squares*—). Estos métodos están basados en el filtro de Kalman y previamente a las implementaciones OSIC, éstos han sido estudiados, paralelizados, evaluados y comparados, junto con un método adicional que resuelve también el problema RLS y está basado en la actualización de la factorización QR de una matriz. Como denominador común a todas las paralelizaciones, y dadas las características de los algoritmos, cabe comentar que todos los algoritmos paralelos han sido diseñados con carácter segmentado, debido a su intrínseca secuencialidad, a la buena eficiencia obtenida con esta perspectiva y a su potencial extrapolación a implementaciones de tipo VLSI.

Resum

Dins del marc dels sistemes de comunicacions de banda ampla, podem trobar canals modelats com sistemes MIMO (Multiple Input Multiple Output) en el qual s'utilitzen diverses antenes en el transmissor (entrades) i diverses antenes en el receptor (eixides), o bé sistemes d'un sol canal que poden ser modelats com els anteriors (sistemes multi-portadora o multicanal amb interferència entre elles, sistemes multi-usuari amb una o diverses antenes per terminal mòbil i sistemes de comunicacions òptiques sobre fibra multimode). Aquests sistemes pretenen arribar a valors de capacitat de transmissió relativa a l'ample de banda molt superiors al d'un únic canal SISO (Single Input Single Output).

Avui dia, existeix, des d'un punt de vista d'implementació del sistema, una gran activitat investigadora dedicada al desenvolupament d'algorismes de codificació, equalització i *detecció*, molts d'ells de gran complexitat, que ajuden a aproximar-se a les capacitats promeses.

En l'aspecte relatiu a la *detecció*, les solucions actuals es poden classificar en tres tipus: solucions subòptimes, ML (*Maximum Likelihood*) o quasi-ML i iteratives. En aquestes últimes, es fa ús explícit de tècniques de control d'errors emprant intercanvi d'informació *soft o indecisa* entre el detector i el decodificador; en les solucions ML o quasi-ML es porta a terme una recerca en arbre que pot ser optimitzada arribant a complexitats polinòmiques en cert marge de relació senyal-soroll; finalment, dins de les solucions subòptimes destaquen les tècniques de forçat de zeros, error quadràtic mig i cancel·lació successiva d'interferències SIC (*Succesive Interference Cancellation*), aquesta última amb una versió ordenada —OSIC—. Les solucions subòptimes, encara que no arriben al rendiment de les ML o quasi-ML, són capaços de proporcionar la solució en temps polinòmic de manera determinista.

En la present tesi doctoral, hem implementat un mètode basat en la literatura per a la solució del problema OSIC; addicionalment hem desenvolupat i implementat un mètode nou per a aquest mateix problema amb millors prestacions, la qual cosa considerem una de les aportacions més interessants d'aquest treball. Les implementacions han estat paral·lelitzades, avaluades i comparades.

Ambdós mètodes OSIC estan basats en sengles mètodes que resolen el problema de mínims quadrats recursius (RLS —*Recursive Least Squares*—). Aquests mètodes estan basats en el filtre de Kalman i prèviament a les implementacions OSIC, aquests han estat estudiats, paral·lelitzats, avaluats i comparats, juntament amb un mètode addicional que resol també el problema RLS i està basat en l'actualització de la factorització QR d'una matriu.

Com a denominador comú a totes les paral·lelitzacions, i donades les característiques dels algorismes, cal comentar que tots els algorismes paral·lels han estat dissenyats amb caràcter segmentat, a causa de la seua intrínseca seqüencialitat, a la bona eficiència obtinguda amb aquesta aproximació i a la seua potencial extrapolació a implementacions de tipus VLSI.

Abstract

In the context of wideband communication systems, we can find communications channels modeled as either MIMO (Multiple Input Multiple Output) systems where several antennas are used in the transmitter (inputs) and several ones in the receiver (outputs), or one channel systems that can be modeled as the previous systems (multi-carrier systems or inter-carrier interference multichannel systems, multi-user systems with one or several mobile terminal antennas and optical communications over multimode fiber). These systems expect to reach bandwidth relative transmission capacity greater than SISO (Single Input Single Output) channel model.

Nowadays, there exists an important researching activity, from the system implementation point of view, devoted to the development of a high complexity coding, equalization and *detection* algorithms that help to get the promising capacities.

The nowadays solutions to the *detection* problem can be classified in three categories: suboptimal, ML (*Maximum Likelihood*) or cuasi-ML, and iterative solutions. The last one uses explicit error control techniques with *soft* information exchange between the detector and the decoder. In the ML or cuasi-ML solutions, a tree searching technique is used that can be optimized in order to get a polynomial complexity cost within certain signal-to-noise ratio interval. We can find in the suboptimal solutions, the zero forcing technique, minimum mean squared error and successive interference cancellation (SIC) techniques and its ordered version (OSIC). The suboptimal solutions do not reach the ML or cuasi-ML performance but they can provide the solution in a polynomial time.

We have implemented in this thesis a method based on the current literature to solve the OSIC problem. Besides, we have developed and implemented an original method to solve

this problem with better performance. This is one of the most interesting contributions of our work. The implementations have been parallelized, evaluated and compared.

Both OSIC methods are based on methods that solve the recursive least squares problem (RLS) and are inspired in the Kalman filter. These methods have been studied, parallelized, evaluated and compared. Besides, an additional method based on the QR factorization updating to solve the RLS problem has been developed.

The design of all the parallel algorithms has been oriented to a pipelined organization due to the intrinsic sequential character of the algorithms, the good efficiency got with this technique and the potential extrapolation to VLSI implementations.

Índice general

1 Introducción y objetivos

En este primer capítulo mostramos la motivación del trabajo desarrollado, dando una visión general de la tesis doctoral, planteando los objetivos que pretendemos cubrir, así como la estructuración del resto de capítulos.

1.1 Motivación

Esta tesis doctoral pretende conjugar el diseño de algoritmos paralelos eficientes, que resuelven ciertos problemas matemáticos concretos, con aplicaciones reales, las cuales se describen a partir de la solución de estos problemas. Es, asimismo, nuestra intención el mostrar claramente las aportaciones a las que este estudio da lugar.

Hemos enfocado nuestro interés en un caso particular del problema matemático de mínimos cuadrados (LS —*Least Squares*—) denominado mínimos cuadrados recursivos (RLS —*Recursive Least Squares*—). Podemos encontrar algoritmos paralelos para resolver el problema RLS utilizando principalmente arquitecturas sistólicas por ejemplo en [1, 2, 3, 4, 5, 6]. Por otra parte, también podemos encontrar algoritmos paralelos para la factorización QR en [7, 8, 9, 10], [11], [12], los cuales pueden ser utilizados para resolver el problema de mínimos cuadrados, para arquitecturas de tipo memoria compartida o de

memoria distribuida. En [13] se muestra la intensa relación existente entre la estimación del estado de un sistema descrito a partir de las ecuaciones de estado mediante el filtro de Kalman y el problema RLS; en [14, 15] aparecen paralelizaciones de algunas modificaciones del filtro de Kalman con aplicaciones en el contexto de Teoría de Control.

Este tipo de problemas ha sido ampliamente utilizado para expresar o solucionar numerosos problemas del mundo real. Algunos ejemplos pueden encontrarse en campos como Control o Procesado de Señal: identificación de sistemas [16] o paramétrica [17], control activo de sonido [18], reproducción de sonido multicanal [19], igualación o *ecualización* de canal [1], detección, ... La solución a problemas de igualación y detección de señales para sistemas con *Múltiples Entradas y Múltiples Salidas* (MIMO —*Multiple Input Multiple Output*) ha resultado ser un reto tecnológico en los últimos años, principalmente debido a la prometedora capacidad de transmisión de información de dichos sistemas [20]. Entre otros esquemas para detección de señales MIMO, el procedimiento de detección *Bell Labs. Layered Space Time* (BLAST) ha mostrado una significativa ganancia en la capacidad de transmisión de información en la práctica [21].

En numerosos sistemas de comunicaciones aparecen canales de transmisión con múltiples entradas y salidas. Este tipo de canales reciben en la literatura científica anglosajona el nombre de sistemas MIMO, y suelen asociarse a la presencia de múltiples antenas en transmisión y en recepción. Existen otros entornos de comunicaciones en los que también se utiliza un único canal que puede modelarse como un canal MIMO. Entre los tipos de sistemas MIMO que podemos encontrarnos tanto en la literatura científica como en aplicaciones reales, y que resultan de nuestro interés son:

- MIMO-SU: sistemas de transmisión punto a punto con múltiples antenas en ambos lados del enlace, en donde sólo se considera un usuario (*single-user*).

- MIMO-MU: sistemas de transmisión multi-usuario (*multi-user*), con una estación base con múltiples antenas y varios usuarios con una o más antenas, distinguiendo los siguientes casos:

  - Conexión ascendente (*uplink*) entre los múltiples usuarios y la estación base, en la literatura recibe el nombre de MIMO con acceso múltiple MIMO-MAC.
  - Conexión descendente (*downlink*) entre la estación base y los múltiples usuarios, en la literatura recibe el nombre de MIMO con transmisión broadcast MIMO-BC.

- OFDM-Var: sistemas que utilizan la modulación OFDM en entornos móviles, dando lugar a canales variantes con el tiempo que provocan la aparición de interferencia entre portadoras (ICI, *Inter-Carrier Interference*). Cabe destacar que estos sistemas utilizan una única antena en cada extremo del enlace, pero pueden ser descritos analíticamente mediante una formulación de canal MIMO debido a la ICI. Por otro lado, los sistemas de comunicación MIMO-SU y MIMO-MU incluyen ya la transmisión OFDM entre los posibles casos de estudio.

- OPT-MIMO: sistemas de comunicaciones ópticas a través de fibras multimodo.

En general, el problema se puede plantear según el esquema de la figura 1.1. La expresión analítica de la salida del canal MIMO en formato vectorial viene dada por:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v},$$

donde el vector $\mathbf{x}$ se obtiene a partir de ciertos símbolos $\mathbf{s}$ de diversas formas dependiendo del contexto; habitualmente se tiene una de estas opciones: directamente los símbolos transmitidos a través de las M entradas, una pre-codificación lineal ($\mathbf{x} = \mathbf{W}\mathbf{s}$) de los símbolos, o una pre-codificación no lineal de los símbolos. El vector $\mathbf{y}$ representa los datos recibidos desde las m salidas del canal. $\mathbf{H}$ es una matriz que representa el efecto del canal sobre el que se ha transmitido. El vector $\mathbf{v}$ contiene el ruido interferente captado en las m salidas receptoras del sistema.

De forma general, cada salida del canal se forma a partir de una copia de los símbolos transmitidos por todas las entradas, ya sea en el mismo instante (por ejemplo, en canales

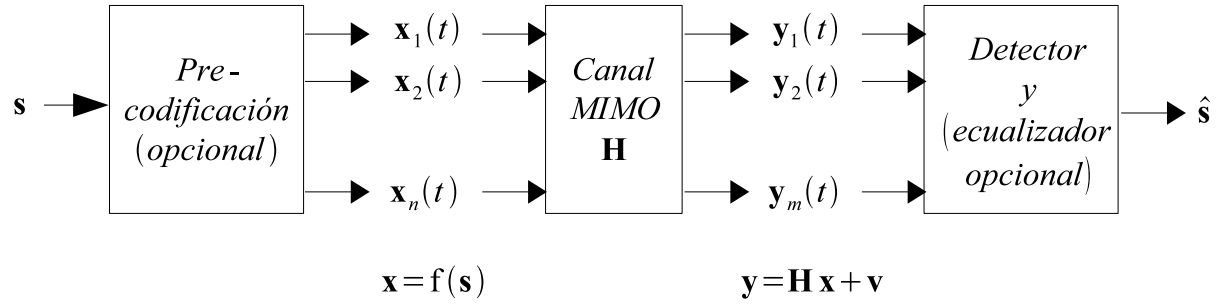

Figura 1.1: Modelo genérico de un sistema de comunicaciones MIMO

con desvanecimiento plano) o como combinación de sucesivos períodos de símbolo (por ejemplo, en canales selectivos en frecuencia). Esta mezcla debe deshacerse en recepción para poder obtener una buena estimación de los símbolos transmitidos $\mathbf{s}$. El método óptimo de máxima verosimilitud (ML, *Maximum Likelihood*) permite estimar el vector $\mathbf{s}$ mediante una búsqueda exhaustiva entre todos los posibles candidatos que minimice la función:

$$\hat{\mathbf{s}} = \arg\min ||\mathbf{HWs} - \mathbf{y}||^2.$$

La solución ML de búsqueda exhaustiva al problema de la detección MIMO tiene un coste computacional demasiado grande, por esta razón, existe un gran interés en la búsqueda de soluciones de baja complejidad. De manera general se pueden clasificar estas soluciones en tres tipos:

- Soluciones subóptimas: en este caso encontramos receptores lineales como el forzador de ceros (ZF) o el de error cuadrático medio mínimo (MMSE), cuya característica principal consiste en la inversión del canal MIMO. Aunque son sencillos, sus prestaciones son pobres. Existen otras soluciones basadas en la cancelación sucesiva de interferencia (SIC, *Succesive Interference Cancellation*) como el algoritmo V-BLAST y sus variantes ordenadas (OSIC, *Ordered SIC*), [22] con notables mejores prestaciones.

- Soluciones ML o cuasi-ML: aquí se encuadrarían los algoritmos basados en búsqueda en árbol como Sphere Decoding (SD) [23, 24, 25] que ha recibido gran atención al po-

der resolver el problema de la detección ML en sistemas MIMO, con una complejidad polinómica en cierto margen de relación señal-ruido (*signal-to-noise ratio* —SNR—) [26, 27]; en el peor caso (por ejemplo, canales mal condicionados y SNR baja), SD puede tener una complejidad cercana a ML. Por otro lado, también aparecen en la literatura otras soluciones prometedoras basadas en programación semidefinida [28, 29, 30, 31], que aunque no llegan a las prestaciones de SD, siempre tienen una complejidad polinómica.

- Soluciones iterativas: estas soluciones aprovechan que la mayor parte de sistemas de comunicaciones hacen uso de códigos para la corrección de errores. Así, en estas soluciones existe un intercambio de información *soft* o *indecisa* entre el detector y el decodificador, que permite refinar los resultados de cada uno de ellos en varias iteraciones, de manera que al terminar se obtienen mejores prestaciones que si cada una de estas etapas trabaja por separado. Entre los detectores hay soluciones lineales, DFE (*Decision Feedback Equalizer*) y por búsqueda en árbol [32].

En este trabajo desarrollaremos algoritmos del primer tipo, y más concretamente de la variante OSIC, debido a su complejidad polinómica.

En el modelo de referencia del sistema MIMO se suele suponer que el canal de transmisión tiene desvanecimiento plano. Este aspecto no tiene por qué ser así, de hecho, en sistemas de banda ancha resulta habitual encontrar canales selectivos en frecuencia. Este problema puede resolverse, como se ha comentado antes, utilizando la modulación OFDM. En el caso de trabajar con modulaciones de portadora única, la solución óptima en recepción consiste en utilizar estimación de secuencia ML multicanal (MLSE) que permita compensar las dos interferencias: la IES debida al canal multicamino y la interferencia co-antena debida al uso de antenas múltiples [33]. La solución ML se complica exponencialmente debido a la ISI (*InterSymbol Interference*), por lo que habitualmente se aplican métodos subóptimos (ecualización ZF, MMSE o DFE) y se resuelven ambas interferencias al mismo tiempo [34]. Otra posibilidad es independizar los problemas y aplicar métodos

subóptimos de ecualización temporal (acortamiento del canal) [35] o espacio-temporal [36] muy similares a los utilizados para ecualización multicanal de sonido, [37].

Las ganancias en capacidad de transmisión que se pueden alcanzar utilizando los algoritmos de detección descritos anteriormente, se obtienen a cambio de un aumento de la complejidad computacional en el receptor. La mayor parte de las publicaciones sobre reducción de complejidad se han planteado teniendo en mente arquitecturas convencionales. El inconveniente de este tipo de tecnología es que, hoy en día y en el futuro inmediato, no es capaz de alcanzar los requerimientos de prestaciones que son necesarios en sistemas de comunicaciones de banda ancha. Por tanto la única solución es realizar la implementación VLSI de los algoritmos; el problema es que muchas de las optimizaciones pensadas para sistemas software o no son inmediatamente aplicables o no suponen ninguna ventaja al aplicarlos a la implementación VLSI. En la implementación VLSI de algoritmos de procesado de señal, el mayor potencial para la reducción de la complejidad es la optimización conjunta de: el algoritmo (reducción de operaciones y dependencias entre ellas, robustez, estabilidad numérica, etc.), su arquitectura, teniendo en cuenta las contraprestaciones a nivel de circuito (grado de paralelización y segmentación) y su aritmética (la cuantificación en coma fija), dejando la síntesis final del circuito en manos de las herramientas automáticas. De esta manera es posible acortar el tiempo de diseño y lograr circuitos prácticos de altas prestaciones. Así pues, un tema de gran relevancia es la optimización de algoritmos y técnicas para la reducción de complejidad de la implementación VLSI. En general, lograr algoritmos y arquitecturas con un alto grado de paralelización y regularidad es clave para lograr implementaciones VLSI de altas prestaciones y bajo consumo [38].

Teniendo presente estas consideraciones, resulta de interés la perspectiva de intentar enfocar la paralelización de las soluciones a los problemas, con un carácter segmentado. En problemas cuya solución conocida es inherentemente secuencial, y con ciertas características en cuanto a las dependencias de los datos, un enfoque segmentado, en ocasiones, produce muy buenos resultados en las paralelizaciones. Nosotros emplearemos computadores paralelos de propósito general para validar las implementaciones, como paso previo a

la continuidad de las ideas en otros sistemas como los basados en FPGA (Field Programmable Gate Array). Con estos sistemas se pueden alcanzar las prestaciones requeridas para la solución del problema y permite un prototipo funcional fácilmente verificable de la solución final. En la actualidad existen sistemas que agrupan las prestaciones de un computador paralelo de propósito general con módulos que integran subsistemas basados en FPGA.

El abordar este tipo de problemas requiere el esfuerzo de integrar conceptos de varias disciplinas científicas como son: Teoría de la Señal, Control, Computación, Arquitectura de Computadores, Álgebra Lineal Numérica, etc. Algunas de las soluciones a los problemas planteados en los sistemas MIMO descritos en el campo de Teoría de la Señal, están apoyadas en la solución a problemas del tipo mínimos cuadrados recursivos (RLS) descritos y resueltos en el campo del Álgebra numérica; a su vez, existen soluciones alternativas al problema RLS basadas en soluciones a problemas clásicos de Teoría de Control como es el filtro de Kalman, empleado para la estimación del estado de un sistema. Las implementaciones paralelas eficientes requieren optimizar la interacción entre los algoritmos paralelos diseñados, intentando conseguir el máximo paralelismo posible, dentro del contexto de la Computación y la Arquitectura del o los procesadores donde se vayan a ejecutar. Un producto final y real requiere la interacción con el mundo exterior: interfaces de entrada y de salida y un tiempo de respuesta que se adecúe a las necesidades del proceso.

El contexto en el que se va a desarrollar la presente tesis doctoral intersecta algunas de las disciplinas comentadas para poder plantear soluciones a los problemas citados. Es necesario conocer mínimamente los problemas físicos que se pretenden resolver con las soluciones actuales. A continuación debemos plasmar estas soluciones en sistemas de cómputo que permitan proporcionarlas en el tiempo de respuesta requerido, evaluando sus características y previendo sus posibles mejoras.

1.2 Objetivos

La presente tesis doctoral versa sobre el desarrollo e implementación de algoritmos paralelos novedosos que resuelven el problema RLS y que, en algunos casos, aportan mejoras sobre los ya existentes o complementan la funcionalidad de otros. Estas mejoras son extrapolables a prácticamente cualquier tipo de arquitectura paralela convencional: memoria distribuida, memoria compartida, memoria compartida y distribuida, multiprocesadores DSP (Digital Signal Processor), implementaciones VLSI, en redes homogéneas o heterogéneas de procesadores, etc. Se han utilizado sólo bibliotecas estándar, que aseguren su fácil portabilidad, como `BLAS-LAPACK` para el cálculo matricial, `MPI` para la programación en memoria distribuida y directivas y opciones de compilación `OpenMP` para memoria compartida.

Adicionalmente, describiremos una modificación de una solución actual al problema OSIC-MIMO, con la cual se consigue un sustancial ahorro en términos de complejidad computacional.

Los objetivos que nos planteamos en la elaboración del presente trabajo son los siguientes:

- Estudio de las arquitecturas paralelas actuales y las disponibles para validar el trabajo a realizar, así como del software necesario para su máximo aprovechamiento y su proyección en el campo de las aplicaciones basadas en la filosofía de los sistemas MIMO.

- Estudio del problema matemático de los mínimos cuadrados en sus distintas vertientes y concretamente la alternativa RLS, junto con aplicaciones reales en donde se requieren soluciones eficientes para estos problemas, como es el aspecto de la detección asociada a problemas de ecualización, codificación y precodificación en sistemas MIMO.

- Estudio de distintos algoritmos que resuelven los problemas planteados, así como de

los algoritmos base en los que se fundamenta, tanto aquéllos inspirados en variaciones del filtro de Kalman como los basados en la actualización de la factorización QR.

- Implementación de los algoritmos secuenciales y paralelos, así como la evaluación de los mismos para su comparación y extracción de conclusiones a partir de los resultados obtenidos.

- Replanteamiento y propuestas de mejora de las implementaciones a partir de las conclusiones.

- Aplicación de algunos de los métodos implementados, con las modificaciones oportunas, para abordar el problema OSIC de manera eficiente.

- Conclusiones definitivas del estudio realizado y reflexión sobre las líneas abiertas y futuras de trabajo.

1.3 Estructura de la memoria de la tesis doctoral

Esta memoria está estructurada de la siguiente forma:

- El capítulo 2 presenta el *estado del arte* de las arquitecturas paralelas, de los paradigmas de programación empleados y de las técnicas sobre evaluación de prestaciones de algoritmos paralelos, con los que pretendemos llevar a cabo implementaciones eficientes, que podrán resolver en la práctica cierta clase de problemas y aplicaciones. Asimismo, se describen someramente las plataformas sobre las que se han realizado las pruebas y la metodología en la experimentación y obtención de resultados.

- El capítulo 3 muestra una introducción al problema de los mínimos cuadrados en las vertientes que resultan de nuestro interés, especialmente la visión recursiva del mismo (problema RLS). A continuación expondremos algunos aspectos fundamentales del Filtro de Kalman debido a que cierta interpretación del mismo es extremadamente

útil para plantear ciertas versiones que resuelven de manera eficiente el problema RLS.

- En el capítulo 4 expondremos una primera solución al problema RLS, basándonos en una de las variantes del filtro de Kalman denominada *versión raíz cuadrada del filtro de Kalman o filtro de covarianza*, mostrando la implementación y conclusiones tanto de la versión secuencial como de la paralela.

- En el capítulo número 5 repetiremos los pasos del capítulo anterior, pero esta vez para una segunda variante del filtro de Kalman denominada *versión raíz cuadrada del filtro de información.*

- En el sexto capítulo abordaremos una tercera solución al problema RLS basándonos en la actualización (*updating*) de la factorización QR de la matriz problema, proporcionando la solución secuencial y paralela con los detalles oportunos que nos han permitido obtener una implementación eficiente.

- Procederemos a realizar una comparación de las prestaciones de los tres algoritmos anteriormente citados en el capítulo séptimo, extrayendo las conclusiones oportunas.

- En el capítulo 8 mostraremos una aplicación en el campo de Teoría de la Señal y Comunicaciones empleando los algoritmos expuestos en dos de los capítulos anteriores, haciendo hincapié en cierta modificación del método basado en el filtro de información para emplearlo en esta aplicación de Comunicaciones.

- El noveno y último capítulo lo dedicaremos a las conclusiones que podemos obtener del trabajo realizado, comentando las líneas abiertas o de futuro que plantea la presente tesis doctoral.

2

Arquitecturas paralelas y programación en paralelo

En el presente capítulo describiremos y clasificaremos las arquitecturas paralelas actuales, comentaremos los paradigmas típicos de programación en paralelo y describiremos las plataformas así como el software empleado para realizar la validación y evaluación de los algoritmos. Describiremos las técnicas usuales de evaluación de prestaciones de algoritmos paralelos, así como la metodología empleada para la elaboración de los algoritmos propuestos en este trabajo.

2.1 Arquitecturas paralelas

2.1.1. El paralelismo en las arquitecturas

El incremento del paralelismo y la localidad del acceso a los datos han dado lugar al aumento de las prestaciones en los computadores a lo largo de su historia [39], [40].

Las memorias *cache*, típicamente con distintos niveles, permiten amortiguar la diferencia de velocidad entre el procesador y la memoria, siempre y cuando se pueda dar el principio de localidad de acceso a los datos espacial y temporalmente.

El paralelismo podemos considerarlo desde dos perspectivas: la replicación de elementos a cualquier nivel (procesadores, memoria, entrada/salida,...) y la segmentación a nivel de procesador que permiten paralelismo a nivel de instrucciones (ILP —*Instruction Level Parallelism*—), como por ejemplo en procesadores superescalares y VLIW (*Very Long Instruction Word*) y a nivel de unidades funcionales, como en los anteriores y en los procesadores vectoriales.

Existen numerosas clasificaciones de arquitecturas de computadores [41, 42, 43], aunque la clásica ha sido la primera de ellas. En ésta se clasifican los computadores atendiendo a la multiplicidad del flujo de instrucciones y de datos: SISD (*Single Instruction Single Data*, computadores monoprocesador), SIMD (*Single Instruction Multiple Data*, procesadores vectoriales y matriciales), MISD (*Multiple Instruction Single Data*) y MIMD (*Multiple Instruction Multiple Data*, multiprocesadores y multicomputadores).

La clasificación de Flynn, [41] pone de manifiesto dos tipos de paralelismo: de datos y funcional. El primero se da cuando una función o instrucción se ejecuta repetidas veces con datos distintos, y se da en los sistemas SIMD y puede darse en los MIMD si la aplicación se implementa como un programa paralelo SPMD (Single Program Multiple Data). El paralelismo funcional ocurre cuando las funciones o instrucciones se ejecutan en paralelo con distintos datos (arquitecturas MIMD). Este paralelismo funcional puede darse a nivel de instrucciones (ILP), a nivel de bucle, a nivel de funciones o a nivel de programas.

Existen otras clasificaciones que atienden a cómo los procesadores y las memorias están interconectados, encontrando sistemas como MPP (*Massively Parallel Processors*), SMP (*Symmetric MultiProcessors*) o UMA (*Uniform Memory Access*), NUMA (*Non Uniform Memory Access*), sistemas distribuidos y *clusters*.

En la arquitectura MPP realmente no se comparte nada. Consisten en cientos de nodos con una red de interconexión rápida. Cada nodo posee una memoria principal y uno o varios procesadores, ejecutándose una copia separada del sistema operativo.

En los sistemas SMP o UMA, por el contrario, se comparte prácticamente todo, teniendo todos los recursos disponibles para todos los procesadores, ejecutándose una única

copia del sistema operativo.

En la arquitectura NUMA, cada procesador tiene una visión global de la memoria, pero con un acceso no uniforme a la misma, dependiendo de dónde está ubicada la información. En la subclase ccNUMA, la característica fundamental es el mecanismo por el que se mantiene una versión coherente de la memoria *cache* en todos los procesadores.

Los sistemas distribuidos se consideran redes convencionales de computadores independientes, ejecutando cada uno de ellos su propio sistema operativo. Cada máquina individual puede ser una combinación de MPP, SMP, clusters o computadores individuales.

Por último, los *clusters* son una colección de estaciones de trabajo o PC interconectados mediante una red de alta velocidad. Trabajan como una colección integrada de recursos.

Arquitecturas con memoria compartida y con memoria distribuida

Una nueva clasificación de máquinas paralelas atiende a la posibilidad de que todos los procesadores tengan acceso a toda la memoria del sistema o no. En caso afirmativo, al sistema se le suele denominar multiprocesador con memoria compartida; en caso contrario con memoria distribuida o multicomputador.

En un multiprocesador con memoria compartida, los elementos de proceso se comunican entre ellos a través de la memoria global, es por lo que poseen un espacio único de direcciones. Cada uno de estos elementos puede ser un procesador secuencial, vectorial, supersegmentado, ... Evidentemente, en este tipo de sistemas el cuello de botella se encuentra precisamente en el acceso intensivo a la memoria por parte de los procesadores.

En un multiprocesador con memoria distribuida o multicomputador, cada nodo de proceso posee su propia memoria no accesible por el resto. Independientemente de este hecho, cada nodo puede ser un procesador escalar, un nuevo multiprocesador, un procesador superescalar, ... La comunicación entre los distintos nodos se lleva a cabo mediante el mecanismo denominado *paso de mensajes*, mediante el cual, y a través de una red de interconexión, se transmite información de un nodo origen hacia uno destino. De manera explícita, aparecen dos conceptos que se entretejen en la evolución de los algoritmos que se

ejecutan en este tipo de máquinas que son el cálculo y las comunicaciones. Estas últimas suelen suponer el cuello de estos sistemas.

Por último, también podemos considerar arquitecturas cuya memoria se califica de distribuida-compartida, es decir, sistemas en los que la memoria se halla físicamente distribuida, pero lógicamente compartida. En estos sistemas multiprocesador es obvio que el tiempo de acceso a la memoria compartida dependerá de dónde de halle la dirección a la que un procesador desea acceder; de ahí surgen los esquemas NUMA (*Non Uniform Memory Access*) con sus variantes cc-NUMA (*cache coherent*), ncc-NUMA (no cc-NUMA) y COMA (*Cache Only Memory Access*).

2.1.2. Redes de interconexión

Los elementos de proceso en una arquitectura paralela necesitarán comunicarse entre ellos utilizando las denominadas redes de interconexión.

Una primera clasificación distingue entre redes regulares (en las que se puede encontrar cierto patrón que se repite) e irregulares. Atendiendo a la posibilidad de cambiar las conexiones entre los nodos, también podemos encontrar redes estáticas o dinámicas [44].

Las redes estáticas pueden, a su vez, dividirse entre redes ortogonales y no ortogonales, atendiendo a su geometría.

Las redes dinámicas pueden clasificarse en redes de medio compartido (buses), monoetapa (barras cruzadas o *crossbar*) y multietapa, y estas últimas en redes bloqueantes, no bloqueantes y reconfigurables, atendiendo a la disponibilidad de caminos para establecer una nueva conexión entre una entrada y una salida libre habiendo conexiones en curso.

Las técnicas de conmutación típicas para hacer llegar la información a su destino, [45], son las de conmutación con almacenamiento y reenvío (con la que los paquetes se almacenan y se reenvían en los nodos intermedios hasta alcanzar su destino, también conocida como conmutación de paquetes), conmutación vermiforme o en forma de gusano (donde el camino es segmentado entre el origen y el destino), conmutación virtual (los paquetes son almacenados en nodos intermedios si no es posible continuar con la transferencia hasta su

destino) y conmutación de circuitos (en la que primeramente se establece el circuito y a continuación se transmite la información).

La topología de una red puede ofrecer múltiples caminos para hacer llegar la información de un nodo origen hacia uno destino. El encaminamiento lleva a cabo esta determinación y debe evitar interbloqueos [46].

2.1.3. Redes heterogéneas

Una red se califica de heterogénea cuando sus elementos poseen distintas características. Habitualmente las redes heterogéneas hacen referencia a sistemas en los que los elementos de proceso tienen distinta característica computacional. En un sentido más amplio, debemos considerar que las redes de interconexión tienen tramos con características distintas, pueden existir distintos sistemas operativos, distintos protocolos de comunicaciones, representación de datos, etc. En nuestro estudio supondremos que un sistema heterogéneo posee elementos de proceso con características diferentes, siendo la red de interconexión subyacente homogénea.

Para el tipo de sistemas con los que vamos a trabajar, en el fondo, no existen grandes diferencias en su concepción. Cuando elaboramos un algoritmo paralelo y se plasma sobre una red homogénea, es necesario el haber realizado previamente un estudio sobre el equilibrado de la carga para obtener las máximas prestaciones posibles. Para un sistema heterogéneo, el proceso es idéntico, solo que considerando que los elementos de proceso tienen potencialmente una velocidad computacional distinta. Si consideráramos que la red de interconexión es también heterogénea, se añade una dimensión más a la complejidad del equilibrado de la carga.

2.1.4. Plataformas de pruebas

Una de las plataformas utilizadas para la evaluación de los algoritmos paralelos diseñados ha sido un multiprocesador cc-NUMA con procesadores Itanium 2 a 1,3 GHz

(hasta 16 procesadores disponibles para un usuario) con sistema operativo Linux de 64 bits. Concretamente es el modelo Altix 3000 de la firma SGI perteneciente a la Universidad Politécnica de Valencia. Los resultados que hacen referencia a esta máquina son identificados por *Aldebarán*, por ser el nombre de la máquina.

Este sistema se clasifica dentro de los denominados DSM (Distributed Shared Memory), donde la memoria es físicamente distribuida, pero lógicamente compartida mediante mecanismos de coherencia de *cache*, basados en directorios. La interconexión entre los nodos se basa en *crossbar* modulares con topología de hipercubo. Cada nodo posee cuatro procesadores Itanium 2, agrupados de dos en dos, compartiendo un bus de ancho de banda 6,4 GB/s, 8 bancos de memoria (de 4 a 32 GB) y cada nodo se puede comunicar con otros nodos con enlaces a 1,6 GB/s bidireccional.

El procesador Itanium 2 posee una arquitectura segmentada con 8 etapas y 11 unidades funcionales: dos para procesamiento de enteros, cuatro para operaciones de acceso a memoria, tres para procesamiento de saltos y dos para operaciones de coma flotante. Posee tres niveles de *cache*: L1, separada en dos *caches* de 16 kB, una para datos y otra para instrucciones; L2 con 256 kB y L3 con 3 MB, estas últimas unificadas. Las latencias de *caches* son de 1 ciclo para L1, 5, 7, 9 ó más ciclos para L2 y de 12 en adelante para L3. La arquitectura implementada se denomina IA-64 que utiliza una serie de conceptos denominados EPIC (*Explicit Parallel Instruction Computation*) para aprovechar el máximo paralelismo entre instrucciones (ILP) a través de mecanismos mediante los cuales el compilador reordena el código a partir del conocimiento global del mismo, planifica el empleo de recursos (registros y unidades funcionales) suficientes para realizar operaciones en paralelo y guardar resultados intermedios. Todo ello requiere el empleo de formatos de instrucciones VLIW para que el compilador pueda comunicar al hardware información clave del programa compilado de manera eficiente.

Adicionalmente se han llevado a cabo pruebas puntuales en un cluster formado por dos SMP cuyos procesadores son Intel Xeon a 2,2 GHz y 512 kB de memoria *cache* con sistema operativo Red Hat Linux 9.0, y cuya denominación fue *Kubrick* (*Kubrick-01* y

Kubrick-02).

2.2 Programación en paralelo

2.2.1. Memoria distribuida

El paradigma básico de paso de mensajes lo podemos encontrar implementado en numerosas bibliotecas: Express [47], PARMACS [48, 49], Chameleon [50], CHIMP [51, 52], PICL [53], Zipcode [54, 55, 56], p4, [57, 58], PVM [59, 60], MPI [61, 62, 63, 64], ...

Nuestros algoritmos se apoyan en la biblioteca MPI debido a su actualidad y disponibilidad de distribuciones en prácticamente cualquier arquitectura. De alguna forma, está inspirado en las primeras bibliotecas citadas anteriormente. En 1995 se estandariza MPI como MPI1.1. Existen varias implementaciones que cumplen con el estándar como MPICH del Argonne National Laboratory y Mississippi State University y LAM-MPI del Ohio Supercomputing Center y Open System Laboratory de Indiana University. En 1997 surgen nuevas extensiones de MPI1.1 conocidas como MPI2.0, incluyendo interfaces para Fortran 90 y C++ entre otras características.

La distribución empleada, MPICH, no es de tipo *thread safe*, es decir, en términos generales, dos o más hilos o tareas que pertenezcan al mismo proceso no deben ejecutar simultáneamente llamadas a las rutinas de la biblioteca, en términos generales, ya que ello produciría resultados imprevisibles. Esto no tiene trascendencia en algunos de los algoritmos implementados, pero sí en otros, en los que se podría replantear las tareas previstas desde otra perspectiva. Existen algunas distribuciones de MPI que sí que son *thread safe*, como por ejemplo OpenMPI, [65, 66]. Nos planteamos como línea futura de trabajo el realizar las modificaciones oportunas de los algoritmos implementados para intentar nuevas mejoras sobre los mismos empleando estas nuevas distribuciones.

MPI permite comunicaciones punto a punto mediante operaciones de envío y recepción [63]. Éstas pueden ser bloqueantes o no bloqueantes, permitiendo estas últimas el solapa-

miento entre el cálculo y las comunicaciones. Las operaciones de envío, tanto bloqueantes como no bloqueantes, pueden ser tamponadas, síncronas, preparadas o estándar (en este caso el sistema es quien decide cuál es efectivamente el tipo de envío seleccionado). Para el primer tipo, el vector de elementos a enviar se copia en un almacenamiento intermedio, permitiendo el retorno de la rutina de envío de manera inmediata; para el segundo caso, el sistema espera a que se haya efectuado una llamada de recepción desde el destino y para el último, se supone que el destino está ya preparado para recibir la información. El comportamiento de un algoritmo paralelo viene condicionado en cierta manera según el tipo de envío/recepción escogido.

MPI también permite operaciones colectivas, como por ejemplo barreras de sincronización, difusiones, recolecciones, distribuciones, operaciones de reducción como máximos, mínimos, sumas, definidos por el usuario, ...

Mediante conceptos como *grupos*, *contextos* y *comunicadores*, en MPI se pueden plantear subconjuntos de nodos entre los que se forman especies de subredes con semejantes características y operaciones a la red global, así como topologías virtuales con las que se crean conexiones lógicas preestablecidas entre ciertos nodos.

Nuestras implementaciones requerirán sólo operaciones elementales dentro del contexto de MPI, como son envíos y recepciones bloqueantes y no bloqueantes, barreras, difusiones y recolecciones.

2.2.2. Memoria compartida

El paradigma de memoria compartida es completamente distinto al de memoria distribuida. La idea básica se centra en ejecutar en una serie de hilos o *threads*, las tareas concurrentes que expresa el algoritmo paralelo, dentro del mismo espacio de memoria. El algoritmo paralelo debe controlar el acceso de lectura y escritura a la zona de variables compartidas para evitar los problemas que originan las condiciones de carrera y las dependencias de los datos, evitando, por tanto, incorrecciones en la ejecución.

Una referencia en cuanto al intento de estandarización para esquemas de programación

en memoria compartida fue el estándar ANSI X3H5, [67], que jamás se llevó realmente a la práctica. Éste, de hecho, fue un punto de partida para que entre los años 1996 y 1997 llegara definitvamente el estándar OpenMP. En [68], [69], [70] podemos encontrar la descripción de OpenMP, así como numerosos ejemplos de programación, incluso conjuntamente con MPI exponiendo las condiciones necesarias para que pueda llevarse a cabo tal cooperación.

Algoritmos paralelos para memoria distribuida en memoria compartida y esquemas híbridos

Un algoritmo paralelo concebido para una arquitectura de memoria distribuida puede, bajo ciertas condiciones, ejecutarse en un sistema que posea una arquitectura de memoria compartida, si se proveen los mecanismos necesarios para emular las comunicaciones (de hecho, ciertas distribuciones de MPI pueden emular el paso de mensajes mediante mecanismos basados en memoria compartida en vez de TCP). De manera que cada proceso se traduciría en un hilo o *thread* y los mecanismos de transmisión o recepción serían los provistos para tal efecto. A la inversa es más complejo, a veces ineficiencte o incluso imposible, debido a que la traducción de los mecanismos provistos para memoria compartida no tienen un fiel reflejo en memoria distribuida.

Podemos emplear sistemas con esquemas híbridos: memoria distribuida en los nodos de cálculo y dentro de ellos, varios elementos de proceso compartiendo memoria (como por ejemplo un *cluster* de SMP). De modo que podemos diseñar el algoritmo de manera regular, suponiendo que todos los elementos de proceso son independientes, pero luego implementar las comunicaciones entre elementos dentro de un mismo nodo utilizando mecanismos de memoria compartida, y entre elementos de proceso de distintos nodos, utilizando los de memoria distribuida.

Como un elemento de proceso dentro de un nodo puede tener que atender a comunicaciones vía memoria distribuida y memoria compartida, en términos generales, la distribución de la biblioteca de comunicaciones, MPI por ejemplo, debe ser *thread safe* para permitir todas las posibilidades en cuanto a las comunicaciones.

2.2.3. Metodología en el diseño de algoritmos paralelos

Nos basaremos en algunas ideas expuestas en [71], [72] y [73], donde se muestran ciertas metodologías para llevar a cabo un diseño eficiente de un algoritmo paralelo.

Inicialmente realizaremos una partición de los datos, donde las operaciones a realizar sean más o menos regulares, identificando clases de particiones y posteriormente subparticiones de tamaño sin determinar (para posteriormente concretarlas al llevar a cabo un equilibrado de la carga). A continuación estableceremos el grafo de dependencias con el que se pretenderá indicar qué partición requiere qué resultados de las operaciones realizadas en otras particiones. Podemos destacar dos tipos de dependencias: las que podríamos calificar de convencionales y las co-dependencias. En una dependencia convencional, para elaborar el trabajo que requiere una partición, se requiere los resultados elaborados de la partición de la que depende; en una co-dependencia, las particiones implicadas necesitan recíprocamente sus datos para elaborar la tarea.

Posteriormente, identificaremos qué tareas pueden llevarse a cabo concurrentemente para asignarlas a los procesadores. En ocasiones nos resultará cómodo tener una visión inversa: convencionalmente, las particiones o ciertos conjuntos de datos son transferidos de un procesador a otro; en algunos algoritmos, resulta cómodo y más sencillo entender el comportamiento del mismo o plantear su diseño suponiendo que son los procesadores los que se *mueven* a lo largo de las particiones. Dicho de otro modo, las particiones son *ocupadas* por los procesadores, realizan su tarea, abandonan la partición y se dirigen hacia otra partición. Evidentemente, en un instante dado, un procesador puede ocupar más de una partición, pero una partición no podrá estar en más de un procesador. Cuando un procesador salte de una partición a otra que estuvo recientemente ocupada por otro, se interpreta en la realidad como que este último ha transferido la información de la partición al primero.

2.2.4. Algoritmos segmentados

Los algoritmos que trataremos en el presente trabajo poseen una secuencialidad intrínseca que los hace difícilmente paralelizables a priori, si nos planteamos como objetivo primordial el obtener unas eficiencias relativamente elevadas, lo cual se traduce en el deseo de obtener una baja sobrecarga aritmética por paralelización y tener a todos los procesadores ocupados y equilibrados.

La solución que proponemos para abordar este tipo de problemas consiste en segmentar el algoritmo paralelo, siempre y cuando sea posible. En [74] se muestran conceptos básicos y el modelado de este tipo de algoritmos calificados como *segmentados de grano grueso* en arquitecturas de tipo memoria distribuida. En [71] se describe el modelo segmentado como un caso particular del modelo productor-consumidor en el que cada elemento de proceso asume ambos roles. Las líneas segmentadas de elementos de proceso pueden ser lineales, multidimiensionales o grafos dirigidos. En nuestros algoritmos, intentaremos que los modelos segmentados sean lineales para hacer mínima la complejidad en la organización de las comunicaciones.

Tomando como referencia el algoritmo secuencial, el proceso consistirá en segmentar cada iteración de las que consta el algoritmo secuencial, asignando cada segmento a un procesador. En los siguientes subapartados abordaremos la segmentación de bucles y las condiciones algorítmicas que deben darse para que pueda llevarse a cabo. Posteriormente trataremos su posible implementación en arquitecturas de memoria distribuida, de memoria compartida y con esquemas híbridos.

Segmentación de bucles

Para cada iteración i-ésima de un bucle supondremos que existen tres fases claramente diferenciadas: una inicialización, un proceso y una extracción de solución. Por claridad a la vez que generalidad, supondremos que el proceso consiste en una secuencia de operaciones que pueden expresarse, a su vez, en forma de bucle interno cuyas iteraciones serán deno-

tadas con el índice j; por generalizar, podremos suponer que el proceso que se realiza en cada iteración interna puede ser distinto. Una clasificación de los tipos de datos genéricos para los algoritmos a tratar es la siguiente:

- *Entradas*: $\mathbf{E}_i$ para cada iteración del algoritmo. Estos valores pueden sufrir un *preproceso* cuyo resultado se denotará por $\mathbf{B}_{j_{\text{inic}}}$.

- *Variables intermedias*: $\mathbf{B}_j$, es decir, información que se genera en la iteración interna j-ésima y es necesaria para la $(j+1)$-ésima.

- *Estado*: $\mathbf{A}_i$, es decir, información que se genera en la iteración i-ésima y es necesaria para la $(i+1)$-ésima. Supondremos que el estado puede expresarse en forma de vector o matriz y que en cada iteración interna j-ésima sólo se accede a cierta parte exclusiva de dicho estado, lo cual denotaremos por $\mathbf{A}_i(j)$.

- *Solución*: $\mathbf{x}_i$, es decir la solución en la iteración i-ésima, basada en resultados procedentes de la iteración anterior, la última instancia de las variables intermedias y del estado del sistema.

Los algoritmos que vamos a tratar tendrán genéricamente la siguiente estructura:

Entrada: Estado inicial: $\mathbf{A}_{j_{\text{inic}}-1}$, entrada de datos: $\mathbf{E}_i$, cada iteración

Salida: Actualización de la solución $\mathbf{x}_i$ cada iteración i-ésima

para $i = 0, \ldots,$ **hacer**

$\mathbf{B}_{j_{\text{inic}}-1} = f_{\text{entrada}}(\mathbf{E}_i)$

para $j = j_{\text{inic}}, \ldots, j_{\text{fin}}$ **hacer**

$[\mathbf{A}_i(j), \mathbf{B}_j] = f_{\text{proceso}_j}(\mathbf{A}_i(j), \mathbf{B}_{j-1})$

fin para

$\mathbf{x}_i = f_{\text{solución}}(\mathbf{x}_{i-1}, \mathbf{A}_i, \mathbf{B}_{j_{\text{fin}}})$

fin para

Esta estructura contempla que:

- Las entradas inicializarán, con el preprocesado oportuno, alguna o algunas variables intermedias.
- El proceso consiste en realizar un conjunto de operaciones, denotadas por el bucle con índice j, al estado y a las variables intermedias. Para generalizar, el proceso puede ser distinto en cada iteración interna, por lo que se ha añadido el subíndice j.
- En este proceso, para cierto valor del índice del bucle j, sólo parte del estado íntimamente relacionado con dicho índice es actualizado.
- $\mathbf{B}_j$ es actualizado y su resultado es requerido en la siguiente iteración interna

En estos ítems se aprecia una dependencia de datos de tipo *flujo* acarreada en las iteraciones del bucle interno para las variables intermedias, lo cual hace problemática la paralelización, ya que una iteración interna o externa no podrá comenzar mientras no hayan finalizado todas las anteriores.

La solución que se propone es segmentar cada iteración del bucle. El algoritmo secuencial segmentado puede expresarse de la siguiente forma:

Entrada: Estado inicial: $\mathbf{A}_{j_{\text{inic}}-1}$, entrada de datos: $\mathbf{E}_i$, cada iteración

Salida: Actualización de la solución $\mathbf{x}_i$ cada iteración i-ésima

para $i = 0, \ldots,$ **hacer**

$\mathbf{B}_{j_{\text{inic}}-1} = f_{\text{entrada}}(\mathbf{E}_i)$

para $j = j_{\text{inic}}, \ldots, j_k$ **hacer**

$[\mathbf{A}_i(j), \mathbf{B}_j] = f_{\text{proceso}_j}(\mathbf{A}_i(j), \mathbf{B}_{j-1})$

fin para

$\vdots$

para $j = j_l, \ldots, j_m$ **hacer**

$[\mathbf{A}_i(j), \mathbf{B}_j] = f_{\text{proceso}_j}(\mathbf{A}_i(j), \mathbf{B}_{j-1})$

fin para

$\vdots$

para $j = j_n, \ldots, j_{\text{fin}}$ **hacer**

$[\mathbf{A}_i(j), \mathbf{B}_j] = f_{\text{proceso}_j}(\mathbf{A}_i(j), \mathbf{B}_{j-1})$

fin para

$\mathbf{x}_i = f_{\text{solución}}(\mathbf{x}_{i-1}, \mathbf{A}_i, \mathbf{B}_{j_{\text{fin}}})$

fin para

Según se desprende del algoritmo, cada *segmento* sólo requiere del *segmento* anterior el resultado final de $\mathbf{B}_j$. Esto ya da lugar a plantear el esqueleto de los algoritmos paralelos que vamos a tratar, asignando cada segmento a un procesador, y requiriendo como comunicación exclusivamente unidireccional entre segmentos contiguos sólo la variable $\mathbf{B}_j$ resultante del proceso en el segmento origen. Cada procesador tendrá la fracción del estado correspondiente al intervalo de valores del índice del bucle interno. El cálculo de la solución puede requerir la *recolección* de todo o parte del estado opcionalmente, la última instancia de las variables intermedias, así como la solución anterior, aunque esto depende de cuáles sean los detalles de cada algoritmo. La figura 2.1 muestra gráficamente el proceso de segmentación algorítmica y la figura 2.2 muestra un ejemplo en el que se compara gráficamente la evolución del algoritmo secuencial (izquierda), con las entradas ($\mathbf{H}_i$ e $\mathbf{y}_i$) y las salidas $\mathbf{x}_{\text{LS}_i}$. En este ejemplo gráfico podemos apreciar cómo una iteración se *segmenta* en tantas porciones como procesadores existen, ejecutándose de manera escalonada en el tiempo. También se observa cómo la figura se corresponde con un sistema heterogéneo perfectamente balanceado, en el que todos los procesadores finalizan su segmento de iteración en el mismo instante de tiempo, tras lo cual, se procede a transmitir los resultados al siguiente procesador de la cadena segmentada.

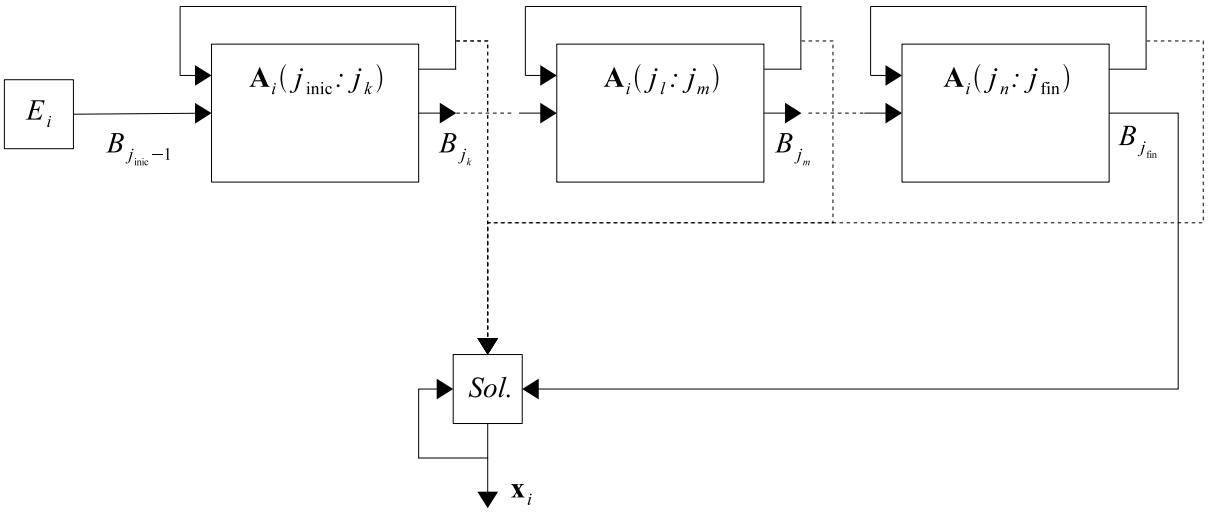

Figura 2.1: Segmentación algorítmica

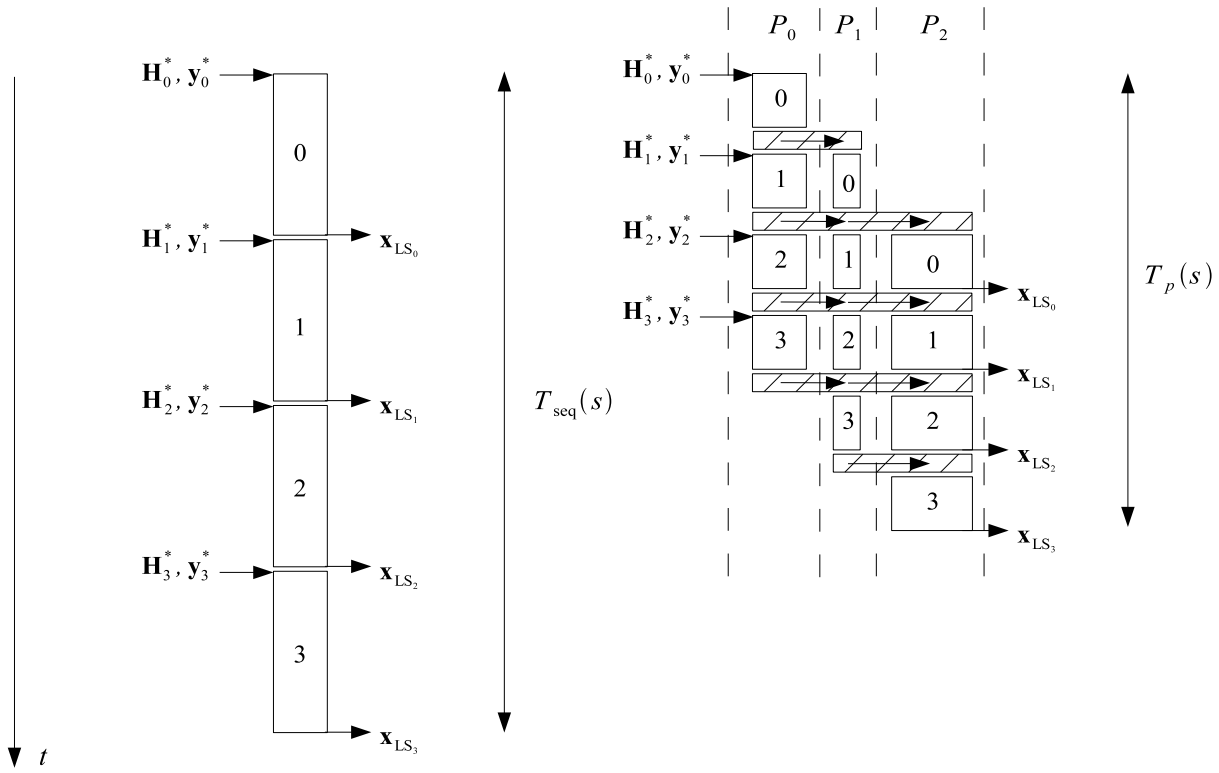

Figura 2.2: Ejemplo gráfico de algoritmo paralelo segmentado.

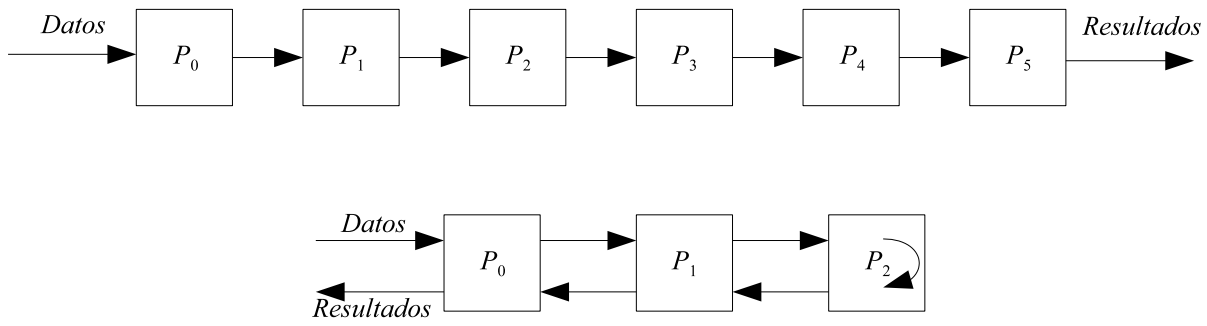

Figura 2.3: Línea segmentada unidireccional y bidireccional

Líneas segmentadas bidireccionales

En ocasiones es difícil equilibrar la carga en un algoritmo paralelo. Esto es debido a la diferencia que probablemente existirá entre el modelo analítico de tiempo de ejecución del que se suele obtener el criterio de equilibrado y la realidad. Existen algoritmos paralelos en los que las particiones de los datos tienen cierta forma específica que hace que el equilibrado sea relativamente fácil si dos o más particiones dispuestas de manera *especular* o *simétrica* son asignadas al mismo procesador. Dada esta disposición y asignación de las particiones, las líneas segmentadas obligatoriamente dejan de ser *unidireccionales* y pasan a ser *bidireccionales* debido a que un mismo procesador asume el *rol* de dos procesadores de la línea segmentada, tal y como puede observarse en la figura 2.3.

La línea segmentada bidireccional requiere el atender a dos flujos de información independientes. Ello puede llevarse a cabo en la práctica empleando sendos *hilos*. Algunas distribuciones de MPI no son *thread safe* o *seguras a nivel de hilo*, por lo que deberemos emplear *cerrojos* o *secciones críticas*, de modo que un hilo sólo pueda acceder a la parte de comunicaciones cuando ningún otro lo esté haciendo. Esto puede plantear problemas cuando se emplean comunicaciones de tipo no bloqueantes; para las comunicaciones de tipo bloqueantes existe la posibilidad de que se colapse la red. La figura 2.4 muestra en forma de diagrama de flujo una posible solución, para cuando MPI no es *thread safe* y que permite emplear comunicaciones no bloqueantes, intentando solapar al máximo posible, según la máquina, las comunicaciones y el cálculo.

El canal de *ida* viene denotado por el subíndice `A` y el de *vuelta* por `B`. El prefijo `I` denota envío o recepción no bloqueante (*immediate*); el recuadro con `C` indica las operaciones aritméticas a realizar por el procesador. En el rombo `¿RECV_A/B?` se comprueba si se ha recibido algo por el canal `A/B`; en el rombo `¿SEND_A/B?` se comprueba si ya se ha mandado definitivamente la información. La variable booleana `Recibiendo_A/B` y el rombo `¿Recibiendo_A/B?` sirven para retornar al punto oportuno cuando se ha finalizado el proceso en el canal opuesto. La intención de esta organización del código es atender a la información entrante lo más rápidamente posible y evitar cualquier bloqueo en las comunicaciones.

Este esquema, aunque salva el problema de trabajar con versiones MPI de tipo *no thread safe*, presenta el inconveniente de no atender a un canal, mientras está llevando a cabo operaciones aritméticas con información del otro canal, por el tratamiento secuencial del procedimiento. Surge una alternativa a este problema que consiste en *mapear* dos procesos MPI en el mismo procesador, atendiendo cada uno a cada una de las particiones especulares o simétricas previstas. Se resuelve así la falta de simultaneidad en el tratamiento del canal de ida y de vuelta. En cualquier caso, no podemos asegurar que esta solución sea extrapolable a cualquier sistema debido a dos motivos principalmente: existen sistemas en los que no es posible realizar dicho *mapeo* y en otros, hemos detectado situaciones en las que la atención del sistema operativo a uno de los procesos MPI es notablemente retardada, por lo que se rompe el equilibrio de carga previsto en el tiempo.

Líneas segmentadas en memoria distribuida

Cuando tenemos un sistema paralelo con una arquitectura con memoria distribuida, la organización de las comunicaciones de una línea segmentada es obvia. Cuando un procesador finalice su *segmento* de operaciones, envía su resultado al siguiente procesador de la cadena y recibe los del anterior. Dentro de este planteamiento surgen numerosas alternativas de implementación que suelen afectar al rendimiento final, como por ejemplo si las transmisiones van a ser *tamponadas* o van a ser síncronas, bloqueantes o no bloqueantes,

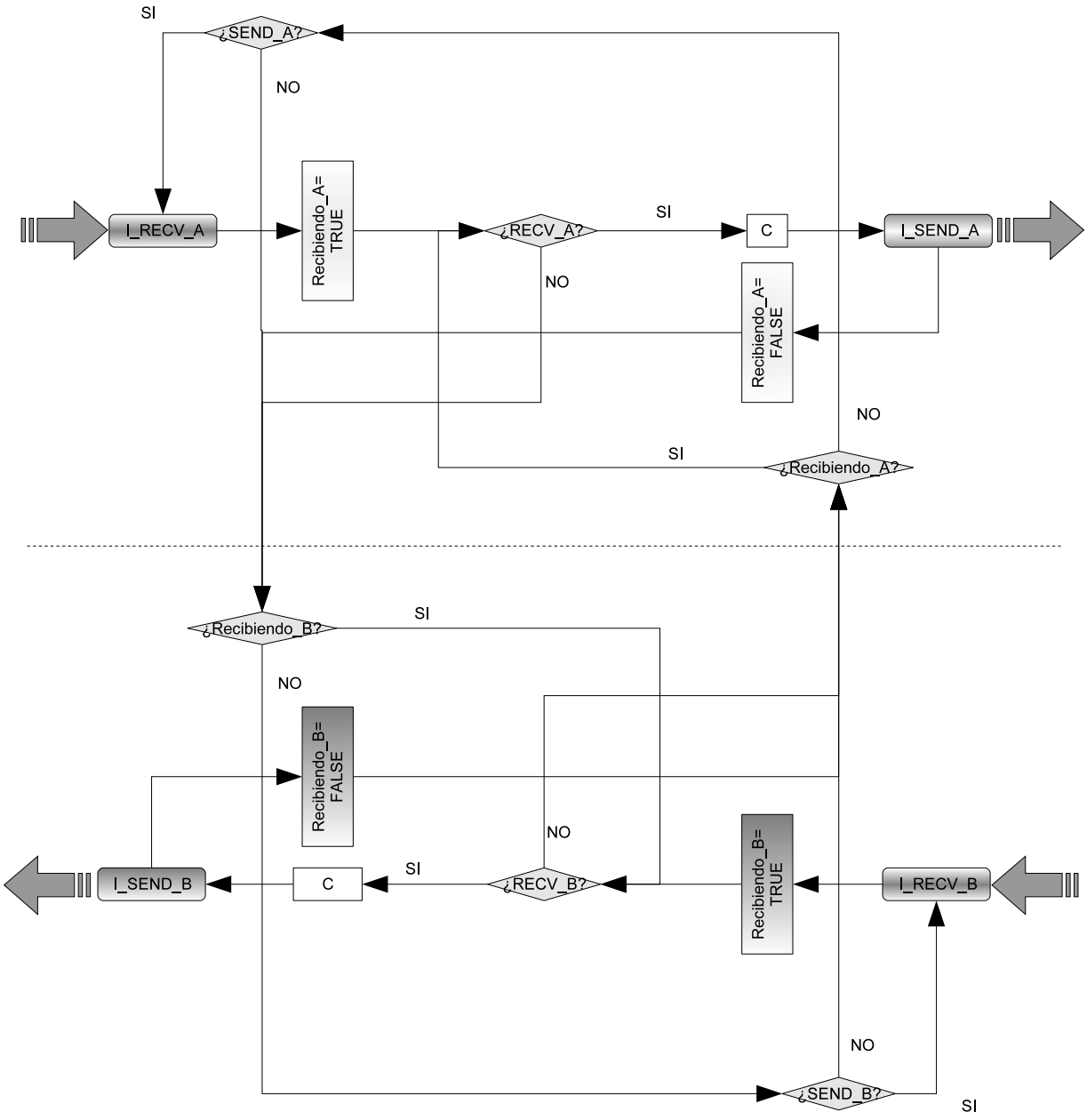

Figura 2.4: Atención de dos flujos de información con MPI no *thread safe* solapando cálculo y comunicaciones.

con o sin solape entre el cálculo y las comunicaciones, etc.

Líneas segmentadas en memoria compartida

Cuando el algoritmo paralelo se concibe para arquitecturas de memoria distribuida, en ocasiones su *traducción* a un esquema de memoria compartida no resulta inmediata, requiriendo modificaciones en el planteamiento del nuevo algoritmo. Este no suele ser el caso de los algoritmos segmentados regulares en los que la organización de las comunicaciones es relativamente simple, ya sea de manera segmentada unidireccional o bidireccional. La traducción, por tanto, se reduce a transmitir de manera eficiente una estructura de datos, o vector, entre una tarea y la que se considere consecutiva dentro de la línea segmentada. El problema del equilibrado de carga entre las tareas queda resuelto con el esquema planteado para memoria distribuida, requiriendo algún tipo de sincronización explícita entre las distintas tareas.

Siempre existe la solución que da MPI de emplear la memoria compartida como sistema de comunicaciones por paso de mensajes, pero nuestra pretensión futura es comparar, para algunos casos, la solución MPI con una solución que emplee las directivas de compilación de OpenMP para realizar esta transferencia de información y comparar los resultados.

Las transferencias de una tarea a la siguiente pueden llevarse a cabo disponiendo de un vector con tantas componentes como hilos o procesadores con memoria compartida existan. Los elementos de este vector tendrán la misma estructura que los datos que se transfieren entre los procesadores de una arquitectura con memoria distribuida, es decir, la estructura de datos $\mathbf{B}_j$ mostrada en el subapartado de **Segmentación de bucles** (página 22). Este vector será un vector compartido por todas las tareas, y cada tarea apuntará en un momento dado a uno de los elementos del vector.

Cuando una tarea finalice su proceso, liberará la posición a la que apunta, pudiendo la siguiente tarea de la línea segmentada apuntar a ella y procesarla. La primera apuntará entonces a la posición que eventualmente liberó la tarea que le precede en la línea segmentada, y así sucesivamente.

Los problemas que hay que solucionar son, igualmente que en el esquema con memoria distribuida, el desequilibrado de la carga que haga que la sobrecarga por tiempo de sincronización sea mínima.

Líneas segmentadas en arquitecturas híbridas

Podemos unir las ideas expuestas para la implementación de líneas segmentadas en memoria distribuida y en memoria compartida de una manera relativamente sencilla, habida cuenta de la regularidad en la organización de las comunicaciones. Es decir, si la arquitectura diferencia claramente los elementos de proceso que comparten memoria y los que no, se puede plantear el esquema descrito para memoria compartida para los primeros elementos, y el de memoria distribuida para unir elementos de proceso que no la comparten.

2.2.5. Herramientas de programación utilizadas

El lenguaje de programación para la codificación de los algoritmos ha sido Fortran 90, [75], con algunas extensiones de Fortran 95, [76], utilizando el compilador de Intel ® Fortran Compiler 9.0 (utilizando opciones de compilación `OpenMP` para las versiones algorítmicas de memoria compartida). Para las versiones por paso de mensajes o memoria distribuida hemos utilizado las bibliotecas `MPICH-1.2.7` en la máquina *Kubrick* y una versión propietaria de MPI en *Aldebarán*. Para ambas máquinas hemos utilizado la biblioteca `BLAS` el Intel ® Math Kernel Library 8.0 sin *threading*.

2.3 Evaluación de prestaciones

2.3.1. Conceptos básicos en la evaluación de algoritmos paralelos

La evaluación del rendimiento de un algoritmo paralelo no puede estar desligada de la arquitectura paralela sobre la cual se está ejecutando e incluso del sistema real que imple-

menta dicha arquitectura. En cualquier caso, es nuestro objetivo el tratar de generalizar al máximo las conclusiones y resultados de los experimentos realizados sobre los algoritmos propuestos. Definiremos un sistema paralelo como la combinación de un algoritmo paralelo y una arquitectura paralela.

El tiempo de ejecución de un algoritmo paralelo depende de numerosos factores: del tamaño del problema, del número de procesadores, y de las características del sistema como la velocidad de dichos procesadores, la velocidad de las comunicaciones entre los procesadores, de la topología de la red de interconexión, de las técnicas de *reencaminamiento*, ... Un algoritmo paralelo que se ejecute eficientemente en un sistema con ciertas características dadas, puede que no lo haga si se cambia alguno o algunos de los parámetros del sistema. Es por ello que la evaluación de un algoritmo paralelo requiere un estudio mínimo de sus características.

Existen numerosos conceptos asociados a la evaluación del rendimiento de un algoritmo paralelo que han ido evolucionando con el tiempo como consecuencia de cierta continua reflexión sobre los mismos, y de la propia evolución de la programación paralela y las arquitecturas paralelas. Podríamos citar el *speedup* o incremento de velocidad, la eficiencia, la fracción serie del tiempo de ejecución, la escalabilidad y funciones iso-paramétricas,...

En muchas ocasiones los conceptos empleados no estaban correctamente contextualizados, en otras, estos conceptos han ido evolucionando o transformándose en el tiempo, para poder justificar nuevas ideas o resultados experimentales que no encajaban con las tesis establecidas.

Por ejemplo, el concepto de incremento de velocidad se ha ido matizando con el tiempo. De hecho, aunque el concepto, en principio, es único (ratio entre el tiempo empleado por un sólo procesador en ejecutar el (mejor) algoritmo secuencial para resolver el problema y el tiempo empleado por p procesadores en ejecutar el algoritmo paralelo para resolver el mismo problema), en la bibliografía podemos encontrar varios tipos de incremento de velocidad, como por ejemplo, el *simple*, el *escalado*, el *asintótico*...

La eficiencia se define como el cociente entre el incremento de velocidad y el número de

procesadores. Teóricamente la eficiencia está acotada superiormente por la unidad, aunque en ocasiones se dan circunstancias con las que un algoritmo paralelo puede conseguir eficiencias superiores a la unidad (condición de *superlinealidad*), pero son debidas a cuestiones de diferencias arquitectónicas entre la ejecución del algoritmo paralelo y secuencial y no propiamente al paralelismo de un algoritmo.

Dado un problema de tamaño fijo, si incrementamos el número de procesadores que puede emplear un algoritmo paralelo, en principio obtendremos un menor tiempo de respuesta, pero a costa de una menor eficiencia, debido a cierta sobrecarga (*overhead*) inherente a la ejecución de un algoritmo paralelo, que además se suele incrementar con el número de procesadores. De hecho, existe la posibilidad de que aumentando el número de procesadores, eventualmente se llegue a cierto mínimo tiempo de ejecución, y al ir añadiendo más procesadores, dicho tiempo comience a aumentar, [77].

Existe otro concepto que, a su vez, ha tenido varias interpretaciones a lo largo del tiempo: la *fracción serie* de un algoritmo. Se define como la fracción de tiempo de ejecución de un algoritmo secuencial que no se puede paralelizar, o bien, como la fracción de tiempo de ejecución de un algoritmo paralelo que obligatoriamente se debe ejecutar secuencialmente. Como comprobaremos posteriormente, este matiz dará lugar a varios tipos de incrementos de velocidad. En [78] podemos encontrarnos una descripción de su obtención práctica y el empleo de este concepto como herramienta de diagnóstico del algoritmo paralelo.

La publicación de Amdahl en 1967 [79] marca una referencia en la evaluación de algoritmos paralelos. Surge el concepto de incremento de velocidad *simple* argumentando que, considerando fijo el tamaño de cierto problema a resolver, el incremento de velocidad estaba acotado superiormente por la inversa de la fracción serie del algoritmo. Amdahl no tuvo en cuenta ciertos factores como la sobrecarga por paralelización (*overhead*), topologías, tamaño variable del problema,.... Gustafson en el año 1988 [80] indicó, para justificar ciertos resultados que se obtuvieron y por tanto como réplica a la publicación de Amdahl, que si concebimos un sistema multiprocesador como un sistema que pretende abordar problemas de tamaño cada vez mayor, y no como un sistema que pretende abordar

el mismo problema en menor tiempo, habría que reconcebir ciertas ideas, dando lugar al concepto de *incremento de velocidad escalado*.

Las anteriores ideas nuevas hicieron que se debieran reconcebir, redefinir o añadir nuevos términos en el campo de la evaluación de algoritmos paralelos. Siguiendo las ideas de Gustafson, si incrementamos el número de procesadores de un sistema, debemos permitir que aumente el tamaño del problema para poder evaluar de una manera más equitativa las características de la nueva combinación arquitectura paralela-algoritmo paralelo.

De alguna manera, surge la necesidad de enfocar el potencial de cálculo de los sistemas paralelos en problemas de gran tamaño. Surge el concepto de *escalabilidad* como forma de predecir el comportamiento de sistemas masivamente paralelos a partir de comportamientos en sistemas a menor escala. Cuando se incrementa el número de procesadores, los parámetros de un sistema paralelo cambian (por ejemplo, generalmente la eficiencia disminuye así como el tiempo de cálculo). Es decir, pueden surgir situaciones de compromiso entre los distintos parámetros en los que podemos estar interesados. Se puede definir una serie de *funciones iso-paramétricas* que relacionan el incremento necesario del tamaño del problema con el número de procesadores para mantener cierto parámetro constante. Ejemplos de parámetros pueden ser: tiempo de ejecución, velocidad de ejecución, eficiencia, memoria, ... A partir de estas funciones se puede establecer una *métrica de la escalabilidad*.

2.3.2. Modelos de tiempo de ejecución de algoritmos paralelos

A lo largo de toda la bibliografía y de su evolución en el tiempo, se ha intentado emplear modelos de tiempos de ejecución cada vez más completos a la par que genéricos y al mismo tiempo sencillos. Estos modelos de tiempo de ejecución deben utilizarse como meras referencias, ya que son demasiado simplificados como para expresar toda la complejidad de la ejecución de un algoritmo sea secuencial o paralelo, por lo que a buen seguro, encontraremos diferencias entre los tiempos de ejecución reales y los modelos, que deberemos intentar contrastar y justificar en la medida de lo posible.

A continuación exponemos los conceptos que estimamos más interesantes para estos modelos. Posteriormente se irán simplificando, en algunos casos, algunos de estos conceptos.

- z: tamaño del problema. Este término implica cierta confusión ya que es un concepto asociado a la aplicación que se pretende resolver. Dependiendo de las referencias bibliográficas, el concepto *tamaño del problema* hace referencia a la cantidad de operaciones que hay que realizar para resolver el problema, o bien, al tamaño que los datos ocupan en memoria, o bien a un parámetro concreto de la estructura de datos que necesita el problema para ser definido. Por ejemplo, si el algoritmo pretende multiplicar dos matrices cuadradas de dimensión n, en función del contexto, *el tamaño del problema* hace referencia a $z = \mathbf{\Theta}(n^3)$ (cantidad de operaciones necesarias para llevar a cabo la multiplicación matricial), o bien a la cantidad de memoria necesaria $z = n^2$, o simplemente a la dimensión de la matriz $z = n$. Si en este ejemplo, la matriz no fuera cuadrada ($m \times n$, $m \neq n$), entonces z vendría dada en función de ambos parámetros. Aunque es una cuestión de definición, es un término que siempre conviene clarificar. En nuestro caso, y en términos generales, el tamaño del problema vendrá definido por las dimensiones de las matrices implicadas y de cierto parámetro, como posteriormente comprobaremos, ligado al tamaño de bloque a procesar. Todos estos parámetros los sintetizaremos en uno sólo: z.

- W: la cantidad de operaciones *elementales* (operaciones en coma flotante o instrucciones o, ...) que hay que realizar sobre los datos de entrada para resolver el problema; de esta forma se independiza el concepto de *tamaño del problema* del problema a resolver. Es por tanto, la cantidad de operaciones que debe realizar el algoritmo secuencial escogido como referencia para resolver el problema. En cualquier caso la cantidad de operaciones W será función del tamaño del problema z, tal y como hemos puntualizado en el apartado anterior, lo cual será denotado genéricamente como $W(z)$, o bien $W(m, n, \ldots)$, dependiendo de lo explícito que se desee ser.

- p: número de procesadores

- $T_{\text{sec}}(z)$ el tiempo de ejecución del algoritmo secuencial o tiempo de ejecución serie. Dependiendo del problema, podemos encontrarnos con más de un algoritmo secuencial que resuelve el problema. Con algún tipo de criterio se debe escoger el algoritmo de referencia para obtener este tiempo de ejecución, siendo el más habitual el algoritmo más rápido. Podemos escribir dicho tiempo como producto de la cantidad de operaciones elementales necesarias $W(z)$ para resolver el problema por el tiempo necesario de cada operación elemental, t_w (s/operación) —valor que si no se supone constante, dependerá del tamaño del problema: $t_w(z)$—, por lo que $T_{\text{sec}}(z) = W(z)t_w(z)$. La inversa de t_w se le suele conocer como *velocidad* (operaciones/s). t_w puede depender de z si el modelo *plano* de acceso a memoria no puede aplicarse debido a la presencia de la memoria *cache* [81]. Llamadas a subrutinas, inicialización de bucles, ..., no tienen el mismo impacto relativo en problemas de alta complejidad que en problemas de baja complejidad, [82]. Cuando se aproxima o se supone que t_w no depende de z, en ocasiones se normaliza el resultado ($t_w(z) = 1$ s/operación), por lo que $T_{\text{sec}}(z) \equiv W(z)$.

- $T_{\text{par}}(z, p)$ el tiempo de ejecución del algoritmo paralelo (en p procesadores). El tiempo paralelo suele descomponerse en una suma de tiempos asociados a distintos conceptos. Uno de ellos es el tiempo aritmético en un procesador $T_{A,P_j}(z)$

$$T_{A,P_j}(z) = W_{P_j}(z)t_{w_j}, j = 0, \ldots, p-1$$

siendo $W_{P_j}(z)$ la carga aritmética asignada al procesador P_j y t_{w_j} el tiempo de operación elemental en el procesador P_j.

Otro concepto importante es el tiempo destinado a las comunicaciones entre los procesadores $T_C(z, p)$ que comentaremos a continuación. Si existe solapamiento entre el cálculo y las comunicaciones, hay que añadir cierto factor negativo en el cómputo

del tiempo paralelo para poder tomar en consideración esta situación.

- $T_C(z,p)$: tiempo de *comunicaciones*, incluyendo el tiempo necesario para transferir información de un procesador origen a uno destino, operaciones de sincronización, etc. Evidentemente dependerá del tamaño del problema, de la cantidad de procesadores, de la topología de la red, de las técnicas de encaminamiento, de la velocidad de transferencia.... Cuando la comunicación es punto a punto, se suele emplear un modelo de función afín para el tiempo empleado en transferir un vector de m elementos:

$$T_C(m) = \beta + \tau m \tag{2.1}$$

donde β es una constante que representa el tiempo de establecimiento de la comunicación, y τ el tiempo en transferir un elemento.

Si la comunicación no es punto a punto, entonces entran en juego distintas posibles técnicas de encaminamiento; en función de la empleada se obtienen unas u otras expresiones para estos tiempos.

La suma de todos los tiempos empleados en las comunicaciones da lugar a la expresión $T_C(z,p)$. Dependiendo del contexto, en ocasiones también se incluye los tiempos de espera en los puntos de sincronización, la aportación negativa del solapamiento con el cálculo, etc.

- $T_O(z,p)$: tiempo de *sobrecarga total* u *overhead*. De alguna forma, se corresponde con el modelado de la cantidad de tiempo empleado en realizar tareas que no se realizan en el algoritmo secuencial, o se realizarían de manera redundante. Podemos encontrar principalmente dos fuentes de sobrecarga [83]:

  - Tiempo extra de ejecución debido a falta de paralelización de ciertas partes del algoritmo, replicación de parte del código...

- Tiempo empleado en las comunicaciones, sincronización, tiempos de espera por desequilibrio de la carga,...

Una interpretación alternativa es que el sistema paralelo resuelve de manera óptima un problema de complejidad superior, es decir:

$$T_{\text{par}}(z,p) = \frac{T_{\text{sec}}(z) + T_O(z,p)}{p} \tag{2.2}$$

En algunas referencias, como en [83], el tiempo de sobrecarga no es el *total*, sino que es el que corresponde a cada procesador, luego $T_{\text{par}}(z,p) = \frac{T_{\text{sec}}(z)}{p} + T_O(z,p)$

En cualquier caso, se considera que el tiempo de sobrecarga es una función monótona creciente, tanto con el tamaño del problema como con el número de procesadores, es decir:

$$T_O(p_i, z_i) \geq T_O(p_j, z_j), \quad \text{con} \quad z_i \geq z_j, \quad \text{y} \quad p_i \geq p_j.$$

Con las anteriores ideas expuestas, el modelo de tiempo de ejecución de un algoritmo paralelo que adoptaremos será aquél que tiene en cuenta el tiempo de ejecución de la carga en cada procesador, $T_{A,P_j}(z,p)$, junto con todos los tiempos de sobrecarga (con todos los conceptos que puede incluir) que para un procesador en particular denotaremos por $T_{O,P_j}(z,p)$. Estos modelos teóricos serán contrastados con medidas experimentales del comportamiento temporal de los algoritmos, tanto en la faceta aritmética como en la de sobrecarga. Por lo que

$$T_{\text{par}}(z,p) = \text{máx}\{W_{P_j}(z)t_{w_j} + T_{O,P_j}(z,p), 0 \leq j \leq p-1\}.$$

Este tipo de modelos oculta algunos detalles de la arquitectura de la máquina donde se ejecuta el algoritmo. En ocasiones convendrá especificar el tiempo de ejecución en función de las llamadas a rutinas de ciertos núcleos computacionales (numéricos en nuestro caso).

De esta forma, para algunas situaciones se pueden realizar algunas comparaciones más sencillas que en el caso del modelo anterior. Las funciones que representan a las llamadas deben estar parametrizadas para poder estimar cuál sería el tiempo de ejecución asociado a dichas llamadas. Este modelo de tiempos de ejecución alternativo lo denominaremos *modelo de llamadas*.

2.3.3. Incremento de velocidad y eficiencia

En el presente subapartado comentaremos dos conceptos útiles para evaluar aspectos cuantitativos del comportamiento de algoritmos paralelos. Éstos son el incremento de velocidad y la eficiencia:

- Incremento de velocidad o *speedup*: cociente entre el tiempo de ejecución del algoritmo serie y el del paralelo.

$$S(z,p) \quad = \quad \frac{T_{\text{sec}}(z)}{T_{\text{par}}(z,p)}$$

  Aunque el concepto de incremento de velocidad es más o menos homogéneo a lo largo de la literatura, existen varias formas de medirlo, dependiendo del contexto y condiciones en el que se realicen las medidas: incremento de velocidad simple, escalado, asintótico, ...

  La ley de Amdahl nos dice que el incremento de velocidad (simple) de un algoritmo paralelo está acotado superiormente por la inversa de la fracción serie del algoritmo, [79]. Más concretamente, el incremento de velocidad está acotado superiormente por el ratio entre la complejidad del algoritmo secuencial y la longitud del camino crítico del grafo del flujo de datos de dicho algoritmo. En la ley de Amdahl no se tiene en cuenta el tiempo de sobrecarga, por lo que en las fórmulas del subapartado anterior se considera que $T_O(z,p) = 0$. Además se supone que el tamaño del problema z es una constante. A pesar de lo pesimista de la conclusión, este incremento de

velocidad está sobreestimado, ya que la ley de Amdahl no tiene en cuenta detalles de la implementación paralela concretados en lo que se suele denominar tiempo de *sobrecarga* u *overhead*. Dicho de otra forma, la ley de Amdahl exige implícitamente que el número de operaciones a realizar, tanto por la versión secuencial como por la versión paralela, sea el mismo, lo cual suele ser algo difícil en la práctica.

La ley de Amdahl muestra conclusiones para un tamaño implícito fijo del problema. Gustafson, [80], argumenta que en la práctica el tamaño del problema no debería ser independiente del número de procesadores: el tamaño del problema debería *escalarse* con el número de procesadores. Añade que los tiempos de ejecución de la parte serie de un algoritmo no tienen por qué aumentar cuando se aumenta el tamaño del problema. En cualquier caso, sigue sin tener en cuenta explícitamente el tiempo de sobrecarga.

Aunque en la formulación de la referencia [80] no aparece explícitamente el tamaño del problema n, posteriormente veremos cualitativamente su relación con el número de procesadores.

Si $T_{\text{sec}_1}(z)$ es la cantidad de tiempo que el algoritmo secuencial emplea en ejecutar la parte *no paralelizable* del algoritmo, y $T_{\text{sec}_p}(z)$ la parte sí paralelizable, entonces un solo procesador secuencial requeriría la misma cantidad de tiempo para ejecutar la parte serie, y p veces mas tiempo que el sistema paralelo, para ejecutar la parte paralelizable: por lo que el denominado *incremento de velocidad escalado* sería $S = p + (1 - p)T_{\text{sec}_1}(z)$, suponiendo un tiempo de ejecución paralelo normalizado a la unidad.

Aunque las leyes de Amdahl y Gustafson-Barsis son aparentemente distintas, algebraicamente son equivalentes y sólo se diferencian en los grados de libertad que se atribuyen a los parámetros: la referencia que se escoge para determinar algebraicamente la fracción de tiempo de ejecución de la parte serie del algoritmo en la ley de Amdahl es el tiempo de ejecución del algoritmo secuencial y en la ley de

Gustafson-Barsis, el tiempo de ejecución del algoritmo paralelo. En el fondo, la diferencia importante estriba en que cuando aumentamos el número de procesadores, la ley de Amdahl supone que el tamaño del problema ha permanecido fijo, y la ley de Gustafson-Barsis supone que el tamaño del problema ha aumentado con tal de mantener el tiempo de ejecución del algoritmo paralelo fijo (1 unidad temporal en el caso normalizado).

En cualquier caso, si tenemos en cuenta el tamaño del problema z, el número de procesadores p y la normalización del tiempo de ejecución del algoritmo paralelo, tenemos que:

$$T_{\text{par}} = T_{\text{sec}_1} + \frac{T_{\text{sec}_p}}{p} = 1 \Longrightarrow T_{\text{par}}(z,p) = T_{\text{sec}_1}(z) + \frac{T_{\text{sec}_p}(z)}{p} = 1 \qquad (2.3)$$

lo cual implica que para un problema dado, si aumentamos el número de procesadores a $p' > p$, entonces el sumando $T_{\text{sec}_p}(z)/p$ disminuirá, por lo que habrá que aumentar la carga a $z' > z$ para que

$$T_{\text{par}}(z',p') = T_{\text{sec}_1}(z') + \frac{T_{\text{sec}_p}(z')}{p'} = 1$$

Es decir, el tamaño del problema se debe *escalar* con el número de procesadores. Dicho de otro modo, la relación entre n y p no es libre, si no que debe respetar cierta relación de *escala* según la expresión (2.3). Tienen en común que no tienen en cuenta el tiempo de sobrecarga del algoritmo paralelo, $T_O(z,p)$.

- Eficiencia, E: incremento de velocidad dividido por el número de procesadores.

$$E(z,p) \;=\; \frac{S(z,p)}{p}$$

Ya que el máximo incremento de velocidad teórico (obviando condiciones de super-

linealidad) es p, este valor nos proporciona una medida cuantitativa normalizada a la unidad de la bondad de la paralelización del algoritmo.

2.3.4. Consideraciones para redes heterogéneas

En una red heterogénea, el problema del equilibrado tiene una dimensión adicional al tener cada procesador una potencia de cálculo que puede ser diferente. Especial mención requiere cierto matiz sobre de la eficiencia para sistemas heterogéneos: si se pretende caracterizar la bondad de un algoritmo paralelo en términos relativos a partir de la eficiencia, esta definición no sería ecuánime al aplicarlo a un sistema heterogéneo. Si redefinimos la eficiencia como el cociente entre el incremento de velocidad y el máximo incremento de velocidad posible (que podemos denominar *eficiencia heterogénea o general*):

$$E_{ht}(z,p) = \frac{S(z,p)}{S_{\text{máx}}(z,p)}$$

entonces, este concepto es válido para analizar la bondad del algoritmo paralelo, tanto para sistemas heterogéneos como homogéneos, ya que, exceptuando situaciones de superlinealidad, el máximo incremento de velocidad para estos últimos sería de p.

El máximo incremento de velocidad se conseguirá con un perfecto equilibrado de la carga, tanto para redes homogéneas como heterogéneas cuando no haya sobrecarga de ningún tipo, es decir

$$T_{A,P_j}(z) = \cdots = T_{A,P_f}(z) = \cdots = T_{A,P_k}(z) = \frac{1}{S_{\text{máx}}(z,p)} T_{\text{sec}}(z)$$

$$W_{P_j}(z)t_{w_j} = \cdots = W_{P_f}(z)t_{w_f} = \cdots = W_{P_k}(z)t_{w_k} = \frac{1}{S_{\text{máx}}(z,p)} W_{\text{sec}}(z)t_{w_{\text{sec}}} \tag{2.4}$$

con

$$W_{\text{sec}}(z) = \sum_{j=0}^{p-1} W_{P_j}(z)$$

con $0 \leq j, \ldots, f, \ldots, k \leq p-1$ y siendo P_f el procesador más rápido de la red.

Definamos el concepto *velocidad relativa normalizada* para cada procesador α_j como una medida de velocidad relativa dentro de una red heterogénea normalizada a la unidad, de manera que la suma de todas las velocidades relativas de una red nos dé la unidad, y un procesador P_j que sea u veces más rápido que otro P_k tenga una velocidad relativa u veces superior ($\alpha_j = u\alpha_k$). Este parámetro lo podemos calcular como

$$\alpha_j = \frac{1}{\sum_{r=0}^{p-1} \frac{t_{w_j}}{t_{w_r}}}, \quad 0 \leq j \leq p-1 \tag{2.5}$$

Podemos comprobar fácilmente que $\sum_{j=0}^{p-1} \alpha_j = 1$, ya que

$$\begin{aligned}
\sum_{j=0}^{p-1} \alpha_j &= \sum_{j=0}^{p-1} \frac{1}{\sum_{r=0}^{p-1} \frac{t_{w_j}}{t_{w_r}}} \\
&= \sum_{j=0}^{p-1} \frac{1}{t_{w_j} \sum_{r=0}^{p-1} \frac{1}{t_{w_r}}} \\
&= \frac{1}{\sum_{r=0}^{p-1} \frac{1}{t_{w_r}}} \sum_{j=0}^{p-1} \frac{1}{t_{w_j}} \\
&= 1
\end{aligned}$$

y además $t_{w_j}\alpha_j = t_{w_k}\alpha_k$:

$$t_{w_j} \frac{1}{\sum_{r=0}^{p-1} \frac{t_{w_j}}{t_{w_r}}} = \frac{1}{\sum_{r=0}^{p-1} \frac{1}{t_{w_r}}} = t_{w_k} \frac{1}{\sum_{r=0}^{p-1} \frac{t_{w_k}}{t_{w_r}}}$$

entonces, si P_j es u veces más rápido que P_k:

$$\begin{aligned}
\frac{1}{t_{w_j}} &= u\frac{1}{t_{w_k}} \\
\frac{\alpha_j}{\alpha_k t_{w_k}} &= u\frac{1}{t_{w_k}} \\
\alpha_j &= u\alpha_k
\end{aligned}$$

Aunque este parámetro se haya definido como *velocidad relativa normalizada*, realmente es una parámetro adimensional al tener un carácter relativo.

De (2.4) podemos deducir el máximo incremento de velocidad, suponiendo, obviamente, que el algoritmo secuencial se ejecutará en el procesador más rápido ($t_{w_{\text{sec}}} = t_{w_f}$):

$$\begin{aligned} S_{\text{máx}}(z,p) &= \frac{W_{\text{sec}}(z)t_{w_f}}{W_{P_f}(z)t_{w_f}} \\ &= \frac{\sum_{j=0}^{p-1} W_{P_j}(z)}{W_{P_f}(z)} \\ &= \frac{\sum_{j=0}^{p-1} W_{P_f}(z)\frac{t_{w_f}}{t_{w_j}}}{W_{P_f}(z)} \\ &= \sum_{j=0}^{p-1} \frac{t_{w_f}}{t_{w_j}} \\ S_{\text{máx}}(z,p) &= \frac{1}{\alpha_f} \end{aligned} \tag{2.6}$$

Por lo que el dato necesario para obtener al eficiencia heterogénea es el inverso de la velocidad relativa normalizada del procesador más rápido, siendo éste el máximo incremento de velocidad alcanzable en una red heterogénea.

De esta forma, podemos reescribir (2.4) en función de las velocidades relativas normalizadas:

$$W_{P_j}(z)\frac{1}{\alpha_j} = \cdots = W_{P_f}(z)\frac{1}{\alpha_f} = \cdots = W_{P_k}(z)\frac{1}{\alpha_k} = W_{\text{sec}}(z) \tag{2.7}$$

es decir

$$W_{P_j}(z) = \alpha_j W_{\text{sec}}(z), \forall\, 0 \leq j \leq p-1 \tag{2.8}$$

ya que

$$\begin{aligned} t_{w_j} &= \frac{t_{w_f}\alpha_f}{\alpha_j} \\ t_{w_{\text{sec}}} &= t_{w_f} \\ \frac{1}{S_{\max}(z,p)} &= \alpha_f \end{aligned} \tag{2.9}$$

El máximo incremento de velocidad no dependería del tamaño del problema z, sino exclusivamente de la cantidad de procesadores y de sus características, por lo que $S_{\max}(z,p) \rightarrow S_{\max}(p)$.

2.3.5. Escalabilidad

En [81] se define la *escalabilidad* como el concepto que pretende describir cómo el tamaño del problema y el tamaño del sistema (número de procesadores) influyen en el rendimiento del propio sistema paralelo que resuelve el problema. La escalabilidad puede emplearse para predecir el comportamiento de un sistema frente a un problema, ambos de tamaño *grande* a partir del conocimiento del comportamiento en un sistema con un problema ambos de tamaño *pequeño*.

Aunque el concepto de escalabilidad se puede aplicar a un algoritmo paralelo, se suele hacer respecto a una arquitectura paralela dada; si la arquitectura representa realmente una clase genérica de arquitecturas, entonces si un algoritmo paralelo es escalable, lo es para dichas clases de arquitecturas. De la misma forma, podemos hablar de la escalabilidad de una arquitectura paralela respecto a un algoritmo o clase de algoritmos paralelos. La escalabilidad es una propiedad inherente de algoritmos, arquitecturas y sus combinaciones. La escalabilidad debería primeramente considerarse para las combinaciones arquitecturas-algoritmos. La escalabilidad del algoritmo o la de la arquitectura puede entonces ser definida a partir de la anterior [81].

Podemos interpretar la escalabilidad de un algoritmo paralelo como la capacidad de dicho algoritmo de hacer un uso efectivo de los recursos de un sistema con un número

mayor de procesadores dentro de una arquitectura paralela dada. No existe una definición uniforme en cuanto al concepto de *uso efectivo*, ya que ello depende de los parámetros que se desean que permanezcan constantes cuando se escala el sistema. Como parámetros de interés podemos citar: el número de procesadores, la carga computacional, el tiempo de ejecución, la velocidad de ejecución, la memoria del sistema, la eficiencia, el incremento de velocidad,... En [84, 85] se muestran algunas métricas para este concepto.

Suelen plantearse dos modelos genéricos de escalado del sistema (aumento del número de procesadores) [84]:

- Modelo de escalado por tiempo crítico: el objetivo, al aumentar el número de procesadores, es disminuir el tiempo de ejecución del algoritmo paralelo al haber incrementado el número de procesadores, manteniendo el tamaño de problema. El grado o métrica de escalabilidad vendrá dado por la necesaria variación del resto de parámetros como el incremento de velocidad o la eficiencia.

- Modelo de escalado por precisión crítica: el objetivo al aumentar el número de procesadores es poder aumentar el tamaño del problema. Este aumento puede realizarse manteniendo cierto parámetro constante, lo que da lugar a los siguientes subtipos de escalado:

  - Por isoeficiencia: se mantiene constante la eficiencia [86], [71]
  - Por isotiempo: se mantiene constante el tiempo de ejecución [80], [83]
  - Por isovelocidad: se mantiene constante la velocidad (carga computacional o tamaño del problema, por unidad de tiempo) [81]
  - Por isomemoria: se mantiene constante la memoria [84]

  El grado o métrica de escalabilidad vendrá dado por la degradación del o los parámetros que no quedan prefijados.

Funciones iso-paramétricas

Lo que resulta de interés en estos tipos de escalado es la forma en la que puede o debe crecer el tamaño del problema con tal de conseguir que el parámetro en cuestión permanezca constante. Ello da lugar a las respectivas funciones iso-paramétricas que relacionan el incremento del número de procesadores con el incremento del tamaño del problema o de la carga computacional manteniendo cierto parámetro constante.

Nuestro interés se centra en la función de isoeficiencia, la cual relaciona el tamaño del problema a resolver con el número de procesadores necesario para mantener cierta eficiencia de referencia.

En [71] se emplea el modelo de tiempo de ejecución del algoritmo paralelo de la ecuación (2.2). La eficiencia podemos reescribirla como

$$E = \frac{S}{p} = \frac{T_{\text{sec}}(z)}{T_{\text{sec}}(z) + T_O(z,p)} = \frac{1}{1 + \frac{T_O(z,p)}{T_{\text{sec}}(z)}}$$

Cuando aumentamos el número de procesadores a $p' > p$, manteniendo constante el tamaño del problema, la eficiencia suele disminuir, debido a que el tiempo de *overhead*, $T_O(z,p)$, aumenta con el número de procesadores. Si se desea mantener una eficiencia constante y aumentar el número de procesadores, de la expresión anterior se deduce que habrá que aumentar $T_{\text{sec}}(z)$ para *contrarrestar* el incremento en $T_O(z,p')$, y esto se consigue, a su vez, aumentando el tamaño del problema en cuestión de z a cierto z'.

Si se desea obtener una eficiencia igual a la de referencia, E_{ref}, la relación que debe existir entre los términos implicados $T_O(z',p')$ y $T_{\text{sec}}(z')$ debe ser:

$$\begin{aligned}
\frac{1}{1+\frac{T_O(z',p')}{T_{\text{sec}}(z')}} &= E_{\text{ref}} = \frac{1}{1+\frac{T_O(z,p)}{T_{\text{sec}}(z)}} \\
1+\frac{T_O(z',p')}{T_{\text{sec}}(z')} &= \frac{1}{E_{\text{ref}}} \\
\frac{T_O(z',p')}{T_{\text{sec}}(z')} &= \frac{1}{E_{\text{ref}}} - 1 = C \\
T_O(z',p') &= CT_{\text{sec}}(z')
\end{aligned} \tag{2.10}$$

Puede observarse que al intentar mantener altas eficiencias, la constante C puede tomar un valor relativamente pequeño, es decir, el tiempo de sobrecarga deberá ser una fracción relativamente pequeña del tiempo del algoritmo secuencial si se desea mantener la eficiencia de referencia.

En [71] se muestra una sencilla técnica para expresar la función de isoeficiencia, relacionando en términos de orden de magnitud el incremento necesario de la carga respecto a un incremento en el número de procesadores. Cuando menor sea la relación más escalable será el sistema paralelo.

2.4 Metodología en la experimentación

Concluimos así la exposición sobre los conceptos que estimamos más importantes y que están más íntimamente relacionados con este trabajo. Estos han sido las arquitecturas paralelas, la programación en paralelo y el análisis de las prestaciones de los algoritmos paralelos. Seguidamente procedemos a comentar brevemente el contexto en el que se pretende evaluar a los algoritmos. También analizaremos la metodología empleada para llevar a cabo la validación y calificación de los algoritmos paralelos implementados.

El marco en el que se van a diseñar los algoritmos, tanto secuenciales como paralelos, pretende corresponderse, en la medida de lo posible, a una situación real en el que el subsistema de cálculo debe realizar ciertos cómputos a partir de observaciones que proceden de cierto subsistema de adquisición. A partir de la solución, se actua en consecuencia

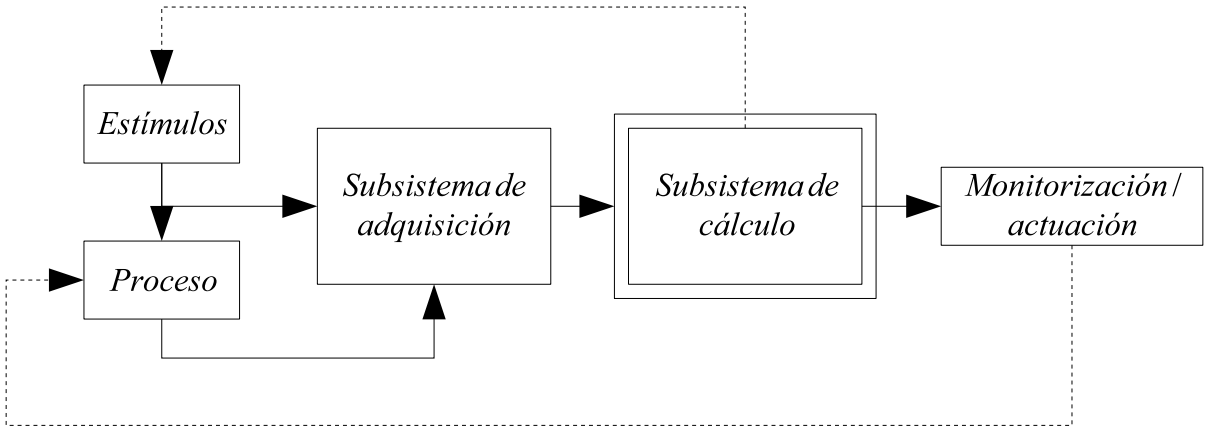

Figura 2.5: Marco de aplicación

(monitorización de los resultados, actuación sobre el proceso —bucle de regulación—, etc.) sobre el proceso que se pretende estudiar. Nuestro cometido se centra en intentar conseguir que el subsistema de cálculo proporcione los resultados lo más rápidamente posible sin perder la perspectiva de la escalabilidad de la solución. La figura 2.5 muestra gráficamente estas ideas.

Primeramente diseñaremos el algoritmo secuencial estableciendo un modelo de tiempo de ejecución. A continuación validaremos dicho algoritmo comprobando los resultados con los de rutinas de la biblioteca numérica `LAPACK`. Posteriormente, empleando la metodología de diseño de algoritmos paralelos expuesta en la subsección 2.2.3, procederemos al diseño y posterior implementación del algoritmo paralelo para las versiones que sean oportunas; de nuevo estableceremos un modelo de tiempo de ejecución, así como los criterios de equilibrado de carga, volviendo a comprobar la corrección de estas implementaciones.

Posteriormente procederemos a la fase de toma de tiempos. Para ello, cada algoritmo embebido en una rutina, será ejecutado diez veces, promediando los tiempos obtenidos. Adicionalmente, tomaremos tiempos en aquellas partes del algoritmo que estimemos oportunas para obtener características particulares (tiempos de ejecución de ciertas partes, observación de similitud o discrepancia con modelos, etc.). Los resultados, escritos en ficheros, serán post-procesados para extraer resultados gráficos y conclusiones.

Los resultados deseados serán, para los algoritmos secuenciales: tiempos de ejecución

secuencial y comparación con el modelo teórico de tiempo de ejecución; para los algoritmos paralelos, observación del equilibrado de carga, deducción del tiempo de sobrecarga por paralelización, tiempos de ejecución paralela, comparación con el modelo teórico, la eficiencia y un estudio teórico de la escalabilidad.

Con todo ello pretendemos valorar cualitativa y cuantitativamente la bondad de las paralelizaciones, intentando justificar los resultados obtenidos y por último extraer conclusiones y propuestas sobre la posible mejora de las mismas.

3

El problema de mínimos cuadrados

Existen numerosos problemas en los que se pretende, de alguna forma, realizar una estimación de una incógnita a partir de ciertas observaciones. En ocasiones, la estimación se lleva a cabo con el objetivo de intentar que el error en la misma sea el menor posible atendiendo a ciertos criterios. A continuación presentamos el problema de mínimos cuadrados (LS) y algunas de sus múltiples concepciones que encajan con el trabajo a realizar.

3.1 Notación

Las variables escritas en negrita denotarán un vector o una matriz, en caso contrario, la variable en cuestión se considerará escalar. Un superíndice con un asterisco/T ($^*/^T$) indicará el/la transpuesto-conjugado/transposición de una matriz; un superíndice con una *daga* ($\mathbf{A}^\dagger$) denotará la pseudoinversa de la matriz; $\mathrm{E}\{\cdot\}$ o una línea sobre una variable denotará el valor esperado de dicha variable; con $\mathrm{cov}(\cdot)$ indicaremos la matriz de covarianza y con $||\cdot||$ la norma euclídea de un vector. Cuando una matriz $\mathbf{A}$ sea simétrica y definida positiva podremos expresarla como $\mathbf{A} = (\mathbf{A}^{1/2})(\mathbf{A}^{1/2})^* = \mathbf{A}^{1/2}\mathbf{A}^{*/2}$.

En descripciones algorítmicas o pseudocódigos utilizaremos una notación similar a la

de Matlab para indicar submatrices o rangos de índices en las oportunas dimensiones.

3.2 Problemas LS deterministas

3.2.1. Solución basada en las ecuaciones normales

Sean $\mathbf{A} \in \mathbb{C}^{m\times n}$, $\mathbf{x} \in \mathbb{C}^{n\times 1}$, $\mathbf{y}, \mathbf{v} \in \mathbb{C}^m$, con los que establecemos la siguiente igualdad:

$$\mathbf{A}\mathbf{x} + \mathbf{v} = \mathbf{y} \tag{3.1}$$

donde $\mathbf{v}$ se interpreta como una perturbación de la observación $\mathbf{y}$.

En ocasiones, debido al desconocimiento determinista de $\mathbf{v}$, se emplea la notación

$$\mathbf{A}\mathbf{x} \cong \mathbf{y} \tag{3.2}$$

denotando que el sistema de ecuaciones es *inconsistente*, ya que $\mathbf{y}$ no es una combinación lineal de las columnas de $\mathbf{A}$ (a no ser que $\mathbf{v}$ lo fuera).

La estimación de $\mathbf{x}$ a partir del conocimiento exclusivamente de la observación $\mathbf{y}$ y de la matriz $\mathbf{A}$, podemos llevarla a cabo aplicando el criterio de intentar hacer mínima la norma del denominado *vector residuo*, $\mathbf{y} - \mathbf{A}\mathbf{x}$. Es decir, si a dicha estimación la denotamos por $\mathbf{x}_{\mathrm{LS}}$, entonces:

$$||\mathbf{y} - \mathbf{A}\mathbf{x}||^2 \geq ||\mathbf{y} - \mathbf{A}\mathbf{x}_{\mathrm{LS}}||^2 \tag{3.3}$$

Si definimos como *función de coste* $\mathrm{J}(\mathbf{x})$

$$\begin{aligned} \mathrm{J}(\mathbf{x}) \triangleq ||\mathbf{y} - \mathbf{A}\mathbf{x}||^2 &= (\mathbf{y} - \mathbf{A}\mathbf{x})^*(\mathbf{y} - \mathbf{A}\mathbf{x}) \\ &= \mathbf{x}^*\mathbf{A}^*\mathbf{A}\mathbf{x} - \mathbf{x}^*\mathbf{A}^*\mathbf{y} - \mathbf{y}^*\mathbf{A}\mathbf{x} + \mathbf{y}^*\mathbf{y} \end{aligned}$$

entonces la minimización de la norma del vector residuo equivale a la minimización de la función de coste $\mathrm{J}(\mathbf{x})$, que podemos calcular como:

$$0 = \left.\frac{\partial \mathrm{J}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_{\mathrm{LS}}} = \mathbf{x}_{\mathrm{LS}}^{*}\mathbf{A}^{*}\mathbf{A} - \mathbf{y}^{*}\mathbf{A} \tag{3.4}$$

De modo que cualquier vector $\mathbf{x}_{\mathrm{LS}}$ que haga que $\mathbf{x}_{\mathrm{LS}}^{*}\mathbf{A}^{*}\mathbf{A} = \mathbf{y}^{*}\mathbf{A}$ será solución LS.

Si $\mathbf{A}$ es de rango completo, entonces $\mathbf{A}^{*}\mathbf{A}$ es no singular, por lo que la solución LS será única:

$$\mathbf{x}_{\mathrm{LS}} = \left(\mathbf{A}^{*}\mathbf{A}\right)^{-1}\mathbf{A}^{*}\mathbf{y}$$

Si $\mathbf{A}$ no es de rango completo, entonces existe más de una solución $\mathbf{x}_{\mathrm{LS}}$, aunque sólo una de ellas posee norma euclídea mínima. Esta puede conseguirse mediante la *factorización QR con pivotamiento de columnas* de la matriz $\mathbf{A}$, o bien mediante la pseudo-inversa de Moore-Penrose[1], [87].

En cualquier caso, cabe destacar la condición de ortogonalidad del vector residuo $\mathbf{y} - \mathbf{A}\mathbf{x}_{\mathrm{LS}}$ con el espacio generado por las columnas de $\mathbf{A}$, ya que, de (3.4):

$$\begin{aligned} \mathbf{0} &= \mathbf{x}_{\mathrm{LS}}^{*}\mathbf{A}^{*}\mathbf{A} - \mathbf{y}^{*}\mathbf{A} \\ \mathbf{A}^{*}\mathbf{A}\mathbf{x}_{\mathrm{LS}} &= \mathbf{A}^{*}\mathbf{y} \\ \mathbf{A}^{*}\left(\mathbf{y} - \mathbf{A}\mathbf{x}_{\mathrm{LS}}\right) &= \mathbf{0} \end{aligned}$$

El valor de la función de coste en $\mathbf{x}_{\mathrm{LS}}$ coincide con el cuadrado de la norma euclídea del vector residuo, luego:

[1] Si $\mathbf{A} = \mathbf{U}\begin{pmatrix} \mathbf{\Sigma} \\ \mathbf{0} \end{pmatrix}\mathbf{V}^{*}$ es la descomposición en valores singulares de $\mathbf{A}$, entonces la pseudo-inversa de Moore-Penrose puede obtenerse como $\mathbf{A}^{\dagger} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^{*}$, resultando entonces la estimación de $\mathbf{x}_{\mathrm{LS}}$ como $\mathbf{x} = \mathbf{A}^{\dagger}\mathbf{y}$

$$
\begin{aligned}
\mathrm{J}(\mathbf{x}_{\mathrm{LS}}) = ||\mathbf{y} - \mathbf{A}\mathbf{x}_{\mathrm{LS}}||^2 \qquad & = (\mathbf{y} - \mathbf{A}\mathbf{x}_{\mathrm{LS}})^*(\mathbf{y} - \mathbf{A}\mathbf{x}_{\mathrm{LS}}) \\
= \mathbf{y}^*(\mathbf{y} - \mathbf{A}\mathbf{x}_{\mathrm{LS}}) - \mathbf{x}_{\mathrm{LS}}^*\mathbf{A}^*(\mathbf{y} - \mathbf{A}\mathbf{x}_{\mathrm{LS}}) \qquad & = \mathbf{y}^*(\mathbf{y} - \mathbf{A}\mathbf{x}_{\mathrm{LS}}) \\
& \quad - \mathbf{x}_{\mathrm{LS}}^*(\mathbf{A}^*\mathbf{y} - \mathbf{A}^*\mathbf{A}\mathbf{x}_{\mathrm{LS}}) \\
= \mathbf{y}^*(\mathbf{y} - \mathbf{A}\mathbf{x}_{\mathrm{LS}}) \qquad & = ||\mathbf{y}||^2 - \mathbf{y}^*\mathbf{A}\mathbf{x}_{\mathrm{LS}} \\
= ||\mathbf{y}||^2 - (\mathbf{A}^*\mathbf{y})^*\,\mathbf{x}_{\mathrm{LS}} \qquad & = ||\mathbf{y}||^2 - (\mathbf{A}^*\mathbf{A}\mathbf{x}_{\mathrm{LS}})^*\,\mathbf{x}_{\mathrm{LS}} \\
= ||\mathbf{y}||^2 - \mathbf{x}_{\mathrm{LS}}^*\mathbf{A}^*\mathbf{A}\mathbf{x}_{\mathrm{LS}} \qquad & = ||\mathbf{y}||^2 - ||\mathbf{A}\mathbf{x}_{\mathrm{LS}}||^2
\end{aligned}
$$

En la figura 3.1 se muestra una interpretación geométrica de los resultados, donde $\mathbf{x}_0$ es la solución correcta y $\mathbf{x}_{\mathrm{LS}}$, la de mínimos cuadrados.

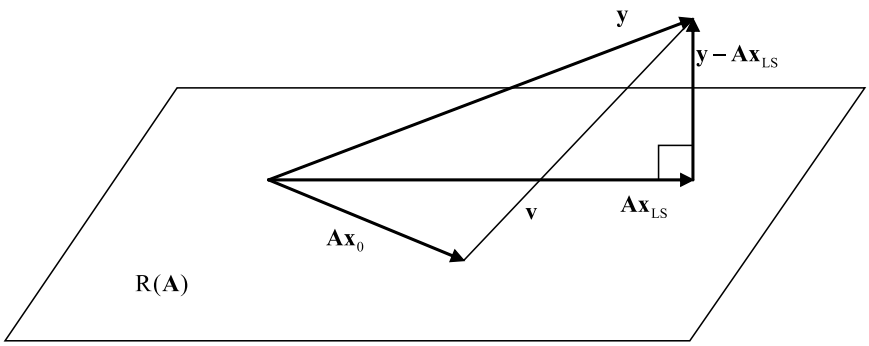

Figura 3.1: Solución LS.

3.2.2. El problema LS utilizando la factorización QR

Sea el sistema[2]:

$$\mathbf{A}\mathbf{x} + \mathbf{v} = \mathbf{y}$$

donde $\mathbf{A} \in \mathbb{R}^{m \times n}$ es una matriz de rango completo con $m \geq n$.

La solución al problema LS, $\mathbf{x}_{\mathrm{LS}}$, obtiene

[2]Supondremos por simplicidad que los datos del problema pertenecen al cuerpo de los reales.

$$\min_{\mathbf{x}} ||\mathbf{A}\mathbf{x} - \mathbf{y}||_2^2$$

Sea $\mathbf{A} = \mathbf{Q}\mathbf{R}$ la factorización QR de $\mathbf{A}$, entonces $\mathbf{Q} \in \mathbb{R}^{m\times m}$ es ortogonal y $\mathbf{R} \in \mathbb{R}^{m\times n}$ es una matriz triangular superior. Podemos reescribir $\mathbf{Q}$ y $\mathbf{R}$ como

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_a & \mathbf{Q}_b \end{pmatrix}, \quad \mathbf{R} = \begin{pmatrix} \mathbf{R}_a \\ \mathbf{0} \end{pmatrix}$$

donde $\mathbf{Q}_a \in \mathbb{R}^{m\times n}$, $\mathbf{Q}_b \in \mathbb{R}^{m\times(m-n)}$ y $\mathbf{R}_a \in \mathbb{R}^{n\times n}$. Así pues:

$$\mathbf{A} = \mathbf{Q}\mathbf{R} = \mathbf{Q}_a\mathbf{R}_a$$

pudiendo resolver el problema LS como, [87]

$$\mathbf{x}_{\mathrm{LS}} = \mathbf{R}_a^{-1}\mathbf{Q}_a^T\mathbf{y} \tag{3.5}$$

siendo la norma euclídea del vector residuo

$$\min_{\mathbf{x}} ||\mathbf{A}\mathbf{x} - \mathbf{y}||_2 = ||\mathbf{Q}_b^T\mathbf{y}||$$

3.2.3. Análisis de perturbación en la solución del problema LS

Las siguientes cotas de los límites de perturbación de la solución del problema LS están obtenidas de la referencia [88].

Dado el sistema de ecuaciones (3.2), supongamos que $\mathbf{x}_0$ es la solución correcta y $\mathbf{x}_{\mathrm{LS}}$ es la solución LS de dicho sistema de ecuaciones. Consideremos, adicionalmente, que $(\mathbf{x}_{\mathrm{LS}} + d\mathbf{x}_{\mathrm{LS}})$ es la solución LS del siguiente sistema perturbado:

$$\tilde{\mathbf{A}}(\mathbf{x}_{\mathrm{LS}} + d\mathbf{x}_{\mathrm{LS}}) \equiv (\mathbf{A} + \mathbf{E})(\mathbf{x}_{\mathrm{LS}} + d\mathbf{x}_{\mathrm{LS}}) \cong \mathbf{y} + d\mathbf{y}$$

con las siguientes condiciones:

$$||\mathbf{E}|| \cdot ||\mathbf{A}^\dagger|| < 1, \text{ y Rango}(\tilde{\mathbf{A}}) \leq \text{Rango}(\mathbf{A})$$

En tal caso, puede demostrarse, [88] que si definimos:

$$\alpha = \frac{||\mathbf{E}||}{||\mathbf{A}||} \qquad \beta = \frac{||d\mathbf{y}||}{||\mathbf{y}||} \qquad \gamma = \frac{||\mathbf{y}||}{||\mathbf{A}|| \cdot ||\mathbf{x}_{\text{LS}}||} \leq \frac{||\mathbf{y}||}{||\mathbf{A}\hat{\mathbf{x}}||}$$
$$\rho = \frac{||\mathbf{v}||}{||\mathbf{A}|| \cdot ||\mathbf{v}||} \leq \frac{||\mathbf{v}||}{||\mathbf{A}\mathbf{v}||} \qquad \kappa = ||\mathbf{A}|| \cdot ||\mathbf{A}^\dagger|| \qquad \hat{\kappa} = \frac{\kappa}{1 - \kappa\alpha}$$

entonces

- $\text{Rango}(\tilde{\mathbf{A}}) = \text{Rango}(\mathbf{A})$

- El error relativo de $\mathbf{x}_{\text{LS}}$ es:

$$\frac{||d\mathbf{x}_{\text{LS}}||}{||\mathbf{x}_{\text{LS}}||} \leq \frac{||\mathbf{A}|| \cdot ||\mathbf{A}^\dagger||}{1 - ||\mathbf{E}|| \cdot ||\mathbf{A}^\dagger||} \cdot \cdot \left[\left(2 + ||\mathbf{A}|| \cdot ||\mathbf{A}^\dagger|| \frac{||\mathbf{v}||}{||\mathbf{A}|| \cdot ||\mathbf{x}_{\text{LS}}||} \right) \frac{||\mathbf{E}||}{||\mathbf{A}||} + \frac{||\mathbf{y}||}{||\mathbf{A}|| \cdot ||\mathbf{x}_{\text{LS}}||} \cdot \frac{||d\mathbf{y}||}{||\mathbf{y}||} \right]$$

La figura 3.2 muestra gráficamente el resultado, para un caso particular:

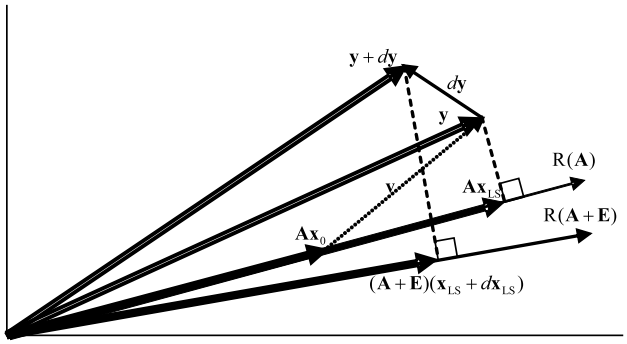

Figura 3.2: Perturbación en el problema LS.

3.2.4. Problemas LS regularizados

Podemos rediseñar el criterio de optimización de la expresion (3.3) de la siguiente forma:

$$(\mathbf{x}-\mathbf{x}_0)^*\mathbf{\Pi}_0^{-1}(\mathbf{x}-\mathbf{x}_0)+||\mathbf{y}-\mathbf{A}\mathbf{x}||^2 \geq (\mathbf{x}_{\mathrm{LS}}-\mathbf{x}_0)^*\mathbf{\Pi}_0^{-1}(\mathbf{x}_{\mathrm{LS}}-\mathbf{x}_0)+||\mathbf{y}-\mathbf{A}\mathbf{x}_{\mathrm{LS}}||^2 \quad (3.6)$$

$\forall\ \mathbf{x} \in \mathbb{C}^{n\times 1}$, donde $\mathbf{x}_0$ es una posible solución para $\mathbf{x}_{\mathrm{LS}}$, y cuya incertidumbre se indica implícitamente en la matriz de ponderación $\mathbf{\Pi}_0$, que supondremos simétrica y definida positiva por lo que $\mathbf{\Pi}_0^{-1} = \mathbf{\Pi}_0^{-*/2}\mathbf{\Pi}_0^{-1/2}$: si, por ejemplo, $\mathbf{\Pi}_0 = \epsilon\mathbf{I}$, con ϵ un valor muy pequeño, estaríamos indicando una alta certidumbre de que $\mathbf{x}_0$ está muy cerca de la solución verdadera, es decir, *daríamos más peso* a que la solución estuviera cerca de $\mathbf{x}_0$ que al valor del vector residuo $\mathbf{y}-\mathbf{A}\mathbf{x}$. Por el contrario, si $\mathbf{\Pi}_0 = \infty\mathbf{I}$, entonces la ecuación (3.6) coincide con (3.3).

Para determinar el mínimo de (3.6), realizaremos el siguiente cambio de variables: $\mathbf{x}' = \mathbf{x}-\mathbf{x}_0$ e $\mathbf{y}' = \mathbf{y}-\mathbf{A}\mathbf{x}_0$:

$$\min_{\mathbf{x}'}\left\{\mathbf{x}'^*\mathbf{\Pi}_0\mathbf{x}'+||\mathbf{y}'-\mathbf{A}\mathbf{x}'||^2\right\} = \min_{\mathbf{x}'}\left\{\left|\left|\begin{pmatrix}\mathbf{0}\\ \mathbf{y}'\end{pmatrix}-\begin{pmatrix}\mathbf{\Pi}_0^{-1/2}\\ \mathbf{A}\end{pmatrix}\mathbf{x}'\right|\right|^2\right\}$$

Del mismo modo que anteriormente, la condición de ortogonalidad obliga a que:

$$\begin{aligned}\begin{pmatrix}\mathbf{\Pi}_0^{-1/2}\\ \mathbf{A}\end{pmatrix}^*\left[\begin{pmatrix}\mathbf{0}\\ \mathbf{y}'\end{pmatrix}-\begin{pmatrix}\mathbf{\Pi}_0^{-1/2}\\ \mathbf{A}\end{pmatrix}\mathbf{x}'_{\mathrm{LS}}\right] &= \mathbf{0}\\ \mathbf{A}^*\mathbf{y}'-\left(\mathbf{\Pi}_0^{-*/2}\mathbf{\Pi}_0^{-1/2}+\mathbf{A}^*\mathbf{A}\right)\mathbf{x}'_{\mathrm{LS}} &= \mathbf{0}\end{aligned}$$

siendo la solución buscada:

$$\mathbf{x}'_{\mathrm{LS}} = \left(\mathbf{\Pi}_0^{-1} + \mathbf{A}^*\mathbf{A}\right)^{-1}\mathbf{A}^*\mathbf{y}'$$

y recordando que $\mathbf{x}' = \mathbf{x} - \mathbf{x}_0$ e $\mathbf{y}' = \mathbf{y} - \mathbf{A}\mathbf{x}_0$, entonces

$$\mathbf{x}_{\mathrm{LS}} = \mathbf{x}_0 + \left(\mathbf{\Pi}_0^{-1} + \mathbf{A}^*\mathbf{A}\right)^{-1}\mathbf{A}^*(\mathbf{y} - \mathbf{A}\mathbf{x}_0) \tag{3.7}$$

La presencia de la matriz $\mathbf{\Pi}_0$ supone cierta ventaja ya que, con el valor oportuno, permite relajar la condición de que $\mathbf{A}$ deba ser de rango completo para que $\mathbf{A}^*\mathbf{A}$ sea invertible.

3.3 Problemas LS estocásticos

Supongamos que $\mathbf{x} \in \mathbb{C}^n \times 1$ denota un *vector aleatorio*, en principio de media nula, que deseamos *estimar*, y existe la conjetura de que está relacionado linealmente con cierto vector aleatorio $\mathbf{y} \in \mathbb{C}^{m\times 1}$, también en principio con media nula, que podemos *observar*. Supongamos también que de estos dos vectores sólo se conocen los estadísticos de primer y segundo orden (medias, varianzas y covarianzas).

Partiendo de la conjetura de la relación lineal entre $\mathbf{x}$ e $\mathbf{y}$, la estimación de $\mathbf{x}$, denotada como $\hat{\mathbf{x}}$, podemos escribirla como

$$\hat{\mathbf{x}} = \mathbf{K_0}\mathbf{y}$$

donde $\mathbf{K_0} \in \mathbb{C}^{n\times m}$ es una matriz determinista incógnita que deberíamos calcular cumpliendo cierto criterio de optimización: concretamente que el error cometido en la estimación de $\mathbf{x}$ sea el mínimo posible, es decir, debemos encontrar una matriz $\mathbf{K_0}$, tal que $\forall\ \mathbf{K} \in \mathbb{C}^{n\times m}$ y $\forall\ \mathbf{a} \in \mathbb{C}^{1\times n}$

$$\mathbf{a}\mathrm{P}(\mathbf{K})\mathbf{a}^* \geq \mathbf{a}\mathrm{P}(\mathbf{K_0})\mathbf{a}^*$$

donde

$$\mathrm{P}(\mathbf{K}) \triangleq \mathrm{E}\left\{(\mathbf{x}-\mathbf{K}\mathbf{y})(\mathbf{x}-\mathbf{K}\mathbf{y})^*\right\}$$

y $\mathrm{E}\{\cdot\}$ denota la esperanza matemática. De esta forma se asegura adicionalmente que, no sólo se minimiza el error en la estimación de $\mathbf{x}$, sino que además, se minimiza el error de cualquier combinación lineal arbitraria de las componentes de $\mathbf{x}$, con lo que podríamos variar el peso de cada componente en dicha estimación, [89]

La figura 3.3 muestra gráficamente el proceso de estimación.

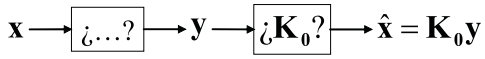

Figura 3.3: Estimación de $\mathbf{x}$ a partir de la observación de $\mathbf{y}$.

3.3.1. Estimadores por mínimos cuadrados medios lineales óptimos

Dados $\mathbf{x}$ e $\mathbf{y}$ definidos tal y como se indicó en el apartado anterior, el estimador por mínimos cuadrados medios lineales (*l.l.m.s.e. —linear least mean squares estimator—*) viene dado por cualquier solución $\mathbf{K_0}$ de las denominadas *ecuaciones normales*

$$\mathbf{K_0}\mathbf{R_y} = \mathbf{R_{xy}}$$

donde

- $\mathbf{R_y} = \mathrm{E}\{\mathbf{y}\mathbf{y}^*\}$, es la matriz de autocorrelación de $\mathbf{y}$
- $\mathbf{R_{xy}} = \mathrm{E}\{\mathbf{x}\mathbf{y}^*\} = \mathbf{R_{yx}}^*$, es la matriz de correlación cruzada de $\mathbf{x}$ e $\mathbf{y}$

siendo $\mathbf{K_0}$ solución del problema de optimización citado si y sólo si $\forall\ \mathbf{a} \in \mathbb{C}^{1\times n}$, $\mathbf{a}\mathbf{K_0}$ es un mínimo de $\mathbf{a}\mathrm{P}(\mathbf{K})\mathbf{a}^*$, donde

$$\begin{aligned}
\mathbf{a}\mathrm{P}(\mathbf{K})\mathbf{a}^* &= \mathbf{a}\left[\mathrm{E}\left\{(\mathbf{x}-\hat{\mathbf{x}})(\mathbf{x}-\hat{\mathbf{x}})^*\right\}\right]\mathbf{a}^* \\
&= \mathbf{a}\left[\mathrm{E}\left\{(\mathbf{x}-\mathbf{K}\mathbf{y})(\mathbf{x}-\mathbf{K}\mathbf{y})^*\right\}\right]\mathbf{a}^* \\
&= \mathbf{a}\left[\mathrm{E}\left\{\mathbf{x}\mathbf{x}^*\right\}-\mathrm{E}\left\{\mathbf{x}\mathbf{y}^*\mathbf{K}^*\right\}-\mathrm{E}\left\{\mathbf{K}\mathbf{y}\mathbf{x}^*\right\}+\mathrm{E}\left\{\mathbf{K}\mathbf{y}\mathbf{y}^*\mathbf{K}^*\right\}\right]\mathbf{a}^* \\
&= \mathbf{a}\left[\mathbf{R}_{\mathbf{x}}-\mathbf{R}_{\mathbf{xy}}\mathbf{K}^*-\mathbf{K}\mathbf{R}_{\mathbf{yx}}+\mathbf{K}\mathbf{R}_{\mathbf{y}}\mathbf{K}^*\right]\mathbf{a}^* \\
&= \mathbf{a}\mathbf{R}_{\mathbf{x}}\mathbf{a}^*-\mathbf{a}\mathbf{R}_{\mathbf{xy}}\mathbf{K}^*\mathbf{a}^*-\mathbf{a}\mathbf{K}\mathbf{R}_{\mathbf{yx}}\mathbf{a}^*+\mathbf{a}\mathbf{K}\mathbf{R}_{\mathbf{y}}\mathbf{K}^*\mathbf{a}^*
\end{aligned}$$

$\mathbf{a}\mathrm{P}(\mathbf{K})\mathbf{a}^*$ es una función escalar del vector complejo $\mathbf{aK}$, por lo tanto, para encontrar el mínimo respecto a $\mathbf{aK}$, derivaremos respecto de este vector e igualaremos a cero

$$\frac{\partial \mathbf{a}\mathrm{P}(\mathbf{K})\mathbf{a}^*}{\partial \mathbf{a}\mathbf{K}} = -\mathbf{R}_{\mathbf{yx}}\mathbf{a}^*+\mathbf{R}_{\mathbf{y}}\mathbf{K}^*\mathbf{a}^* = 0$$

entonces

$$\mathbf{R}_{\mathbf{yx}}\mathbf{a}^* = \mathbf{R}_{\mathbf{y}}\mathbf{K}^*\mathbf{a}^*$$

Por lo tanto, puede comprobarse que si $\mathbf{R}_{\mathbf{yx}} = \mathbf{R}_{\mathbf{y}}\mathbf{K}^*$, entonces $\forall\ \mathbf{a} \in \mathbb{C}^{1\times n}$, estamos en un mínimo. Dicho $\mathbf{K}$ lo denominaremos $\mathbf{K_0}$, cumpliéndose que:

$$\mathbf{R}_{\mathbf{yx}} = \mathbf{R}_{\mathbf{y}}{\mathbf{K_0}}^*,\ \ \mathbf{R}_{\mathbf{xy}} = \mathbf{K_0}\mathbf{R}_{\mathbf{y}},\ \ \mathbf{K_0} = \mathbf{R}_{\mathbf{xy}}{\mathbf{R}_{\mathbf{y}}}^{-1}$$

si existe ${\mathbf{R}_{\mathbf{y}}}^{-1}$, ya que

$$\begin{aligned}
\mathbf{R_{yx}} &= \mathrm{E}\{\mathbf{yx}^*\} \\
\mathbf{R_{yx}}^* &= \mathrm{E}\{\mathbf{yx}^*\}^* = \mathrm{E}\{\mathbf{xy}^*\} = \mathbf{R_{xy}} \\
\mathbf{R_y} &= \mathrm{E}\{\mathbf{yy}^*\} = \mathrm{E}\{\mathbf{yy}^*\}^* = \mathbf{R_y}^*
\end{aligned}$$

Por lo tanto, la estimación buscada será:

$$\begin{aligned}
\hat{\mathbf{x}} &= \mathbf{K}_0\mathbf{y} \\
&= \mathbf{R_{xy}}\mathbf{R_y}^{-1}\mathbf{y}
\end{aligned} \tag{3.8}$$

y el error cuadrático medio mínimo (*m.m.s.e.: minimum-mean-square-error*) en la estimación:

$$\begin{aligned}
m.m.s.e. \triangleq \mathrm{P}(\mathbf{K_0}) &= \mathrm{E}\{(\mathbf{x}-\hat{\mathbf{x}})(\mathbf{x}-\hat{\mathbf{x}})^*\} \\
&= \mathrm{E}\{(\mathbf{x}-\mathbf{K_0y})(\mathbf{x}-\mathbf{K_0y})^*\} \\
&= \mathrm{E}\{\mathbf{xx}^*\} - \mathrm{E}\{\mathbf{xy}^*\mathbf{K_0}^*\} - \mathrm{E}\{\mathbf{K_0yx}^*\} + \mathrm{E}\{\mathbf{K_0yy}^*\mathbf{K_0}^*\} \\
&= \mathbf{R_x} - \mathbf{R_{xy}}\mathbf{K_0}^* - \mathbf{K_0}\mathbf{R_{yx}} + \mathbf{K_0}\mathbf{R_y}\mathbf{K_0}^* \\
&= \mathbf{R_x} - \mathbf{R_{xy}}\mathbf{K_0}^* - \mathbf{K_0}\mathbf{R_y}\mathbf{K_0}^* + \mathbf{K_0}\mathbf{R_y}\mathbf{K_0}^* \\
&= \mathbf{R_x} - \mathbf{R_{xy}}\mathbf{K_0}^* \\
&= \mathbf{R_x} - \mathbf{R_{xy}}\mathbf{R_y}^{-1}\mathbf{R_{xy}}^* \\
&= \mathbf{R_x} - \mathbf{R_{xy}}\mathbf{R_y}^{-1}\mathbf{R_{yx}}
\end{aligned}$$

Si no es cierta la hipótesis de partida que la media de las variables implicadas sea nula, puede operarse inicialmente con la variable aleatoria menos su media, obteniendo como resultado:

$$\hat{\mathbf{x}} = \overline{\mathbf{x}} + \mathbf{R_{xy}}{\mathbf{R_y}}^{-1}(\mathbf{y} - \overline{\mathbf{y}}) \tag{3.9}$$

3.3.2. Modelo lineal

Un caso particular de nuestro interés es aquél en el que la variable aleatoria a estimar $\mathbf{x}$ y la variable observada $\mathbf{y}$ están relacionadas linealmente mediante la matriz $\mathbf{A}$, más cierta perturbación aditiva $\mathbf{v}$ no determinista, tal y como se muestra en la figura 3.4

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{v} \tag{3.10}$$

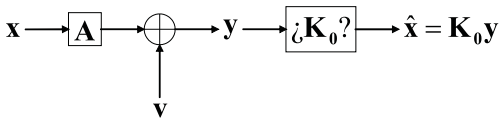

Figura 3.4: Estimación de $\mathbf{x}$ a partir de la observación de $\mathbf{y}$.

También en este caso, podemos intentar estimar $\mathbf{x}$ a partir de cierta combinación lineal de $\mathbf{y}$, por lo que son válidas las conclusiones y resultados anteriores:

$$\hat{\mathbf{x}} = \mathbf{R_{xy}}{\mathbf{R_y}}^{-1}\mathbf{y}$$

En esta nueva situación, suponiendo que el ruido y la variable a estimar están incorreladas ($\mathbf{R_{xv}} = \mathbf{0}$):

$$\mathbf{R_y} = \mathrm{E}\{\mathbf{y}\mathbf{y}^*\} = \mathrm{E}\{(\mathbf{A}\mathbf{x} + \mathbf{v})(\mathbf{x}^*\mathbf{A}^* + \mathbf{v}^*)\} = \mathbf{A}\mathbf{R_x}\mathbf{A}^* + \mathbf{R_v}$$

y

$$\mathbf{R_{xy}} = \mathrm{E}\{\mathbf{x}\mathbf{y}^*\} = \mathrm{E}\{\mathbf{x}(\mathbf{x}^*\mathbf{A}^* + \mathbf{v}^*)\} = \mathbf{R_x}\mathbf{A}^*$$

Por lo tanto, la *estimación lineal por mínimos cuadrados medios* buscada tendrá como

solución, según la expresión (3.8):

$$\hat{\mathbf{x}} = \mathbf{R_x}\mathbf{A}^* \left(\mathbf{A}\mathbf{R_x}\mathbf{A}^* + \mathbf{R_v}\right)^{-1} \mathbf{y}$$

o, alternativamente, empleando el lema de inversión de matrices[3] (si $\mathbf{R_x}$ y $\mathbf{R_v}$ son invertibles)

$$\hat{\mathbf{x}} = \left(\mathbf{R_x}^{-1} + \mathbf{A}^*\mathbf{R_v}^{-1}\mathbf{A}\right)^{-1} \mathbf{A}^*\mathbf{R_v}^{-1}\mathbf{y} \tag{3.11}$$

Si tanto $\mathbf{y}$ como $\mathbf{x}$ no tienen media nula, es decir, $\mathrm{E}\{\mathbf{y}\} = \overline{\mathbf{y}} = \mathbf{A}\overline{\mathbf{x}} + \overline{\mathbf{v}} \neq \mathbf{0}$, y $\mathrm{E}\{\mathbf{x}\} = \overline{\mathbf{x}} \neq \mathbf{0}$ la solución al problema puede reducirse a los mismos términos que anteriormente, estimando $\hat{\mathbf{x}} - \overline{\mathbf{x}}$ en función de $\hat{\mathbf{y}} - \overline{\mathbf{y}}$, ahora ya con media nula, y empleando las matrices de covarianzas o covarianzas cruzadas en vez de las de correlación[4], con lo que la estimación quedaría como:

$$\hat{\mathbf{x}} = \overline{\mathbf{x}} + \mathrm{cov}(\mathbf{x})\mathbf{A}^* \left(\mathbf{A}\mathrm{cov}(\mathbf{x})\mathbf{A}^* + \mathrm{cov}(\mathbf{v})\right)^{-1} \left(\mathbf{y} - \mathbf{A}\overline{\mathbf{x}} - \overline{\mathbf{v}}\right)$$

o, alternativamente

$$\hat{\mathbf{x}} = \overline{\mathbf{x}} + \left(\mathrm{cov}^{-1}(\mathbf{x}) + \mathbf{A}^*\mathrm{cov}^{-1}(\mathbf{v})\mathbf{A}\right)^{-1} \mathbf{A}^*\mathrm{cov}^{-1}(\mathbf{v})(\mathbf{y} - \mathbf{A}\overline{\mathbf{x}} - \overline{\mathbf{v}}) \tag{3.12}$$

3.3.3. Nexos entre los problemas LS estocásticos y deterministas

Si comparamos las ecuaciones (3.12) y (3.7) podremos observar una analogía inmediata si $\overline{\mathbf{v}} = 0$ y $\mathrm{cov}(\mathbf{v}) = \mathbf{I}$, siendo entonces $\overline{\mathbf{x}}$ equivalente a $\mathbf{x}_0$ y $\mathrm{cov}(\mathbf{x})$ equivalente a $\mathbf{\Pi}_0$.

Por lo tanto, si queremos resolver un problema determinista, podemos concebir la solución como si fuera un problema estocástico con $\overline{\mathbf{v}} = 0$, $\mathrm{cov}(\mathbf{v}) = \mathbf{I}$, $\overline{\mathbf{x}} = \mathbf{x}_0$ y $\mathrm{cov}(\mathbf{x}) =$

[3] $(\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{DA}^{-1}\mathbf{B} + \mathbf{C}^{-1}\right)^{-1}\mathbf{DA}^{-1}$, si $\mathbf{A}$ y $\mathbf{C}$ son invertibles

[4] $\mathrm{cov}(\mathbf{a}, \mathbf{b}) = \mathrm{E}\{(\mathbf{a} - \overline{\mathbf{a}})(\mathbf{b} - \overline{\mathbf{b}})^*\}; \mathrm{cov}(\mathbf{a}) = \mathrm{cov}(\mathbf{a}, \mathbf{a})$

$\mathbf{\Pi}_0$, y a la inversa.

3.4 Soluciones recursivas y totales

En aplicaciones de tiempo real, se requiere que la solución a cierto problema haya sido obtenida antes de una cota de tiempo. En ocasiones, la información necesaria para resolver el problema es entregada por un subsistema de adquisición de datos, con cierta cadencia hasta completar la totalidad. De forma general, el subsistema de cálculo deberá esperar a tener toda la información para empezar a procesarla. Existen ciertos problemas cuya solución total o final puede plantearse a partir de soluciones anteriores basadas en sólo parte de los datos; estas soluciones o problemas se definen como recursivos.

Habitualmente, el tiempo que requiere la obtención de la solución con el método total suele ser inferior al método recursivo, pero tiene el inconveniente de necesitar toda la información. Por el contrario, las soluciones o métodos recursivos pueden ir trabajando entre muestra y muestra de manera que cuando se entregue la última muestra, la solución final se obtenga de manera prácticamente inmediata. Esto, siempre y cuando el sistema se capaz de actualizar la solución en el intervalo de tiempo comprendido entre la entrega de dos muestras consecutivas; de esta forma, el método recursivo es más eficiente que el total. En el marco de esta tesis doctoral trabajaremos con este tipo de métodos o problemas recursivos.

En ocasiones, los problemas recursivos pueden parametrizarse tratando más de una muestra al mismo tiempo; ello da lugar a una dimensión más a tener en cuenta en el momento de prever el coste temporal del tratamiento y realizar posibles ajustes que relacionen períodos de muestreo con tiempos de ejecución. De esta forma, el tratamiento puede estar entre dos extremos: muestra a muestra y la totalidad de las muestras. En el último caso estaríamos frente al método total y en el primero, en el método recursivo elemental.

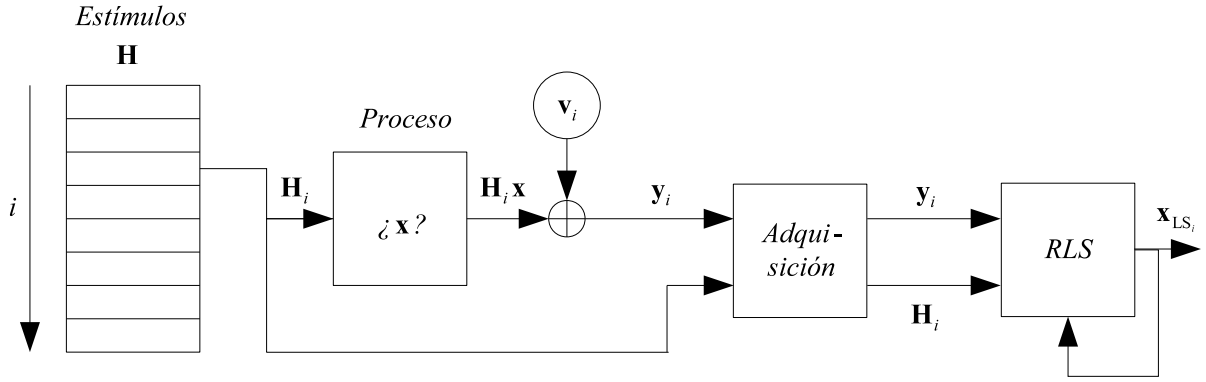

Figura 3.5: Esquema de estimación RLS

3.5 El problema de mínimos cuadrados recursivos

Si poseemos ya la solución mínimos cuadrados al disponer de cierta cantidad de filas de la matriz de datos, la intención es, al obtener del sistema de adquisición *nuevas* filas, recalcular una nueva solución (actualizada) con la nueva matriz basándonos en la solución anterior sin tener que realizar los cálculos desde el principio. Este es el denominado problema de mínimos cuadrados recursivos (RLS —*Recursive Least Squares*—).

Ésta es la variante del problema de mínimos cuadrados en la que vamos a centrar toda nuestra atención debido a su uso frecuente en situaciones prácticas reales.

Será condición sine qua non que todos los algoritmos implementados en lo referente a mínimos cuadrados proporcionen en cada iteración la solución RLS única, para proceder a realizar las comparaciones pertinentes de una manera más equitativa.

En la figura 3.5 se muestra gráficamente una situación concreta en la que se pretende conocer la incóginta $\mathbf{x}$ a partir del conocimiento de los estímulos aplicados al sistema, la respuesta del mismo y la solución al problema en el instante anterior $\mathbf{x}_{\mathrm{LS}_{i-1}}$.

3.6 El problema RLS basado en la actualización de la factorización QR

3.6.1. Actualización de la factorización QR

Sea la factorización QR de $\mathbf{A}_i = \mathbf{Q}_i\mathbf{R}_i$ la expuesta en la subseccion 3.2.2. La factorización QR de la matriz *aumentada* $\mathbf{A}_{i+1} \in \mathbb{R}^{(m_i+q_i)\times n}$ podemos escribirla, utilizando los resultados de la factorización QR de $\mathbf{A}_i$, como, [90, 87]:

$$\mathbf{A}_{i+1} = \begin{pmatrix} \mathbf{A}_i \\ \mathbf{W}_i \end{pmatrix} = \mathbf{Q}_{i+1,a}\mathbf{R}_{i+1,a}$$

donde $\mathbf{W}_i \in \mathbb{R}^{q_i\times n}$,

Podemos descomponer

$$\begin{pmatrix} \mathbf{A}_i \\ \mathbf{W}_i \end{pmatrix} = \begin{pmatrix} \mathbf{Q}_{i,a} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{q_i} \end{pmatrix} \begin{pmatrix} \mathbf{R}_{i,a} \\ \mathbf{W}_i \end{pmatrix}$$

y obtener una matriz triangular superior $\mathbf{T}_{i+1} \in \mathbb{R}^{(n+q_i)\times n}$ por medio de una transformación ortogonal $\mathbf{H}_{i+1} \in \mathbb{R}^{(n+q_i)\times(n+q_i)}$:

$$\mathbf{H}_{i+1} \begin{pmatrix} \mathbf{R}_{i,a} \\ \mathbf{W}_i \end{pmatrix} = \mathbf{T}_{i+1} = \begin{pmatrix} \mathbf{R}_{i+1,a} \\ \mathbf{0} \end{pmatrix} \tag{3.13}$$

donde $\mathbf{R}_{i+1,a}$ es la matriz triangular superior de la factorización QR de $\mathbf{A}_{i+1}$. Entonces

$$\begin{aligned}
\begin{pmatrix} \mathbf{A}_i \\ \mathbf{W}_i \end{pmatrix} &= \begin{pmatrix} \mathbf{Q}_{i,a} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{q_i} \end{pmatrix} \mathbf{H}_{i+1}^T \mathbf{H}_{i+1} \begin{pmatrix} \mathbf{R}_{i,a} \\ \mathbf{W}_i \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{Q}_{i,a} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{q_i} \end{pmatrix} \mathbf{H}_{i+1}^T \begin{pmatrix} \mathbf{R}_{i+1,a} \\ \mathbf{0} \end{pmatrix} \\
&= (\mathbf{Q}_{i+1,a} \quad \mathbf{M}) \begin{pmatrix} \mathbf{R}_{i+1,a} \\ \mathbf{0} \end{pmatrix} \\
&= \mathbf{Q}_{i+1,a}\mathbf{R}_{i+1,a}
\end{aligned}$$

por lo que

$$(\mathbf{Q}_{i+1,a} \quad \mathbf{M}) = \begin{pmatrix} \mathbf{Q}_{i,a} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{q_i} \end{pmatrix} \mathbf{H}_{i+1}^T \quad \text{ó} \quad \begin{pmatrix} \mathbf{Q}_{i+1,a}^T \\ \mathbf{M}^T \end{pmatrix} = \mathbf{H}_{i+1} \begin{pmatrix} \mathbf{Q}_{i,a}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{q_i} \end{pmatrix} \tag{3.14}$$

en principio, no estaremos interesados en la submatriz $\mathbf{M}$.

Más adelante, estaremos interesados en el producto matriz-vector $\mathbf{Q}_{i+1,a}^T \mathbf{y}_{i+1}$, donde $\mathbf{y}_{i+1} = \begin{pmatrix} \mathbf{y}_i^T & \mathbf{y}_{q_i}^T \end{pmatrix}^T$, con $\mathbf{y}_i \in \mathbb{R}^{m_i}$ y $\mathbf{y}_{q_i} \in \mathbb{R}^{q_i}$, entonces:

$$\begin{aligned}
\mathbf{H}_{i+1} \begin{pmatrix} \mathbf{Q}_{i,a}^T \mathbf{y}_i \\ \mathbf{y}_{q_i} \end{pmatrix} &= \mathbf{H}_{i+1} \begin{pmatrix} \mathbf{Q}_{i,a}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{q_i} \end{pmatrix} \begin{pmatrix} \mathbf{y}_i \\ \mathbf{y}_{q_i} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{Q}_{i+1,a}^T \\ \mathbf{M}^T \end{pmatrix} \mathbf{y}_{i+1} \\
&= \begin{pmatrix} \mathbf{Q}_{i+1,a}^T \mathbf{y}_{i+1} \\ \mathbf{z} \end{pmatrix}
\end{aligned}$$

El vector $\mathbf{z}$ tampoco será de nuestro interés. Podemos obtener estos mismos resultados

aplicando $\mathbf{H}_{i+1}$ a la matriz compuesta siguiente:

$$\mathbf{H}_{i+1}\left(\begin{array}{c|cc|c} \mathbf{R}_{i,a} & \mathbf{Q}_{i,a}^T & \mathbf{0} & \mathbf{Q}_{i,a}^T\mathbf{y}_i \\ \mathbf{W}_i & \mathbf{0} & \mathbf{I}_{q_i} & \mathbf{y}_{q_i} \end{array}\right) = \left(\begin{array}{c|c|c} \mathbf{R}_{i+1,a} & \mathbf{Q}_{i+1,a}^T & \mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1} \\ \mathbf{0} & \mathbf{M}^T & \mathbf{z} \end{array}\right) \tag{3.15}$$

Esta forma de organizar la actualización no necesita realmente crear ni almacenar la transformación ortogonal, simplemente aplicarla. Esto es ventajoso si empleamos una secuencia de transformaciones de Householder como transformación ortogonal.

En [91] se muestran resultados de esta actualización. En una versión previa de esta referencia, se sugiere la representación WY [92] en el caso de necesitar acumular las transformaciones.

3.6.2. El problema RLS utilizando la actualización de la factorización QR

Podemos encontrar la *solución mínimos cuadrados* del sistema

$$\mathbf{A}_i\mathbf{x} + \mathbf{v}_i = \mathbf{y}_i$$

donde $\mathbf{A}_i \in \mathbb{R}^{m_i \times n}$, con $m_i \geq n$, como en (3.5):

$$\mathbf{x}_{\mathrm{LS},i} = \mathbf{R}_{i,a}^{-1}\mathbf{Q}_{i,a}^T\mathbf{y}_i$$

siendo $\mathbf{A}_i = \mathbf{Q}_i\mathbf{R}_i = \mathbf{Q}_{i,a}\mathbf{R}_{i,a}$ la factorización QR de $\mathbf{A}_i$.

La solución mínimos cuadrados $\mathbf{x}_{\mathrm{LS},i+1}$ del sistema *actualizado*

$$\begin{array}{rcl} \left(\begin{array}{c} \mathbf{A}_i \\ \mathbf{W}_i \end{array}\right)\mathbf{x} + \left(\begin{array}{c} \mathbf{v}_i \\ \mathbf{v}_{q_i} \end{array}\right) & = & \left(\begin{array}{c} \mathbf{y}_i \\ \mathbf{y}_{q_i} \end{array}\right) \\ \mathbf{A}_{i+1}\mathbf{x} + \mathbf{v}_{i+1} & = & \mathbf{y}_{i+1} \end{array}$$

donde $\mathbf{W}_i \in \mathbb{R}^{q_i \times n}$, es

$$\mathbf{x}_{\mathrm{LS}_{i+1}} = \mathbf{R}_{i+1,a}^{-1} \mathbf{Q}_{i+1,a}^T \mathbf{y}_{i+1}$$

pudiéndose calcular utilizando parte de (3.15):

$$\mathbf{H}_{i+1,a} \left(\begin{array}{c|c} \mathbf{R}_{i,a} & \mathbf{Q}_{i,a}^T \mathbf{y}_i \\ \mathbf{W}_i & \mathbf{y}_{q_i} \end{array} \right) = \left(\begin{array}{c|c} \mathbf{R}_{i+1,a} & \mathbf{Q}_{i+1,a}^T \mathbf{y}_{i+1} \\ \mathbf{0} & \mathbf{z} \end{array} \right) \tag{3.16}$$

Mientras $m_i + q_i < n$ evidentemente no será posible calcular la solución mínimos cuadrados única. Si se desea calcular una solución, aunque no sea ni única ni de norma mínima, podemos proceder como se muestra a continuación:

$$\mathbf{R}_{i+1,a} = \left(\begin{array}{cc} \mathbf{R}_1 & \mathbf{R}_2 \end{array} \right) \in \mathbb{R}^{\mathrm{mín}\{m_i+q_i,n\} \times n}$$

con

$$\begin{aligned} \mathbf{R}_1 &\in \mathbb{R}^{\mathrm{mín}\{m_i+q_i,n\} \times \mathrm{mín}\{m_i+q_i,n\}} \\ \mathbf{R}_2 &\in \mathbb{R}^{\mathrm{mín}\{m_i+q_i,n\} \times (n - \mathrm{mín}\{m_i+q_i,n\})} \end{aligned}$$

y

$$\mathbf{x} = \left(\begin{array}{c} \mathbf{x}_1 \\ \mathbf{x}_2 \end{array} \right)$$

con $\mathbf{x}_1 \in \mathbb{R}^{\mathrm{mín}\{m_i+q_i,n\}}$ y $\mathbf{x}_1 \in \mathbb{R}^{n - \mathrm{mín}\{m_i+q_i,n\}}$, entonces

$$
\begin{aligned}
||\mathbf{A}_{i+1}\mathbf{x} - \mathbf{y}_{i+1}||_2^2 &= ||\mathbf{Q}_{i+1,a}\mathbf{R}_{i+1,a}\mathbf{x} - \mathbf{y}_{i+1}||_2^2 \\
&= ||\mathbf{R}_{i+1,a}\mathbf{x} - \mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1}||_2^2 \\
&= \left|\left|\begin{pmatrix} \mathbf{R}_1 & \mathbf{R}_2 \end{pmatrix}\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} - \mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1}\right|\right|_2^2 \\
&= ||\mathbf{R}_1\mathbf{x}_1 + \mathbf{R}_2\mathbf{x}_2 - \mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1}||_2^2
\end{aligned}
$$

si forzamos a que $\mathbf{x}_2 = \mathbf{0}$ entonces

$$
||\mathbf{A}_{i+1}\mathbf{x} - \mathbf{y}_{i+1}||_2^2 = ||\mathbf{R}_1\mathbf{x}_1 - \mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1}||_2^2
$$

por lo que

$$
\mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{R}_1^{-1}\mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1} \\ \mathbf{0} \end{pmatrix}
$$

De esta forma, se ofrece continuidad en el proceso del algoritmo al pasar de solución no única a solución única.

El **Algoritmo 1** muestra el pseudocódigo de una iteración del algoritmo RLS. Al comienzo del algoritmo, no existe todavía factorización QR, por lo que unos valores iniciales para las variables pueden ser $\left(\mathbf{R}_{i,a},\ \ (\mathbf{Q}_{i,a}^T\mathbf{y}_i)\right) = \mathbf{0}$.

A este algoritmo y sus variantes se les denomina en la literatura de Teoría de la Señal como QRD (QR Decomposition). Existe una variante interesante que se basa en propagar en vez de $\mathbf{R}_{i,a}$, su inversa $\mathbf{R}_{i,a}^{-1}$, por lo que no es necesario el proceso de sustitución regresiva para calcular la solución; a esta última variante se le conoce con el nombre de *Inverse* QR o IQRD. Una primera propuesta de esta variación aparece en [93]. Algunos algoritmos

Algoritmo 1 RLS secuencial: iteración RLS por actualización de QR

Entrada: $\mathbf{R}_{i,a} \in \mathbb{R}^{\min\{m_i,n\}\times n}$, $(\mathbf{Q}_{i,a}^T\mathbf{y}_i) \in \mathbb{R}^{\min\{m_i,n\}}$, $\mathbf{W}_i \in \mathbb{R}^{q_i\times n}$, $\mathbf{y}_{q_i} \in \mathbb{R}^{q_i}$

Salida: $\mathbf{R}_{i+1,a} \in \mathbb{R}^{\min\{m_i+q_i,n\}\times n}$, $(\mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1}) \in \mathbb{R}^{\min\{m_i+q_i,n\}}$, y $\mathbf{x}_{\mathrm{LS},i} \in \mathbb{R}^n$

1: Calcular $\mathbf{H}_{i+1,a} \in \mathbb{R}^{(\min\{m_i,n\}+q_i)\times(\min\{m_i,n\}+q_i)}$: transformación ortogonal que triangulariza superiormente $\begin{pmatrix} \mathbf{R}_{i,a}^T & \mathbf{W}_i^T \end{pmatrix}^T$

2: Aplicar $\mathbf{H}_{i+1,a}$

$$\begin{pmatrix} \mathbf{R}_{i+1,a} & (\mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1}) \\ \mathbf{0} & \mathbf{z} \end{pmatrix} \leftarrow \mathbf{H}_{i+1,a} \begin{pmatrix} \mathbf{R}_{i,a} & (\mathbf{Q}_{i,a}^T\mathbf{y}_i) \\ \mathbf{W}_i & \mathbf{y}_{q_i} \end{pmatrix}$$

3: **si** $(m_i + q_i) \geq n$ **entonces**

4:

$$\mathbf{x}_{\mathrm{LS}_{i+1}} \leftarrow \mathbf{R}_{i+1,a}^{-1}\left(\mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1}\right)$$

5: **sino**

6:

$$\begin{aligned} \mathbf{R}_1 &\leftarrow \mathbf{R}_{i+1,a}(:, 1 : \min\{m_i + q_i, n\}) \\ \mathbf{x}_{\mathrm{LS}_{i+1}} &\leftarrow \begin{pmatrix} \mathbf{R}_1^{-1}\mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1} \\ \mathbf{0} \end{pmatrix} \end{aligned}$$

7: **fin si**

derivados del Filtro de Kalman, que se comentan en la siguiente sección, están íntimamente ligados con esta última variedad.

3.7 El Filtro de Kalman

El uso del Filtro de Kalman para la estimación recursiva en el espacio de estados de cierto sistema, como un algoritmo para resolver el problema RLS, se muestra en [13]. En esta referencia se pone en evidencia la similitud, y en algunos casos coincidencia, de numerosos algoritmos que con distinta denominación en distintas disciplinas, se corresponden en el fondo al mismo algoritmo.

Consideremos el siguiente sistema descrito por sus ecuaciones en el espacio de estados:

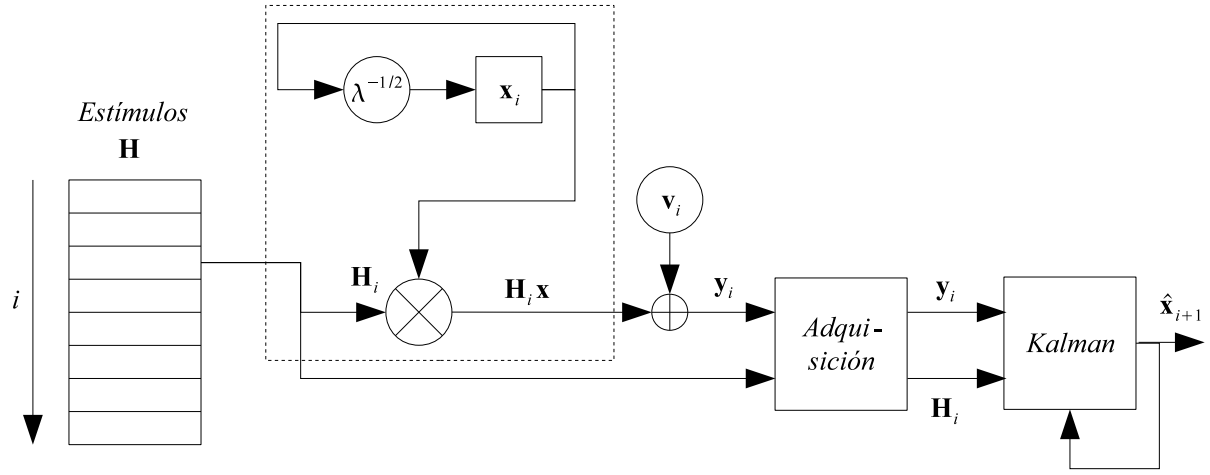

Figura 3.6: Estimación del estado del sistema

$$\mathbf{x}_{i+1} = \lambda^{-1/2}\mathbf{x}_i \tag{3.17}$$

$$\mathbf{y}_i = \mathbf{H}_i\mathbf{x}_i + \mathbf{v}_i \tag{3.18}$$

donde $\mathbf{x}_i \in \mathbb{C}^{n\times 1}$ es el vector del estado cuya evolución viene dictada por (3.17), $\mathbf{y}_i \in \mathbb{C}^{q\times 1}$ es el vector de la observación, $\mathbf{v}_i \in \mathbb{C}^{q\times 1}$ es el vector del ruido en la observación, $\mathbf{H}_i \in \mathbb{C}^{q\times n}$ es la denominada matriz de medida, $0 \ll \lambda \leq 1$, puede interpretarse como un factor de *olvido*, $q \ll n$ es el número de observaciones que serán procesadas en una iteración del algoritmo e $i \geq 0$ es el índice de la iteración.

El Filtro de Kalman calcula recursivamente $\hat{\mathbf{x}}_{i+1}$, es decir, una estimación lineal por mínimos cuadrados medios del estado $\mathbf{x}_{i+1}$, tal y como se muestra gráficamente en la figura 3.6 y a continuación algebraicamente [13]:

Entrada: $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$, $\mathbf{y} = (\mathbf{y}_0^*, \mathbf{y}_1^*, \ldots)^*$, $\mathbf{P}_0$, $\hat{\mathbf{x}}_0$, λ

Salida: Actualización de la estimación del estado, $\hat{\mathbf{x}}_{i+1}$, cada iteración i-ésima

para $i = 0, \ldots$ **hacer**

$$\hat{\mathbf{x}}_{i+1} = \lambda^{-1/2}\left(\hat{\mathbf{x}}_i + \mathbf{P}_i\mathbf{H}_i^*\left[\mathbf{H}_i\mathbf{P}_i\mathbf{H}_i^* + \mathbf{I}\right]^{-1}\left(\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i\right)\right) \tag{3.19}$$

$$\mathbf{P}_{i+1} = \mathbf{P}_{i+1}^* = \lambda^{-1}\left(\mathbf{P}_i - \mathbf{P}_i\mathbf{H}_i^*\left[\mathbf{H}_i\mathbf{P}_i\mathbf{H}_i^* + \mathbf{I}\right]^{-1}\mathbf{H}_i\mathbf{P}_i\right) \tag{3.20}$$

fin para

La variable $\mathbf{P}_{i+1}$ satisface la recursión en diferencias de Riccati, con $\mathbf{P}_i = \mathrm{cov}(\mathbf{x}_i - \hat{\mathbf{x}}_i) \in \mathbb{C}^{n\times n}$ —la matriz de covarianza del error en la estimación—, siendo $\mathbf{P}_0 = \mathrm{cov}(\mathbf{x}_0)$. La variable $\overline{\mathbf{x}}_0$ es el valor esperado del estado inicial y su estimación es $\hat{\mathbf{x}}_0 = \overline{\mathbf{x}}_0$, con $\mathrm{cov}(\mathbf{x}_0, \mathbf{v}_i) = \mathbf{0}$; finalmente $\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i$ es el error en la estimación de la observación.

De (3.17), $\mathbf{x}_M = \left(\lambda^{-1/2}\right)^M \mathbf{x}_0$. Si $\hat{\mathbf{x}}_M$ is la estimación *l.l.m.s.e* por parte del Filtro de Kalman de $\mathbf{x}_M$ dados $\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{M-1}$, entonces

$$\hat{\mathbf{x}}_0 = \left(\lambda^{1/2}\right)^M \hat{\mathbf{x}}_M \tag{3.21}$$

Si expandimos la ecuación (3.18) para $0 \le i \le M-1$, utilizamos (3.21) y definimos

$$\begin{aligned}
\mathbf{y} &= \left(\mathbf{y}_0^*, \mathbf{y}_1^*, \ldots, \mathbf{y}_{M-1}^*\right)^* \\
\mathbf{H} &= \left(\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots, \mathbf{H}_{M-1}^*\right)^* \\
\mathbf{v} &= \left(\mathbf{v}_0^*, \mathbf{v}_1^*, \ldots, \mathbf{v}_{M-1}^*\right)^* \\
\mathbf{\Delta} &= \mathrm{diag}(\lambda^{-i/2}, i = 0, \ldots M-1) \\
\mathbf{A} &= \mathbf{\Delta H}
\end{aligned}$$

entonces

$$\mathbf{y} = \mathbf{\Delta H x}_0 + \mathbf{v} = \mathbf{A x}_0 + \mathbf{v} \tag{3.22}$$

Si ahora estimamos $\mathbf{x}_0$ con el Filtro de Kalman y (3.21), entonces podemos resolver un

poblema de mínimos cuadrados (3.22) con ciertas condiciones iniciales[5] atendiendo a la relación existente entre los problemas de mínimos cuadrados deterministas y estocásticos. De (3.22), si $m = Mq$, entonces $\mathbf{A}, \mathbf{H} \in \mathbb{C}^{m \times n}$, $\mathbf{y}, \mathbf{v} \in \mathbb{C}^m$, y $\mathbf{x} \in \mathbb{C}^n$. Ya que el Filtro de Kalman actualiza *recursivamente* la solución con cada muestra de datos (o con cada q muestras de datos), la solución al problema de mínimos cuadrados puede ser actualizada *recursivamente* de la misma forma.

3.7.1. Variantes del Filtro de Kalman

En [13] podemos encontrar numerosas variantes del algoritmo del Filtro de Kalman básico: el *Filtro de Información*, en el que la inversa de la variable de Riccati $\mathbf{P}_i$ es propagada a lo largo de las iteraciones en vez de $\mathbf{P}_i$, evitando entonces trabajar con valores iniciales quizá extremadamente grandes para $\mathbf{P}_i$ cuando la incertidumbre sobre el estado inicial es alta; el *Filtro de Kalman en versión raíz cuadrada*, donde $\mathbf{P}_i^{1/2}$ —un factor raíz cuadrada de $\mathbf{P}_i$— es propagada en vez de $\mathbf{P}_i$, asegurando entonces que $\mathbf{P}_i^{*/2}\mathbf{P}_i^{1/2}$ es numéricamente definida positiva; el *Filtro de Información en versión raíz cuadrada y extendida*, donde un factor raíz cuadrada de $\mathbf{P}_i^{-1}$ y la inversa del otro son propagados; el *Filtro de Chandrasekhar en versión raíz cuadrada*, utilizado cuando $\mathbf{H}$ es una matriz *estructurada*, disminuyendo notablemente el coste computacional, el *Filtro de Chandrasekhar en versión explícita*, ...

En la presente tesis doctoral, analizaremos e implementaremos versiones secuenciales y paralelas del *Filtro de Kalman en versión raíz cuadrada* y del *Filtro de Información en versión raíz cuadrada*, debido a sus a priori buenas propiedades para la paralelización (alto incremento de velocidad y eficiencia, un relativamente fácil equilibrado de carga, tanto para sistemas homogéneos como heterogéneos, etc.) y a su genericidad respecto al tipo de problemas que pueden tratarse.

[5] $\hat{\mathbf{x}}_0 = \mathbf{0}, \quad \mathbf{P}_0^{1/2} = \epsilon^{-1/2}\mathbf{I}_n$, con ϵ suficientemento pequeño [13].

3.7.2. Algoritmos raíz cuadrada

Los algoritmos raíz cuadrada son versiones algorítmicas en la que las operaciones se agrupan de forma matricial apareciendo en las expresiones obtenidas factores raíz cuadrada de una matriz. Para ello, la o las matrices en cuestión deben ser definidas positivas y en tal caso, podemos escribir

$$\mathbf{A} = \mathbf{A}^{1/2}\mathbf{A}^{*/2}$$

Esta descomposición como productos de *factores raíz cuadrada* no es única, lo cual supone cierta ventaja, ya que podemos hacer que cierta estructura matricial conveniente se propague entre las distintas iteraciones del algoritmo.

A continuación, se mostrarán sucintamente dos versiones raíz cuadrada de algoritmos que resuelven el problema RLS, que serán detallados en capítulos posteriores.

Algoritmo raíz cuadrada del Filtro de Kalman

Veamos un primer ejemplo denominado *versión raíz cuadrada del Filtro de Kalman*, donde se aprecia la caracterización de esta variedad en los algoritmos raíz cuadrada y sus ventajas.

Entrada: $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$, $\mathbf{y} = (\mathbf{y}_0^*, \mathbf{y}_1^*, \ldots)^*$, $\hat{\mathbf{x}}_0$, $\mathbf{P}_0^{1/2}$, λ

Salida: Actualización de $\mathbf{x}_{\mathrm{LS}}$ cada iteración i-ésima

para $i = 0, \ldots, m/q - 1$ **hacer**

Calcular y aplicar $\mathbf{\Theta}_i$ unitaria tal que:

$$\mathbf{E}_i\mathbf{\Theta}_i = \begin{pmatrix} \mathbf{I}_q & \mathbf{H}_i\mathbf{P}_i^{1/2} \\ \mathbf{0} & \lambda^{-1/2}\mathbf{P}_i^{1/2} \end{pmatrix} \mathbf{\Theta}_i = \begin{pmatrix} \mathbf{R}_{e,i}^{1/2} & \mathbf{0} \\ \overline{\mathbf{K}}_{p,i} & \mathbf{P}_{i+1}^{1/2} \end{pmatrix} = \mathbf{F}_i$$

$$\hat{\mathbf{x}}_{i+1} = \lambda^{-1/2}\hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i}\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i\right)$$

$$\mathbf{x}_{\mathrm{LS}_i} = \left(\lambda^{1/2}\right)^{i+1}\hat{\mathbf{x}}_{i+1}$$

fin para

---

donde $\mathbf{P}_i^{1/2}$ es un factor raíz cuadrada de $\mathbf{P}_i$, es decir $\mathbf{P}_i = \mathbf{P}_i^{1/2}\mathbf{P}_i^{*/2}$, siendo ésta la matriz de covarianza del error en la solución, que se supone definida positiva, dadas las características de los datos del problema; la matriz $\mathbf{R}_{e,i} = \mathbf{H}_i\mathbf{P}_i\mathbf{H}_i^* + \mathbf{I}_q$, teniendo como factor raíz cuadrada a $\mathbf{R}_{e,i}^{1/2}$; por último, la matriz $\overline{\mathbf{K}}_{p,i} = \lambda^{-1/2}\mathbf{P}_i\mathbf{H}_i^*\mathbf{R}_{e,i}^{-*/2}$. A partir de estas definiciones, podemos comprobar que el estado se actualiza de la misma forma que en (3.19). La diferencia, por tanto, radica en cómo se propaga la matriz de covarianza del error en la estimación del estado, siendo, en este caso, el factor raíz cuadrada $\mathbf{P}_i^{1/2}$ el que realmente es propagado.

Si formamos $\mathbf{E}_i$, tal y como se indica en el algoritmo, y mediante una transformación unitaria, conseguimos $\mathbf{F}_i$ y actualizamos la solución, entonces estamos realizando los mismos pasos que en las ecuaciones (3.19) y (3.20). Una comprobación parcial se muestra a continuación:

$$
\begin{aligned}
\mathbf{E}_i\boldsymbol{\Theta}_i\boldsymbol{\Theta}_i^*\mathbf{E}_i^* &= \mathbf{F}_i\mathbf{F}_i^* \\
\mathbf{E}_i\mathbf{E}_i^* &= \mathbf{F}_i\mathbf{F}_i^* \\
\begin{pmatrix} \mathbf{I}_q & \mathbf{H}_i\mathbf{P}_i^{1/2} \\ \mathbf{0} & \lambda^{-1/2}\mathbf{P}_i^{1/2} \end{pmatrix}
\begin{pmatrix} \mathbf{I}_q & \mathbf{0} \\ \mathbf{P}_i^{*/2}\mathbf{H}_i^* & \lambda^{-1/2}\mathbf{P}_i^{*/2} \end{pmatrix} &= \begin{pmatrix} \mathbf{R}_{e,i}^{1/2} & \mathbf{0} \\ \overline{\mathbf{K}}_{p,i} & \mathbf{P}_{i+1}^{1/2} \end{pmatrix} \\
&\quad \cdot \begin{pmatrix} \mathbf{R}_{e,i}^{*/2} & \overline{\mathbf{K}}_{p,i}^* \\ \mathbf{0} & \mathbf{P}_{i+1}^{*/2} \end{pmatrix}
\end{aligned}
\tag{3.23}
$$

Si multiplicamos la segunda fila de $\mathbf{E}_i$ por la segunda columna de $\mathbf{E}_i^*$, deberíamos obtener el mismo resultado que multiplicar entre sí las mismas filas y columnas de $\mathbf{F}_i$ y $\mathbf{F}_i^*$ respectivamente:

$$
\begin{aligned}
\lambda^{-1/2}\mathbf{P}_i^{1/2}\mathbf{P}_i^{*/2}\lambda^{-1/2} &= \overline{\mathbf{K}}_{p,i}\overline{\mathbf{K}}_{p,i}^* + \mathbf{P}_{i+1}^{1/2}\mathbf{P}_{i+1}^{*/2} \\
\lambda^{-1}\mathbf{P}_i &= \overline{\mathbf{K}}_{p,i}\overline{\mathbf{K}}_{p,i}^* + \mathbf{P}_{i+1}
\end{aligned}
$$

Comprobemos a continuación el valor de

$$
\begin{aligned}
\lambda^{-1}\mathbf{P}_i - \overline{\mathbf{K}}_{p,i}\overline{\mathbf{K}}_{p,i}^* &= \lambda^{-1}\mathbf{P}_i - \lambda^{-1/2}\mathbf{P}_i\mathbf{H}_i^*\mathbf{R}_{e,i}^{-*/2}\lambda^{-1/2}\mathbf{R}_{e,i}^{-1/2}\mathbf{H}_i\mathbf{P}_i^* \\
&= \lambda^{-1}\mathbf{P}_i - \lambda^{-1}\mathbf{P}_i\mathbf{H}_i^*\mathbf{R}_{e,i}^{-1}\mathbf{H}_i\mathbf{P}_i^* \\
&= \lambda^{-1}\mathbf{P}_i - \lambda^{-1}\mathbf{P}_i\mathbf{H}_i^*\left(\mathbf{H}_i\mathbf{P}_i\mathbf{H}_i^* + \mathbf{I}_q\right)^{-1}\mathbf{H}_i\mathbf{P}_i^* \\
&= \mathbf{P}_{i+1}, \quad \text{según (3.20)}
\end{aligned}
$$

Podríamos proceder de manera similar para el resto de filas y columnas de $\mathbf{E}_i$ y $\mathbf{F}_i$ comprobando que efectivamente se da la igualdad (3.23).

Algoritmo raíz cuadrada del Filtro de Información

Un segundo y último ejemplo es el denominado *Filtro de Información* cuyas iteraciones se muestran en el siguiente algoritmo:

Entrada: $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$, $\mathbf{y} = (\mathbf{y}_0^*, \mathbf{y}_1^*, \ldots)^*$, $\hat{\mathbf{x}}_0$, $\mathbf{P}_0^{1/2}$, λ

Salida: Actualización de $\mathbf{x}_{\mathrm{LS}}$ cada iteración i-ésima

para $i = 0, \ldots, m/q - 1$ **hacer**

Calcular y aplicar $\mathbf{\Theta}_i$ unitaria tal que:

$$\mathbf{E}_i\mathbf{\Theta}_i = \begin{pmatrix} \lambda^{1/2}\mathbf{P}_i^{-*/2} & \lambda^{1/2}\mathbf{H}_i^* \\ \hat{\mathbf{x}}_i^*\mathbf{P}_i^{-*/2} & \mathbf{y}_i^* \\ \lambda^{-1/2}\mathbf{P}_i^{1/2} & \mathbf{0} \end{pmatrix} \mathbf{\Theta}_i = \begin{pmatrix} \mathbf{P}_{i+1}^{-*/2} & \mathbf{0} \\ \hat{\mathbf{x}}_{i+1}^*\mathbf{P}_{i+1}^{-*/2} & (\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i)^*\mathbf{R}_{e,i}^{-*/2} \\ \mathbf{P}_{i+1}^{1/2} & -\overline{\mathbf{K}}_{p,i} \end{pmatrix}$$

$$= \mathbf{F}_i$$

$$\hat{\mathbf{x}}_{i+1} = \lambda^{-1/2}\hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i}\mathbf{R}_{e,i}^{-1/2}(\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i)$$

$$\mathbf{x}_{\mathrm{LS}_i} = \left(\lambda^{1/2}\right)^{i+1}\hat{\mathbf{x}}_{i+1}$$

fin para

En este caso, lo que se propaga de iteración a iteración son los factores raíz cuadrada de $\mathbf{P}_i^{-1} = \mathbf{P}_i^{-*/2}\mathbf{P}_i^{-1/2}$ (en este caso, la matriz $\mathbf{P}_i^{1/2}$ que aparece en el algoritmo es la inversa de $\mathbf{P}_i^{-1/2}$ de la factorización raíz cuadrada mostrada). De nuevo, podemos comprobar que se vuelve a cumplir la igualdad $\mathbf{E}_i\mathbf{E}_i^* = \mathbf{F}_i\mathbf{F}_i^*$.

De modo que mediante transformaciones unitarias conseguimos lo mismo que de manera explícita con las ecuaciones del algoritmo básico, con las ventajas que numéricamente ello aporta. Este es el fundamento de los algoritmos raíz cuadrada que estudiaremos en sendos capítulos posteriores. En [94] se muestra una comparativa entre varios métodos basados en el Filtro de Kalman, junto con consideraciones en el ruido de la observación e

inicializaciones de los algoritmos.

3.8 Conclusiones

En este capítulo hemos planteado el problema de mínimos cuadrados (LS) como problema a tratar, tanto en su versión determinista como estocástica. El planteamiento recursivo de la solución puede aportar ventajas en aras de conseguir una solución de manera prácticamente inmediata cuando se reciben los últimos datos a procesar, en aplicaciones como las descritas en el apartado de **Introducción y objetivos**. Para este problema RLS hemos descrito algunas soluciones íntimamente relacionadas entre sí: la solución por actualización de la factorización QR y soluciones derivadas del Filtro de Kalman. Todas las variantes tienen en común la propagación de ciertas variables a lo largo de las iteraciones del algoritmo. En el caso de la actualización de la factorización QR, entre otras, se transmite la matriz triangular superior $\mathbf{R}_{i,a}$; en el Filtro de Kalman, la matriz de covarianza del error en la solución $\mathbf{P}_i$; en la versión raíz cuadrada del Filtro de Kalman, la matriz $\mathbf{P}_i^{1/2}$ y en la versión raíz cuadrada del Filtro de Información , los factores $\mathbf{P}_i^{-*/2}$ y $\mathbf{P}_i^{-1/2}$. En los siguientes capítulos se detallan tanto los algoritmos secuenciales como los paralelos de algunas de las versiones citadas.

4

Algoritmo RLS basado en la versión raíz cuadrada del Filtro de Kalman

A continuación mostramos el estudio sobre un primer algoritmo que resuelve el problema RLS, basado en la versión raíz cuadrada del Filtro de Kalman o filtro de covarianza. Finalizaremos con la paralelización del mismo y las conclusiones que de ella hemos obtenido.

4.1 Algoritmo secuencial

Una iteración genérica del algoritmo raíz cuadrada del Filtro de Kalman o filtro de covarianza, que resuelve el problema RLS se muestra en (4.1). En el último subapartado del capítulo anterior, comprobamos que para este algoritmo, $\mathbf{E}_i\mathbf{E}_i^* = \mathbf{F}_i\mathbf{F}_i^*$, siendo por tanto un algoritmo *raíz cuadrada*. Emplearemos el acrónimo SRKF-RLS (*Square Root Kalman Filter*-RLS) para denotar a este algoritmo. En [95] y [96] se muestran dos primeras versiones paralelas de este algoritmo, la primera no orientada a bloque y la segunda, orientada a bloque y ambas no segmentadas. En el presente capítulo mostramos una versión bloque segmentada.

El objetivo principal que nos planteamos en el diseño del algoritmo es el conseguir

hacer ceros de manera eficiente en la matriz de la parte derecha de la igualdad, aplicando sólo transformaciones ortogonales por la derecha. Y aunque las matrices $\mathbf{P}_i^{1/2}$ ó $\mathbf{P}_{i+1}^{1/2}$ no tienen estructura única, conviene, para ahorrar coste aritmético, preservar estructura de matriz triangular inferior; asimismo, aprovecharemos la estructura de la matriz $\mathbf{E}_i$ para minimizar la cantidad de operaciones a realizar.

Entrada: $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$, $\mathbf{y} = (\mathbf{y}_0^*, \mathbf{y}_1^*, \ldots)^*$, $\hat{\mathbf{x}}_0$, $\mathbf{P}_0^{1/2}$

Salida: Actualización de $\mathbf{x}_{\mathrm{LS}}$ cada iteración i-ésima

para $i = 0, \ldots, m/q - 1$ **hacer**

Calcular y aplicar $\mathbf{\Theta}_i$ unitaria tal que:

$$\begin{aligned} \mathbf{E}_i\mathbf{\Theta}_i = \begin{pmatrix} \mathbf{I}_q & \mathbf{H}_i\mathbf{P}_i^{1/2} \\ \mathbf{0} & \lambda^{-1/2}\mathbf{P}_i^{1/2} \end{pmatrix} \mathbf{\Theta}_i &= \begin{pmatrix} \mathbf{R}_{e,i}^{1/2} & \mathbf{0} \\ \overline{\mathbf{K}}_{p,i} & \mathbf{P}_{i+1}^{1/2} \end{pmatrix} = \mathbf{F}_i \\ \hat{\mathbf{x}}_{i+1} &= \lambda^{-1/2}\hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i}\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i\right) \\ \mathbf{x}_{\mathrm{LS}_i} &= \left(\lambda^{1/2}\right)^{i+1}\hat{\mathbf{x}}_{i+1} \end{aligned} \tag{4.1}$$

fin para

La submatriz $\mathbf{H}_i\mathbf{P}_i^{1/2} \in \mathbb{C}^{q\times n}$ es una matriz general. Si obtenemos ceros en esta submatriz por columnas (de derecha a izquierda), podemos preservar la estructura triangular inferior de $\lambda^{-1/2}\mathbf{P}_i^{1/2}$, obteniendo $\mathbf{P}_{i+1}^{1/2}$, y convirtiendo $\mathbf{I}_q$ en una matriz triangular inferior $\mathbf{R}_{e,i}^{1/2}$.

La figura 4.1 muestra gráficamente el contexto en el que puede desenvolverse el algoritmo.

A continuación mostramos un ejemplo con $q = 2$ y $n = 3$.

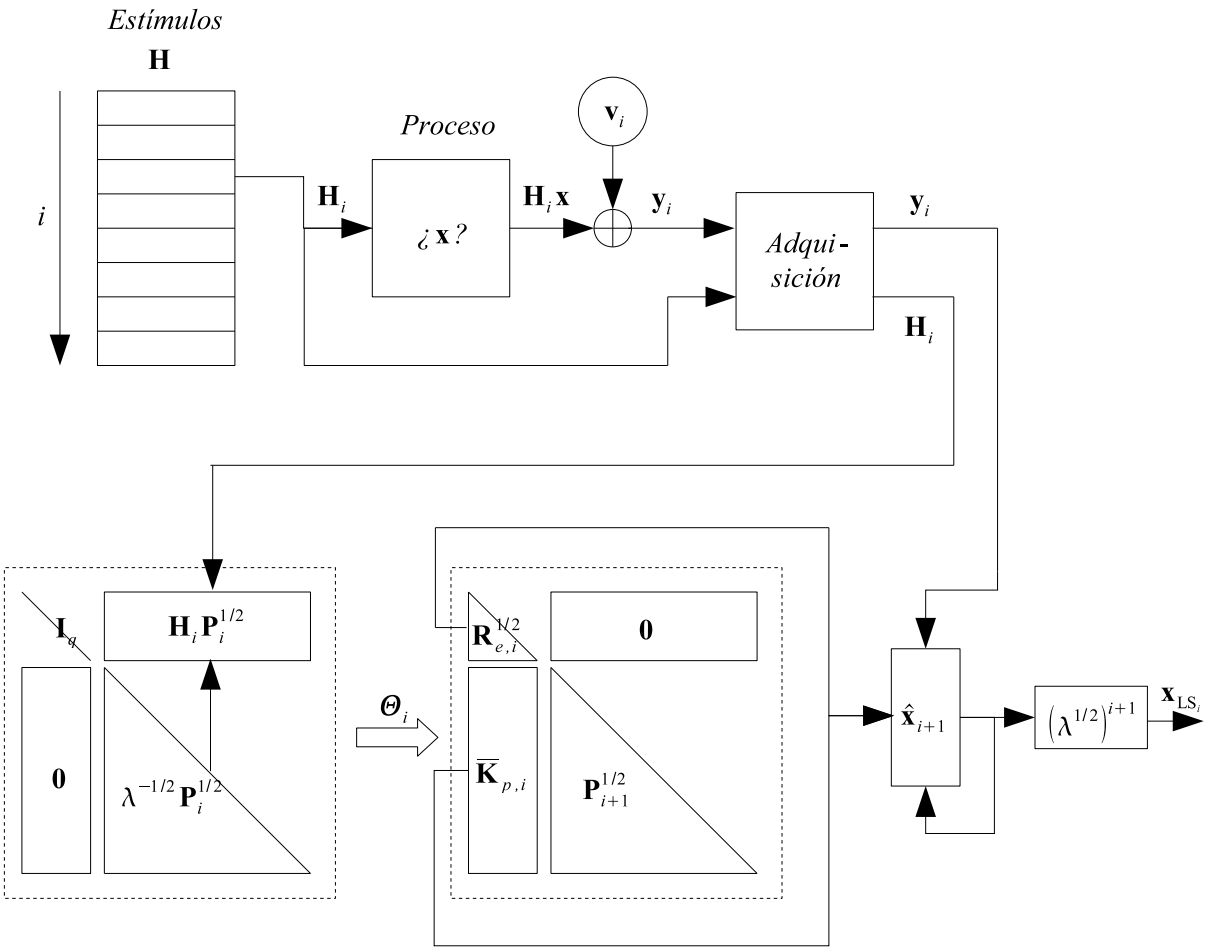

Figura 4.1: Esquema del algoritmo SRKF-RLS

$$\left(\begin{array}{c|c} \mathbf{I}_q & \mathbf{H}_i\mathbf{P}_i^{1/2} \\ \hline \mathbf{0} & \lambda^{-1/2}\mathbf{P}_i^{1/2} \end{array}\right) = \left(\begin{array}{cc|ccc} 1 & 0 & x & x & x \\ 0 & 1 & x & x & x \\ \hline 0 & 0 & x & 0 & 0 \\ 0 & 0 & x & x & 0 \\ 0 & 0 & x & x & x \end{array}\right)$$

Podemos obtener ceros en la última columna de la submatriz $\mathbf{H}_i\mathbf{P}_i^{1/2}$ aplicando dos rotaciones de Givens:

$$\left(\begin{array}{cc|ccc} 1 & 0 & x & x & x \\ 0 & 1 & x & x & x \\ \hline 0 & 0 & x & 0 & 0 \\ 0 & 0 & x & x & 0 \\ 0 & 0 & x & x & x \end{array}\right) \mathbf{\Theta}_{i,(1,3)}\mathbf{\Theta}_{i,(2,3)} = \left(\begin{array}{cc|ccc} * & 0 & x & x & 0 \\ * & 1 & x & x & * \\ \hline 0 & 0 & x & 0 & 0 \\ 0 & 0 & x & x & 0 \\ * & 0 & x & x & * \end{array}\right) \mathbf{\Theta}_{i,(2,3)}$$

$$= \left(\begin{array}{cc|ccc} * & 0 & x & x & 0 \\ * & * & x & x & 0 \\ \hline 0 & 0 & x & 0 & 0 \\ 0 & 0 & x & x & 0 \\ * & * & x & x & * \end{array}\right)$$

donde $\mathbf{\Theta}_{i,(r,c)}$ es la rotación de Givens que obtiene un cero en la entrada $(r, c+q)$-ésima de la matriz $\mathbf{E}_i$ (la entrada (r,c)-ésima de la submatriz $\mathbf{H}_i\mathbf{P}_i^{1/2}$) acumulando la rotación en la entrada diagonal r-ésima de $\mathbf{E}_i$, con $r = 1, \ldots, q$.

Podemos obtener zeros en la penúltima columna de de $\mathbf{H}_i\mathbf{P}_i^{1/2}$ de manera análoga:

$$\left(\begin{array}{cc|ccc} * & 0 & x & x & 0 \\ * & * & x & x & 0 \\ \hline 0 & 0 & x & 0 & 0 \\ 0 & 0 & x & x & 0 \\ * & * & x & x & * \end{array}\right) \boldsymbol{\Theta}_{i,(1,2)}\boldsymbol{\Theta}_{i,(2,2)} = \left(\begin{array}{cc|ccc} * & 0 & x & 0 & 0 \\ * & * & x & 0 & 0 \\ \hline 0 & 0 & x & 0 & 0 \\ * & * & x & * & 0 \\ * & * & x & * & * \end{array}\right)$$

y finalmente

$$\left(\begin{array}{cc|ccc} * & 0 & x & 0 & 0 \\ * & * & x & 0 & 0 \\ \hline 0 & 0 & x & 0 & 0 \\ * & * & x & * & 0 \\ * & * & x & * & * \end{array}\right) \boldsymbol{\Theta}_{i,(1,1)}\boldsymbol{\Theta}_{i,(2,1)} = \left(\begin{array}{cc|ccc} * & 0 & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 \\ \hline * & * & * & 0 & 0 \\ * & * & * & * & 0 \\ * & * & * & * & * \end{array}\right)$$

$$= \left(\begin{array}{c|c} \mathbf{R}_{e,i}^{1/2} & \mathbf{0} \\ \hline \overline{\mathbf{K}}_{p,i} & \mathbf{P}_{i+1}^{1/2} \end{array}\right)$$

entonces la transformación ortogonal $\boldsymbol{\Theta}_i$ de (4.1) resulta ser:

$$\boldsymbol{\Theta}_i = \boldsymbol{\Theta}_{i,(1,3)}\boldsymbol{\Theta}_{i,(2,3)}\boldsymbol{\Theta}_{i,(1,2)}\boldsymbol{\Theta}_{i,(2,2)}\boldsymbol{\Theta}_{i,(1,1)}\boldsymbol{\Theta}_{i,(2,1)}$$

y en general:

$$\boldsymbol{\Theta}_i = \prod_{c=n:-1:1} \left(\prod_{r=1:q} \boldsymbol{\Theta}_{i,(r,c)} \right)$$

Hemos explotado la estructura de las matrices implicadas en la iteración de la siguiente forma: sólo las columnas r y $c+q$ están implicadas en la aplicación de la rotación $\boldsymbol{\Theta}_{i,(r,c)}$ y sólo los elementos de estas columnas con índices que van desde r hasta q y desde $c+q$ hasta $q+n$ son modificados ($q-r+1+q+n-c-q+1 = q+n-c+2-r$ elementos en

total). Un detalle de interés, que también será relevante en el algoritmo paralelo, es cómo la submatriz inferior izquierda de $\mathbf{E}_i$, inicialmente nula, es llenada de abajo hacia arriba conforme las columnas de $\mathbf{H}_i\mathbf{P}_i^{1/2}$ van siendo anuladas de derecha a izquierda, en la misma proporción. Observemos que si la anulación de columnas se hubiera realizado de izquierda a derecha, con la anulación de la primera habríamos provocado un *llenado* de la submatriz inferior izquierda de $\mathbf{E}_i$, lo que habría conllevado un mayor coste en la ejecución global del algoritmo. El almacenamiento por columnas en Fortran es el que implica un menor tiempo de acceso a los datos. El algoritmo 2 muestra el pseudocódigo.

Algoritmo 2 RLS secuencial: algoritmo raíz cuadrada del Filtro de Kalman o de covarianza

Entrada: $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$, $\mathbf{y} = (\mathbf{y}_0^*, \mathbf{y}_1^*, \ldots)^*$, λ, ϵ
Salida: Actualización de $\mathbf{x}_{\mathrm{LS}}$ cada iteración i-ésima

1: $\hat{\mathbf{x}}_0 = \mathbf{0}$
2: $\mathbf{P}_0^{1/2} = \epsilon^{1/2}\mathbf{I}_n$
3: **para** $i = 0, \ldots$ **hacer**
4: $\quad \mathbf{E}_i \leftarrow \begin{pmatrix} \mathbf{I}_q & \mathbf{H}_i\mathbf{P}_i^{1/2} \\ \mathbf{0} & \lambda^{-1/2}\mathbf{P}_i^{1/2} \end{pmatrix}$
5: $\quad$ **para** $c = n : -1 : 1$ **hacer**
6: $\quad\quad$ **para** $r = 1 : q$ **hacer**
7: $\quad\quad\quad$ Calcular $\mathbf{\Theta}_{i,(r,c)}$
8: $\quad\quad\quad$ Aplicar $\mathbf{\Theta}_{i,(r,c)}$ to $\mathbf{E}_i([r : q, q + c : q + n], [r, q + c])$
9: $\quad\quad$ **fin para**
10: $\quad$ **fin para**
11: $\quad \begin{pmatrix} \mathbf{R}_{e,i}^{1/2} \\ \overline{\mathbf{K}}_{p,i} \end{pmatrix} \leftarrow \mathbf{E}_i([1 : q + n, 1 : q])$
12: $\quad \mathbf{P}_{i+1}^{1/2} \leftarrow \mathbf{E}_i([q + 1 : q + n], [q + 1 : q + n])$
13: $\quad$ Actualizar la estimación del estado

$$\hat{\mathbf{x}}_{i+1} \leftarrow \lambda^{-1/2}\hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i}\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i\right)$$

14: $\quad$ Actualizar la solución LS

$$\mathbf{x}_{\mathrm{LS}_i} \leftarrow \left(\lambda^{1/2}\right)^{i+1}\hat{\mathbf{x}}_{i+1}$$

15: **fin para**

4.1.1. Coste aritmético

El desglose del coste aritmético de las operaciones es el siguiente:

- En la línea 4 del pseudocódigo nos encontramos con dos operaciones:

  - La multiplicación matricial $\mathbf{H}_i\mathbf{P}_i^{1/2}$, donde $\mathbf{H}_i \in \mathbb{C}^{q\times n}$ y $\mathbf{P}_i^{1/2} \in \mathbb{C}^{n\times n}$ es triangular inferior. El coste será

  $$qn^2 \quad \text{flops} \tag{4.2}$$

  - El escalado (si $\lambda \neq 1$) $\lambda^{-1/2}\mathbf{P}_i^{1/2}$, habida cuenta de la estructura triangular inferior de $\mathbf{P}_i^{1/2}$ conllevará un coste de

  $$\frac{1}{2}n^2 + \frac{1}{2}n$$

  Por lo que el coste total de la línea 4, si $\lambda \neq 1$, es

  $$qn^2 + \left(\frac{1}{2}n^2 + \frac{1}{2}n\right) \quad \text{flops}$$

- Denotemos por w_{CG} el coste del cálculo de una rotación de Givens (línea 7).

- La aplicación de la rotación de Givens (línea 8) se realiza sobre un par de vectores con un total de $q - r + n - c + 2$ elementos. Denotemos por w_{AG} el coste de aplicar una rotación de Givens a un par de elementos, entonces la aplicación a todos los elementos necesarios conllevará un coste de

  $$w_{AG}(q - r + n - c + 2) \quad \text{flops}$$

- El bucle que comienza en la línea 6 implica un coste de:

$$\sum_{r=1}^{q} w_{CG} + w_{AG}(q - r + n - c + 2) \quad = \tag{4.3}$$

$$w_{CG}q + w_{AG}(q + n - c + 2)q - w_{AG}\sum_{r=1}^{q} r \quad =$$

$$w_{CG}q + w_{AG}\left(\frac{1}{2}q + n - c + \frac{3}{2}\right)q \quad \text{flops} \tag{4.4}$$

- El bucle que comienza en la línea 5 conlleva un coste de:

$$\sum_{c=1}^{n} w_{CG}q + w_{AG}\left(\frac{1}{2}q + n - c + \frac{3}{2}\right)q \quad =$$

$$w_{CG}qn + w_{AG}\left(\frac{1}{2}q + n + \frac{3}{2}\right)qn - w_{AG}q\sum_{c=1}^{n} c \quad =$$

$$w_{CG}qn + w_{AG}\left(\frac{1}{2}q + \frac{1}{2}n + 1\right)qn \quad \text{flops} \tag{4.5}$$

- El coste de la actualización del estado proviene de:

  - $\mathbf{H}_i\hat{\mathbf{x}}_i$: $q(2n-1)$ flops
  - $\mathbf{y}_i - (\mathbf{H}_i\hat{\mathbf{x}}_i)$: q flops
  - $\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i\right)$: q^2 flops —$\mathbf{R}_{e,i}^{-1/2}$, triangular inferior—
  - $\mathbf{K}_{p,i}\left[\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i\right)\right]$: $n(2q-1)$ flops
  - $\lambda^{-1/2}\hat{\mathbf{x}}_i$: n flops, si $\lambda \neq 1$
  - $\hat{\mathbf{x}}_{i+1} = \lambda^{-1/2}\hat{\mathbf{x}}_i + \mathbf{K}_{p,i}\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i\right)$: n flops
  - $\mathbf{x}_{\mathrm{LS}_i} = \left(\lambda^{1/2}\right)^{i+1}\hat{\mathbf{x}}_{i+1}$: $n+1$ flops, si $\lambda \neq 1$

  Por lo que el coste es

$$4qn + q^2 \quad \text{flops} \tag{4.6}$$

  más $2n+1$ flops si $\lambda \neq 1$.

Definitivamente, el coste aproximado de una iteración del algoritmo será:

$$W_{\mathrm{sec},i}(z)|_{\lambda=1} = qn^2 + w_{CG}qn + w_{AG}\left(\frac{1}{2}q + \frac{1}{2}n + 1\right)qn + (4qn + q^2) \quad \text{flops} \tag{4.7}$$

ó

$$\begin{aligned} W_{\mathrm{sec},i}(z)|_{\lambda\neq 1} &= qn^2 + w_{CG}qn + \left[\frac{1}{2}n^2 + \frac{1}{2}n\right] + w_{AG}\left(\frac{1}{2}q + \frac{1}{2}n + 1\right)qn \\ &\quad +(4qn + q^2) + [2n + 1] \quad \text{flops} \end{aligned} \tag{4.8}$$

El número total de iteraciones es m/q, luego el coste del algoritmo es

$$\begin{aligned} W_{\mathrm{sec}}(z)|_{\lambda=1} &= \sum_{i=0}^{m/q-1} W_{\mathrm{sec},i}(z) \\ &= \sum_{i=0}^{m/q-1} qn^2 + w_{CG}qn + w_{AG}\left(\frac{1}{2}q + \frac{1}{2}n + 1\right)qn + (4qn + q^2) \\ &= n^2m + w_{CG}nm + w_{AG}\left(\frac{1}{2}q + \frac{1}{2}n + 1\right)nm + 4mn + qm \\ &= \left(1 + w_{AG}\frac{1}{2}\right)n^2m + \left[w_{CG} + w_{AG}\left(\frac{1}{2}q + 1\right) + 4\right]nm + qm \\ &\approx \left(1 + w_{AG}\frac{1}{2}\right)n^2m \quad \text{flops} \end{aligned} \tag{4.9}$$

ó

$$\begin{aligned} W_{\mathrm{sec}}(z)|_{\lambda\neq 1} &= W_{\mathrm{sec}}(z)|_{\lambda=1} + \left\{\left[\frac{1}{2}n^2 + \frac{1}{2}n\right] + [2n + 1]\right\}\frac{m}{q} \\ &\approx \left(1 + \frac{1}{2q} + w_{AG}\frac{1}{2}\right)n^2m \quad \text{flops} \end{aligned} \tag{4.10}$$

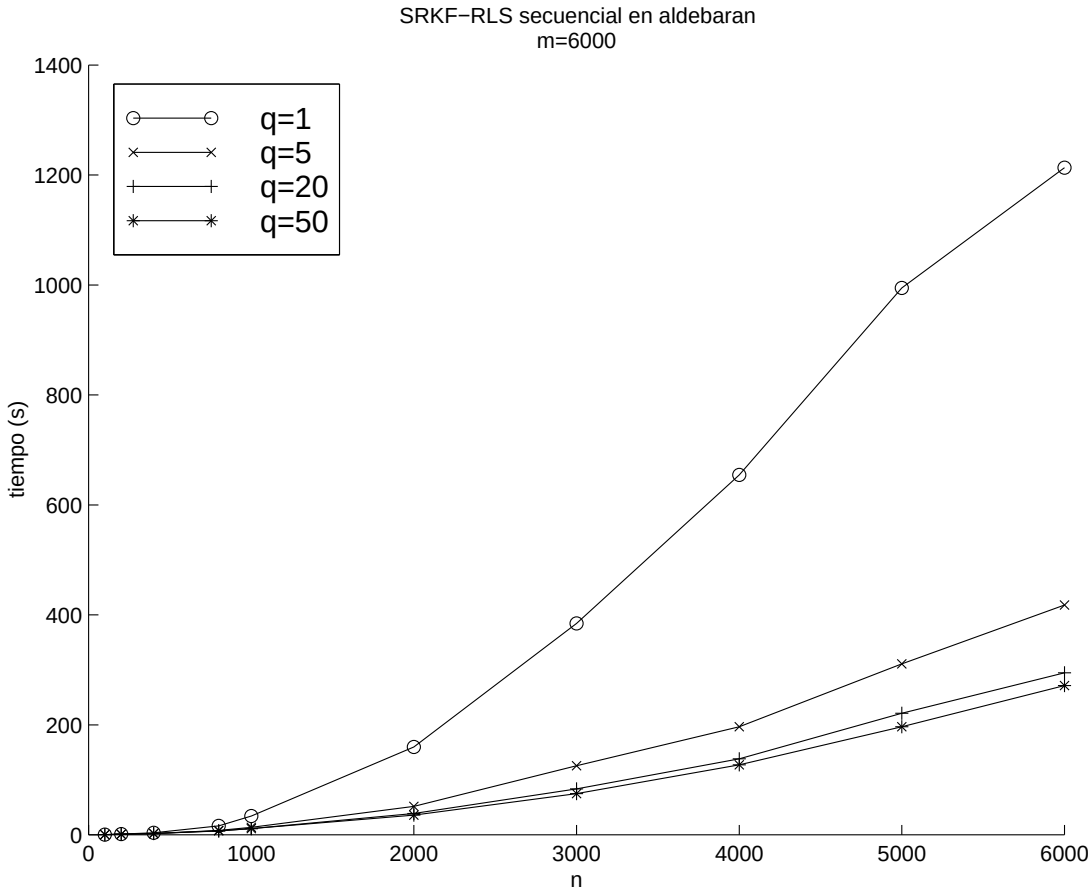

Figura 4.2: Tiempos de ejecución secuencial para el algoritmo SRKF-RLS

El algoritmo tiene una complejidad, por tanto, de $\mathbf{\Theta}(n^2 m)$ flops.

4.1.2. Resultados experimentales

La figura 4.2 muestra los tiempos de ejecución del algoritmo secuencial para varios valores de q, siendo $\lambda = 1$.

Se observa un comportamiento asintótico en las curvas conforme crece el valor del parámetro q. Esto no concuerda con los resultados de (4.9), por lo que debe existir alguna diferencia sustancial entre el modelo de tiempo de ejecución y la realidad.

Para detallar un poco más el comportamiento del algoritmo, reharemos el modelo de tiempo de ejecución basándonos en llamadas, y por simplicidad en la exposición, supondremos que $\lambda = 1$. Las operaciones que más coste aportan son una multiplicación matricial en

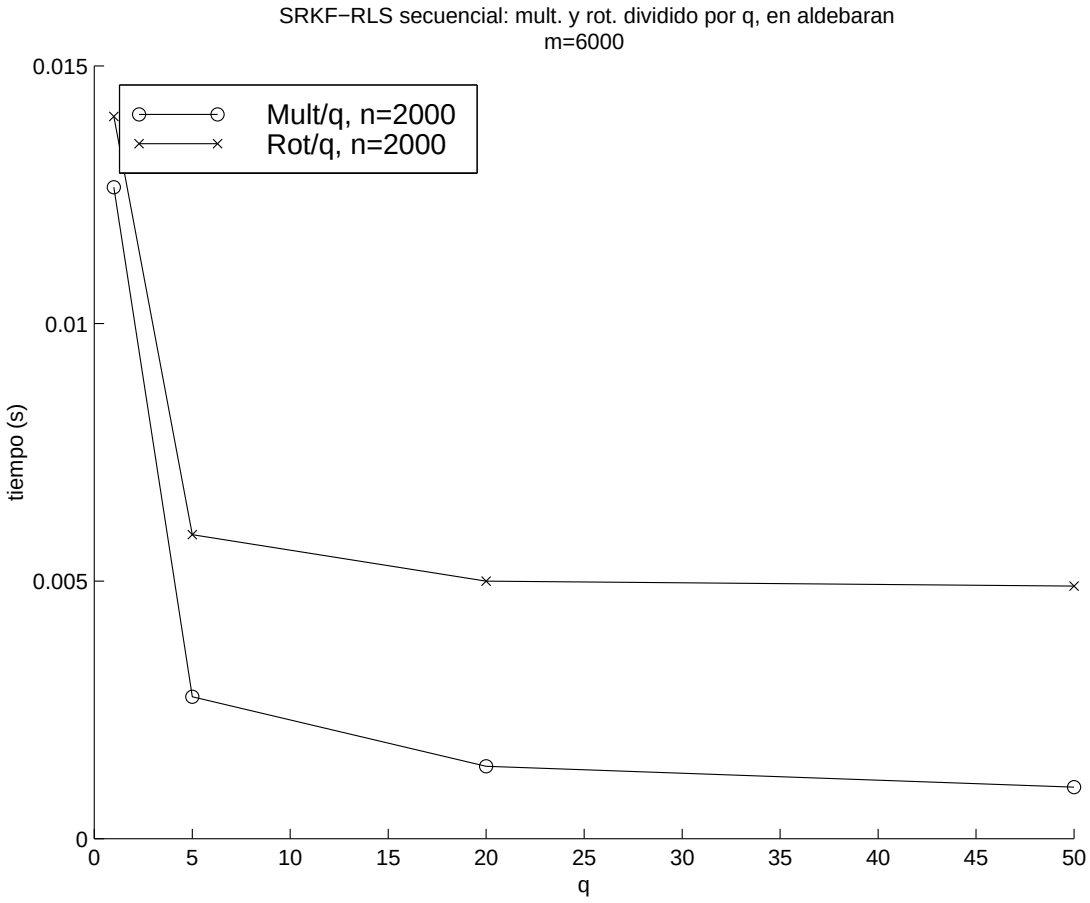

Figura 4.3: Tiempos de multiplicación y de aplicación de rotaciones divididos por q, por iteración, para $n = 2000$.

la que una de las matrices es triangular, empleando para ello la subrutina `DTRMM` de `BLAS`, junto con una serie de aplicaciones de rotaciones de Givens, empleando `DROT` también de `BLAS`. La carga de ambas operaciones la denotaremos por $\mathrm{w}_{\mathrm{DTRMM}}(f,c)$ —donde f y c son las dimensiones de las matrices implicadas— y $\mathrm{w}_{\mathrm{DROT}}(f)$ —donde f es la cantidad de elementos a los que se aplica la rotación—. Así pues, la expresión alternativa del coste es:

$$\begin{aligned}
W_{\mathrm{sec}}(z) &\approx \sum_{i=0}^{m/q-1} \left\{ \mathrm{w}_{\mathrm{DTRMM}}(q,n) + \right. \\
&\qquad \left. \sum_{c=1}^{n}\sum_{r=1}^{q} \left[\mathrm{w}_{\mathrm{DROT}}(q-r+1) + \mathrm{w}_{\mathrm{DROT}}(n-c+1)\right] \right\} \\
&\approx \sum_{i=0}^{m/q-1} \left\{ \mathrm{w}_{\mathrm{DTRMM}}(q,n) + \right. \\
&\qquad \left. \sum_{c=1}^{n} \left[q\mathrm{w}_{\mathrm{DROT}}(n-c+1) + \sum_{r=1}^{q} \mathrm{w}_{\mathrm{DROT}}(r) \right] \right\} \\
&\approx \sum_{i=0}^{m/q-1} \left\{ \mathrm{w}_{\mathrm{DTRMM}}(q,n) + n\sum_{r=1}^{q} \mathrm{w}_{\mathrm{DROT}}(r) + q\sum_{c=1}^{n} \mathrm{w}_{\mathrm{DROT}}(c) \right\} \\
&\approx \frac{m}{q}\mathrm{w}_{\mathrm{DTRMM}}(q,n) + \frac{m}{q}q\sum_{c=1}^{n} \mathrm{w}_{\mathrm{DROT}}(c)
\end{aligned}$$

En la figura 4.3 se representa el tiempo medio por iteración de estas dos operaciones, dividido por q, es decir:

$$\begin{aligned}
\frac{\frac{m}{q}\mathrm{w}_{\mathrm{DTRMM}}(q,n)}{\frac{m}{q}}t_w}{q} &= \frac{1}{q}\mathrm{w}_{\mathrm{DTRMM}}(q,n)t_w \\
\frac{\frac{\frac{m}{q}q\sum_{c=1}^{n}\mathrm{w}_{\mathrm{DROT}}(c)}{\frac{m}{q}}t_w}{q} &= \sum_{c=1}^{n}\mathrm{w}_{\mathrm{DROT}}(c)t_w
\end{aligned}$$

El coste de la multiplicación (4.2) era teóricamente $\mathrm{w}_{\mathrm{DTRMM}}(q,n) = qn^2$, que dividido por q debería dar una constante con q. El coste de la aplicación de las rotaciones era (4.5) que podemos aproximar por $\sum_{c=1}^{n}\mathrm{w}_{\mathrm{DROT}}(c) = w_{AG}\frac{1}{2}n^2q$ flops, que de nuevo dividido por

q debe darnos de nuevo una constante con q.

La figura 4.3 muestra que estos resultados no son constantes con q, siendo este efecto más acusado para valores pequeños de q. Podemos suponer que t_w, el tiempo de ejecución de operación elemental, es el término que *distorsiona* los resultados, ya que éste debe de depender de q de forma monótona decreciente en el intervalo de valores de q considerado, para que las expresiones teóricas coincidan con las experimentales. La hipótesis más verosímil que justifica este comportamiento es la presencia de la memoria *cache* en el sistema. Para el caso de la aplicación de las rotaciones, a mayor valor de q, menor cantidad de iteraciones se llevan a cabo, y más veces se reutiliza las columnas de $\mathbf{P}_i^{1/2}$ ó $\mathbf{P}_{i+1}^{1/2}$ y $\overline{\mathbf{K}}_{p,i}$ sobre las que se aplican dichas rotaciones, lo cual implica una menor tasa de *pérdidas de cache* o *cache miss*; en cuanto a la multiplicación, las consideraciones son análogas. De esta conclusión, se puede deducir que debe de existir un valor óptimo para el parámetro q; sólo en el caso de que t_w decrezca monótonamente con q, para cualquier valor de q, dicho valor óptimo sería $q = m$, es decir, la solución óptima en tiempo de ejecución dejaría de ser la recursiva. Para la determinación de dicho valor óptimo se requiere bien un modelo de tiempo de ejecución muy preciso, a partir del cual se obtenga el mínimo, bien la aplicación de métodos heurísticos. En cualquier caso, el valor de este mínimo estará íntimamente ligado con el tipo concreto de máquina.

Por último, la solución LS final ha sido comparada con la proporcionada por la rutina `DGELSY` de `LAPACK`, proporcionando resultados con el mismo orden de magnitud en la precisión de los mismos.

4.2 Algoritmo paralelo

A continuación se exponen las ideas generales que nos han conducido hacia el diseño del algoritmo paralelo definitivo. Cabe comentar que hemos enfocado el esfuerzo de la paralelización en la parte del algoritmo que supone una mayor complejidad computacional, es decir, el código comprendido entre las líneas 4 y 9 del algoritmo 2. En primer lugar,

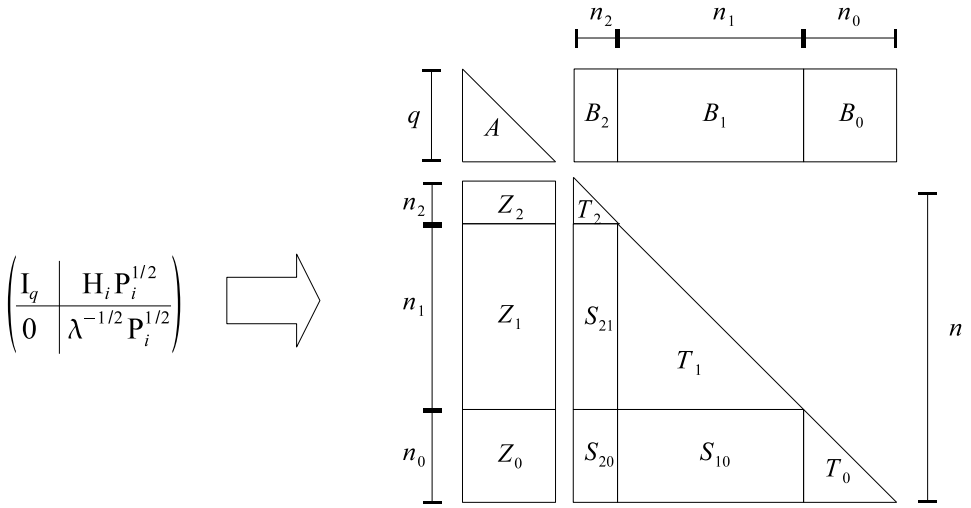

Figura 4.4: Descomposición de los datos para el algoritmo paralelo SRKF.

estableceremos una descomposición de los datos en submatrices, estableciendo las operaciones o tareas que requiere cada una de ellas así como sus dependencias. Posteriormente realizaremos un *mapeo* de estas tareas en procesadores, deduciendo que la mejor traslación consiste en segmentar las iteraciones del algoritmo.

4.2.1. Descomposición de los datos, tareas y dependencias

En la figura 4.4 se muestra una posible descomposición de los datos que para un caso particular podemos plantear en una iteración i-ésima cualquiera del algoritmo. La figura 4.5 muestra la dependencia entre las tareas que requiere la manipulación de dichos datos (las co-dependencias han sido indicadas con un arco bidireccional discontinuo grueso y las dependencias con un arco unidireccional continuo fino).

Los distintos tipos de particiones, dependencias y co-dependencias planteadas en la figura 4.5 son las siguientes:

- Co-dependencias de tipo A y B_j: la submatriz de tipo A es triangular inferior de dimensiones $q \times q$ y la de tipo B_j de tipo rectangular de dimensiones $q \times n_j$. Las operaciones a realizar son formar primeramente la submatriz B_j y luego anularla, actualizando consecuentemente A:

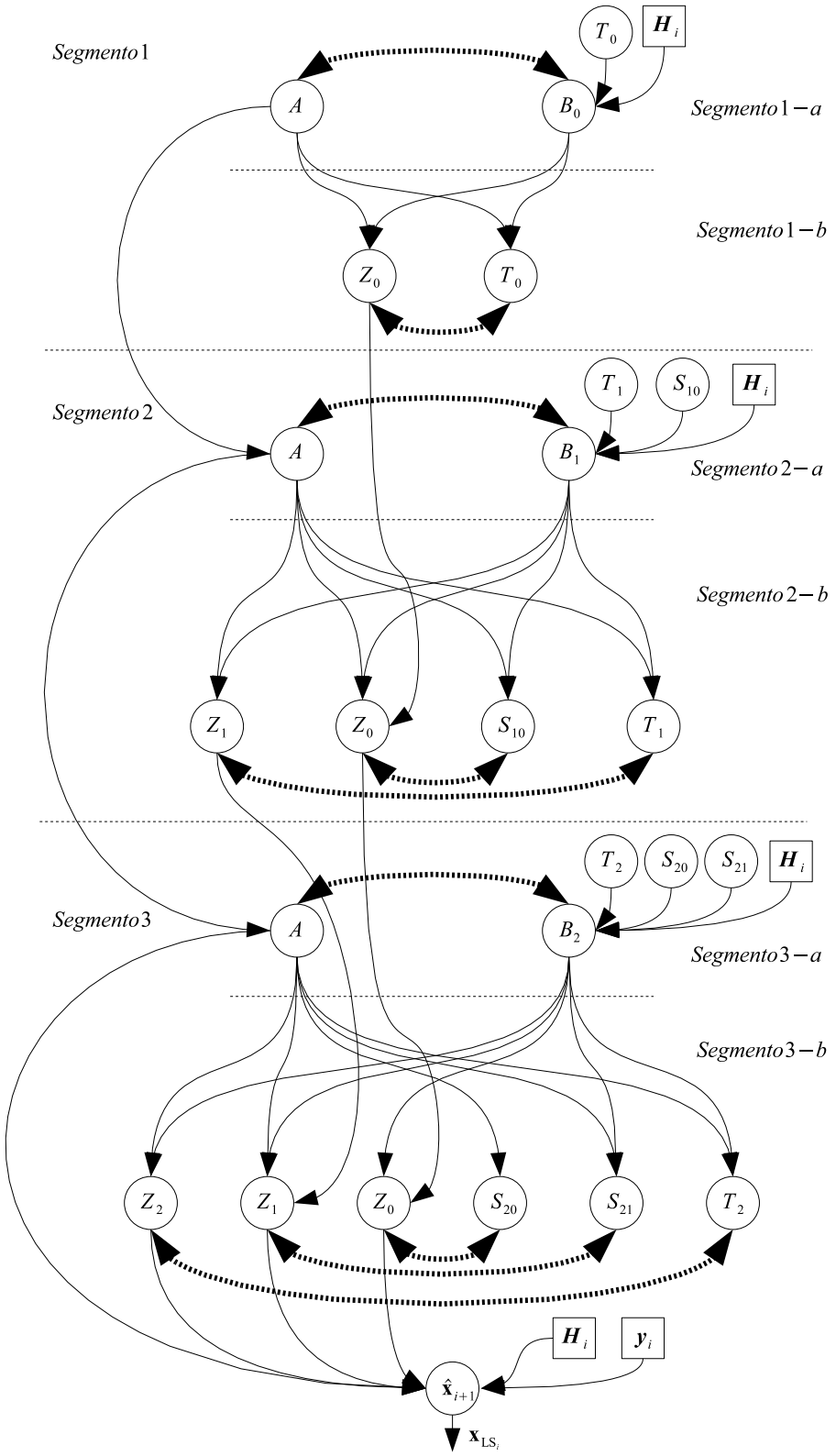

Figura 4.5: Dependencias entre las tareas del algoritmo paralelo SRKF.

  - Creación de B_j: esta submatriz tendrá las columnas oportunas de la matriz $\mathbf{H}_i\mathbf{P}_i^{1/2}$, por lo que necesitará toda la matriz $\mathbf{H}_i$ y sólo las columnas correspondientes de $\mathbf{P}_i^{1/2}$ para llegar a formarla; dichas columnas se hallan en las submatrices T_j y S_{jk} que se hallan en su vertical.
  - Se debe anular las n_j columnas de B_j, pudiéndolo hacer aplicando rotaciones de Givens a sendas columnas de las submatrices A y B_j

- Co-dependencias de tipo Z_j y T_j: inicialmente la submatriz Z_j es nula, por lo que es necesario actualizarla cuando co-dependa de otra submatriz T_j (triangular inferior). La información necesaria para dicha actualización son las rotaciones de Givens que han sido generadas en la co-dependencia entre las submatrices A y B_j, por ello se supondrá que las dimensiones son $n_j \times n_j$ para T_j y consecuentemente $n_j \times q$ para Z_j. La eliminación de cualquier elemento de la columna $c = n_j, \ldots, 1$ de B_j implica la aplicación de la rotación de Givens a un par de vectores de las submatrices Z_j y T_j respectivamente.

- Co-dependencias de tipo Z_j y S_{kj}: las submatrices Z_j dejan de ser nulas cuando han co-dependido de una submatriz T_j. A partir de ese momento, sólo tendrán co-dependencia con submatrices de tipo S_{kj}, con $k > j$. Para que encajen las dimensiones, S_{kj} debe ser una submatriz $n_j \times n_k$, ya que dependerá *verticalmente* de una submatriz B_k y co-dependerá *horizontalmente* de una Z_j. La eliminación de cualquier elemento de la columna $c = n_k, \ldots, 1$ de B_k implica la aplicación de la rotación de Givens a un par de vectores de las submatrices Z_j y S_{kj} respectivamente.

A partir de este planteamiento podemos concluir que:

- La finalización de una iteración del algoritmo requerirá el trabajo sobre tantos segmentos de iteración como submatrices B existan.

- Existirán tantas submatrices Z y T como B existan.

- El grado de concurrencia dependerá del segmento de la iteración. Las tareas que pueden abordarse concurrentemente se corresponden con aquéllas que liberan las co-dependencias $A - B_{j+1}$, $Z_j - T_j$ y las $Z_0 - E_{j0}$, ..., $Z_{j-1} - E_{j,j-1}$. En general, si tenemos t submatrices B, en cualquier iteración i-ésima tendremos t segmentos, habiendo $t + 2$ tareas secuenciales imbricadas en la secuencia de segmentos, donde puede existir la siguiente cantidad de tareas paralelas: 1, 2, 3,..., t, t, 1, tal y como puede observarse en la gráfica del ejemplo en la figura 4.5 (el último valor 1 de la sucesión de tareas paralelas se refiere a la extracción de la solución; el penúltimo valor, t al del segmento $3 - b$ del ejemplo y el antepenúltimo valor, también t, a la co-dependencia A-B_2 simultaneada con las del segmento $2 - b$).

Debido al comportamiento del grado de concurrencia obtenido a partir del grafo de dependencias, estimamos que la idea de paralelizar cada iteración del algoritmo secuencial no dará buenos resultados de eficiencia, por lo que optamos por una estrategia segmentada en la paralelización, que se detalla en el siguiente subapartado.

4.2.2. Segmentación de las iteraciones

El algoritmo secuencial, a priori, es difícilmente paralelizable con un buen rendimiento debido a su secuencialidad intrínseca, tal y como se ha podido observar en el subapartado anterior. Una posible solución es, en vez de paralelizar cada iteración, segmentar las operaciones secuenciales de mayor complejidad algorítmica en cada iteración, es decir, establecer tantos segmentos como procesadores, y asignar las labores propias de cada segmento a un procesador. La información a transferir de procesador a procesador son la submatriz A y ciertas submatrices Z, tal y como se puede observar en la figura 4.5.

Supondremos que existen p procesadores en el sistema, denotando P_j cualquier procesador desde P_0 hasta P_{p-1}. Una matriz encerrada entre corchetes con un procesador como subíndice denotará que parte de dicha matriz está en dicho procesador, $[\mathbf{X}]_{P_j}$ o ha sido *ocupada* por dicho procesador. Una matriz encerrada entre paréntesis con un procesador

como subíndice denotará que toda ella reside en dicho procesador, $(\mathbf{X})_{P_j}$, o bien, que ha sido ocupada por el mismo.

Por simplicidad en la exposición, supondremos que existen $p = 3$ procesadores en el sistema: P_0, P_1 y P_2. Supongamos la siguiente descomposición de los datos:

- Habida cuenta de la fuerte relación existente entre las submatrices A y Z, las uniremos todas en una única submatriz. Algebraicamente, esta nueva submatriz constará de las primeras q columnas de $\mathbf{E}_i$, denotadas por $(\mathbf{D}_i)_{P_j}$, y serán procesadas por todos los procesadores de forma segmentada (de manera que el procesador subíndice irá cambiando acorde con el proceso) para obtener las primeras q columnas de la matriz de la parte derecha de la igualdad (4.1). En cualquier instante y en cualquier procesador, la estructura de $(\mathbf{D}_i)_{P_j}$ será

$$(\mathbf{D}_i)_{P_j} = \begin{pmatrix} \mathbf{L}_i \\ \mathbf{M}_i \end{pmatrix}$$

con $\mathbf{L}_i \in \mathbb{C}^{q\times q}$ una matriz triangular inferior y $\mathbf{M}_i \in \mathbb{C}^{n\times q}$. $\mathbf{M}_i$ inicialmente será una matriz nula que se irá llenando de abajo hacia arriba conforme vaya siendo procesada segmentadamente por todos los procesadores. Por ejemplo, al principio de la iteración i-ésima, $\mathbf{D}_i$ residirá íntegramente en el primer procesador P_0:

$$(\mathbf{D}_i)_{P_0} = \begin{pmatrix} \mathbf{L}_i \\ \mathbf{M}_i \end{pmatrix} = \begin{pmatrix} \mathbf{I}_q \\ \mathbf{0} \end{pmatrix}$$

Del mismo modo, al final de la iteración i-ésima, en el último procesador P_{p-1}:

$$(\mathbf{D}_i)_{P_{p-1}} = \begin{pmatrix} \mathbf{L}_i \\ \mathbf{M}_i \end{pmatrix} = \begin{pmatrix} \mathbf{R}_{e,i}^{1/2} \\ \overline{\mathbf{K}}_{p,i} \end{pmatrix}$$

Por simplicidad en la exposición dividiremos la matriz $\mathbf{M}_i$ por filas en p grupos.

Denotaremos con un superíndice a la izquierda la cantidad de filas de la agrupación. Así pues:

$$\mathbf{M}_i = \begin{pmatrix} {}^{n_2}[\mathbf{M}_i] \\ {}^{n_1}[\mathbf{M}_i] \\ {}^{n_0}[\mathbf{M}_i] \end{pmatrix}$$

con $n_0 + n_1 + n_2 = n$. Los distintos valores n_j, $j = 0, \ldots, p-1$, se especificarán posteriormente.

- Las submatrices B, T y S, es decir, las últimas n columnas de $\mathbf{E}_i$ serán denotadas de la siguiente forma:

$$\mathbf{C}_i = \begin{pmatrix} \mathbf{H}_i \mathbf{P}_i^{1/2} \\ \lambda^{-1/2} \mathbf{P}_i^{1/2} \end{pmatrix}$$

Esta submatriz estará distribuidas por columnas entre los procesadores, teniendo cada uno n_j columnas con, evidentemente, $\sum_{j=0}^{p-1} n_j = n$. El valor c_{0_j} denotará el índice de la primera de ellas que reside en el procesador P_j. Este índice puede expresarse como:

$$c_{0_j} = 1 + \sum_{k=j+1}^{t-1} n_k = n + 1 - \sum_{k=j}^{t-1} n_k \tag{4.11}$$

donde t representa el número de submatrices B, que en este caso serán $t = p$.

Esta distribución de datos no cambiará a lo largo de la ejecución del algoritmo paralelo (a no ser que se realizara un equilibrado de carga adaptativo). Es decir, uniremos las submatrices B_j, T_j y S_{jk}, $\forall\, k$, en una única submatriz y ésta será asignada al procesador P_j.

Por lo tanto, realizaremos una descomposición de la matriz problema por columnas entre los procesadores, teniendo la matriz de partida $\mathbf{E}_i$ la siguiente distribución:

$$\mathbf{E}_i = \begin{pmatrix} \mathbf{D}_i & \mathbf{C}_i \end{pmatrix} = \begin{pmatrix} (\mathbf{D}_i)_{P_0} & [\mathbf{C}_i]_{P_2} & [\mathbf{C}_i]_{P_1} & [\mathbf{C}_i]_{P_0} \end{pmatrix}$$

4.2.3. Tareas de los procesadores

Cuando un procesador P_j obtenga ceros en su submatriz $[\mathbf{C}_i]_{P_j}$ en la iteración i-ésima, podemos observar que ya obtiene, en las n últimas filas, $\left[\mathbf{P}_{i+1}^{1/2}\right]_{P_j}$ —véase la ecuación (4.1)—, por lo que si dispone de $\mathbf{H}_{i+1}$, podrá elaborar $[\mathbf{C}_{i+1}]_{P_j}$ y por tanto comenzar de nuevo su trabajo para la iteración $i+1$-ésima. La concepción segmentada del algoritmo se basa en esta idea.

El primer paso en el algoritmo paralelo segmentado sería[1]:

$$\begin{aligned}
\mathbf{E}_i' &= \mathbf{E}_i \boldsymbol{\Theta}_{i,P_0} \\
&= \begin{pmatrix} (\mathbf{D}_i)_{P_0} & [\mathbf{C}_i]_{P_2} & [\mathbf{C}_i]_{P_1} & [\mathbf{C}_i]_{P_0} \end{pmatrix} \boldsymbol{\Theta}_{i,P_0} \\
&= \begin{pmatrix} \begin{pmatrix} \mathbf{I}_q \\ {}^{n_2}[\mathbf{0}] \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_0}[\mathbf{0}] \end{pmatrix}_{P_0} & [\mathbf{C}_i]_{P_2} & [\mathbf{C}_i]_{P_1} & [\mathbf{C}_i]_{P_0} \end{pmatrix} \boldsymbol{\Theta}_{i,P_0} \\
&= \begin{pmatrix} \begin{pmatrix} \mathbf{L}_i \\ {}^{n_2}[\mathbf{0}] \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_0}[\mathbf{M}_i] \end{pmatrix}_{P_0} & [\mathbf{C}_i]_{P_2} & [\mathbf{C}_i]_{P_1} & \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \end{bmatrix}_{P_0} \end{pmatrix}
\end{aligned}$$

A partir de este instante de tiempo, si P_0 ya ha recibido $\mathbf{H}_{i+1}$ (por ejemplo del subsistema de adquisición), entonces tiene todos los datos necesarios para arrancar la iteración $(i+1)$-ésima: debe formar $[\mathbf{C}_{i+1}]_{P_0}$ y comenzar de nuevo. El comportamiento segmentado

[1]El apóstrofo en una variable denotará la sobreescritura de la misma por los cálculos realizados

se basa en esta actuación. En cualquier caso, ahora P_0 debe transferir a P_1 tanto $\mathbf{L}_i$ como ${}^{n_0}[\mathbf{M}_i]$, por lo que la cantidad total de elementos a transferir es $\frac{q(q+1)}{2} + n_0 q$. Una vez en P_1, en éste:

$$
\begin{aligned}
\mathbf{E}_i'' &= \mathbf{E}_i' \mathbf{\Theta}_{i,P_1} \\
&= \left(\begin{pmatrix} \mathbf{L}_i \\ {}^{n_2}[\mathbf{0}] \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_0}[\mathbf{M}_i] \end{pmatrix}_{P_1} \quad [\mathbf{C}_i]_{P_2} \quad [\mathbf{C}_i]_{P_1} \quad \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \end{bmatrix}_{P_0} \right) \mathbf{\Theta}_{i,P_1} \\
&= \left(\begin{pmatrix} \mathbf{L}_i' \\ {}^{n_2}[\mathbf{0}] \\ {}^{n_1}[\mathbf{M}_i] \\ {}^{n_0}[\mathbf{M}_i]' \end{pmatrix}_{P_1} \quad [\mathbf{C}_i]_{P_2} \quad \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \end{bmatrix}_{P_1} \quad \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \end{bmatrix}_{P_0} \right)
\end{aligned}
$$

En este instante, P_1 podría arrancar la segunda etapa segmentada de la iteración $(i+1)$-ésima si ya ha recibido la información necesaria desde P_0 y $\mathbf{H}_{i+1}$ del subsistema de adquisición de datos (comportamiento segmentado de nuevo). En cualquier caso, P_1 debe transferir $\mathbf{L}_i'$, ${}^{n_1}[\mathbf{M}_i]$ y ${}^{n_0}[\mathbf{M}_i]'$ a P_2 —un total de $\frac{q(q+1)}{2} + (n_0+n_1)q$ elementos—. Entonces, en P_2:

$$\mathbf{E}_i''' = \mathbf{E}_i'' \mathbf{\Theta}_{i,P_2}$$

$$
\begin{aligned}
&= \left(\begin{pmatrix} \mathbf{L}_i' \\ {}^{n_2}[\mathbf{0}] \\ {}^{n_1}[\mathbf{M}_i] \\ {}^{n_0}[\mathbf{M}_i]' \end{pmatrix}_{P_2} \quad [\mathbf{C}_i]_{P_2} \quad \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \end{bmatrix}_{P_1} \quad \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \end{bmatrix}_{P_0} \right) \boldsymbol{\Theta}_{i,P_2} \\
&= \left(\begin{pmatrix} \mathbf{L}_i'' \\ {}^{n_2}[\mathbf{M}_i] \\ {}^{n_1}[\mathbf{M}_i]' \\ {}^{n_0}[\mathbf{M}_i]'' \end{pmatrix}_{P_2} \quad \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \end{bmatrix}_{P_2} \quad \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \end{bmatrix}_{P_1} \quad \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \end{bmatrix}_{P_0} \right) = \mathbf{F}_i
\end{aligned}
$$

En este instante:

$$
\begin{pmatrix} \mathbf{R}_{e,i}^{1/2} \\ \overline{\mathbf{K}}_{p,i} \end{pmatrix} = \begin{pmatrix} \mathbf{L}_i \\ \mathbf{M}_i \end{pmatrix} = \begin{pmatrix} \mathbf{L}_i'' \\ \sum_{k=0}^{2} {}^{n_k}[\mathbf{M}_i] \end{pmatrix}_{P_2} = \begin{pmatrix} \mathbf{L}_i'' \\ {}^{n_2}[\mathbf{M}_i] \\ {}^{n_1}[\mathbf{M}_i]' \\ {}^{n_0}[\mathbf{M}_i]'' \end{pmatrix}_{P_2}
$$

por lo que la iteración ya ha finalizado, y la estimación del estado así como la solución mínimos cuadrados puede actualizarse:

$$
\begin{aligned}
\hat{\mathbf{x}}_{i+1} &= \lambda^{-1/2}\hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i}\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i\right) \\
\mathbf{x}_{\mathrm{LS}_i} &= \left(\lambda^{1/2}\right)^{i+1}\hat{\mathbf{x}}_{i+1}
\end{aligned}
$$

De nuevo, a partir de este instante de tiempo, P_2 podría arrancar la tercera etapa segmentada del algoritmo para la iteración $(i+1)$-ésima si ha recibido los datos necesarios de P_1 y del subsistema de adquisición.

La figura 4.6 muestra gráficamente el comportamiento del algoritmo.

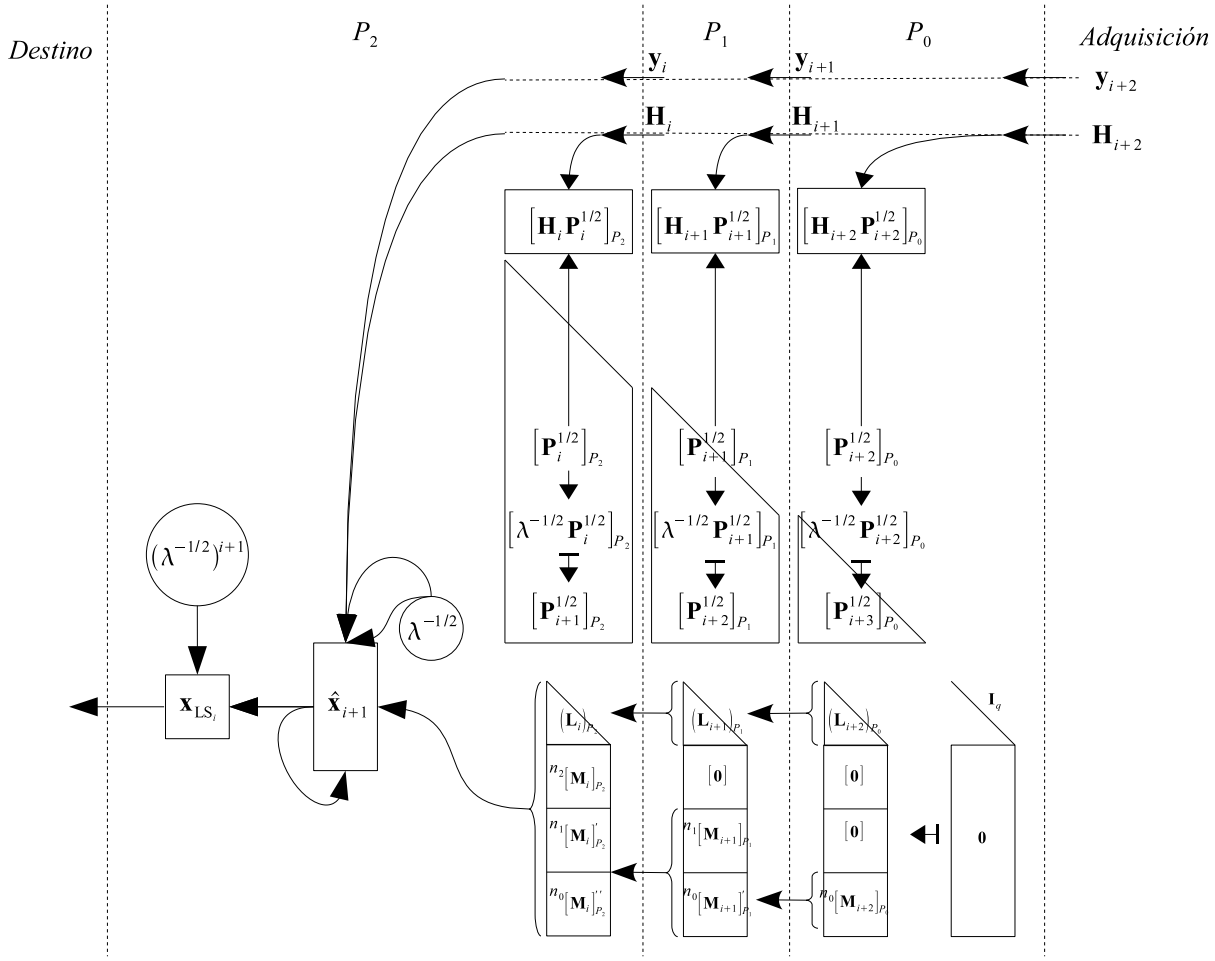

Figura 4.6: Comportamiento del algoritmo SRKF descrito gráficamente

4.2.4. Pseudocódigo

El **Algoritmo 3** muestra el pseudocódigo del algoritmo paralelo expuesto en los subapartados anteriores. Una flecha simple ($\leftarrow$) denota una asignación aritmética y una flecha doble hacia la izquierda o derecha ($\Leftarrow$ / $\Rightarrow$), una transferencia de información de un procesador al siguiente (por simplicidad se supone que los datos son transferidos desde y hacia las posiciones oportunas de las submatrices origen y destino).

4.2.5. Coste aritmético

El coste de cualquier iteración i-ésima del algoritmo paralelo es debido a:

- El coste de la multiplicación matricial de la línea 9 se debe a una multiplicación matricial y a un escalado:

  - Multiplicación $\mathbf{H}_i \left[\mathbf{P}_i^{1/2}\right]_{P_j}$, donde $\mathbf{H}_i \in \mathbb{C}^{q\times n}$ y $\left[\mathbf{P}_i^{1/2}\right]_{P_j} \in \mathbb{C}^{n\times n_j}$, ésta última con la siguiente estructura:

$$\left[\mathbf{P}_i^{1/2}\right]_{P_j} = \begin{pmatrix} \mathbf{0} \\ \mathbf{J} \\ \mathbf{K} \end{pmatrix}$$

    Si c_{0_j} denota el índice de la primera columna de $\mathbf{C}_i$ asignada a P_j, entonces $\mathbf{0} \in \mathbb{C}^{(c_{0_j}-1)\times n_j}$, $\mathbf{J} \in \mathbb{C}^{n_j\times n_j}$ and $\mathbf{K} \in \mathbb{C}^{(n-c_{0_j}-n_j+1)\times n_j}$.

    Expresemos $\mathbf{H}_i$ como:

$$\mathbf{H}_i = \begin{pmatrix} \mathbf{H}_{i_0} & \mathbf{H}_{i_J} & \mathbf{H}_{i_K} \end{pmatrix}$$

    donde $\mathbf{H}_{i_0} \in \mathbb{C}^{q\times(c_{0_j}-1)}$, $\mathbf{H}_{i_J} \in \mathbb{C}^{q\times n_j}$ y $\mathbf{H}_{i_K} \in \mathbb{R}^{q\times(n-c_{0_j}-n_j+1)}$. Entonces, la multiplicación podemos realizarla aprovechando la estructura de $\left[\mathbf{P}_i^{1/2}\right]_{P_j}$:

Algoritmo 3 RLS paralelo: versión raíz cuadrada del Filtro de Kalman o de covarianza

Entrada: $\mathbf{y} = (\mathbf{y}_0^*, \mathbf{y}_1^*, \ldots)^*$ y $\hat{\mathbf{x}}_0$ en P_{p-1}, $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$, $\mathbf{P}_0^{1/2}$
Salida: $\mathbf{x}_{\mathrm{LS}}$ cada iteración i-ésima, en P_{p-1}

1: **para todo** P_j : en P_j **hacer**
2: **para** $i = 0, \ldots$ **hacer**
3: **si** $P_j \neq P_0$ **entonces**
4: $(\mathbf{D}_i)_{P_j} \Leftarrow \begin{pmatrix} \mathbf{L}_i \\ \sum_{k=0}^{j-1} n_k [\mathbf{M}_i] \end{pmatrix}_{P_{j-1}}$
5: **sino**
6: $(\mathbf{D}_i)_{P_j} \leftarrow \begin{pmatrix} \mathbf{I}_q \\ \mathbf{0} \end{pmatrix}$
7: **fin si**
8: Input $\mathbf{H}_i$
9: $[\mathbf{C}_i]_{P_j} \leftarrow \begin{pmatrix} \mathbf{H}_i \\ \lambda^{1/2}\mathbf{I}_n \end{pmatrix} \left[\mathbf{P}_i^{1/2}\right]_{P_j}$
10: **para** $c = n_j : -1 : 1$ **hacer**
11: **para** $r = 1 : q$ **hacer**
12: Calcular $\mathbf{\Theta}_{i,(r,c),P_j}$ a partir de $(\mathbf{D}_i(r,r))_{P_j}$ y $[\mathbf{C}_i(r,c)]_{P_j}$
13: Aplicar $\mathbf{\Theta}_{i,(r,c),P_j}$ a los subvectores $(\mathbf{D}_i(z,r))_{P_j}$ y $[\mathbf{C}_i(z,c)]_{P_j}$ donde z es el vector de índices $z = [r : q, c_{0_j} + q + c - 1 : q + n]$
14: **fin para**
15: **fin para**
16: $\left[\mathbf{P}_{i+1}^{1/2}\right]_{P_j} = [\mathbf{C}_i([q+1:n],[1:n_j])]_{P_j}$
17: **si** $P_j \neq P_{p-1}$ **entonces**
18: $(\mathbf{D}_i)_{P_j} \Rightarrow \begin{pmatrix} \mathbf{L}_i \\ \sum_{k=0}^{j} n_k [\mathbf{M}_i] \end{pmatrix}_{P_{j+1}}$
19: **sino**
20: Input $\mathbf{y}_i$
21: $\begin{pmatrix} \mathbf{R}_{e,i}^{1/2} \\ \overline{\mathbf{K}}_{p,i} \end{pmatrix} \leftarrow \begin{pmatrix} \mathbf{L}_i \\ \mathbf{M}_i \end{pmatrix}_{P_{p-1}}$
22: Actualizar estimación del estado

$$\hat{\mathbf{x}}_{i+1} \leftarrow \lambda^{-1/2}\hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i}\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i\right)$$

23: Actualizar solución LS: $\mathbf{x}_{\mathrm{LS}_i} \leftarrow \left(\lambda^{1/2}\right)^{i+1}\hat{\mathbf{x}}_{i+1}$
24: **fin si**
25: **fin para**
26: **fin para**

$$\begin{aligned}\mathbf{H}_i \left[\mathbf{P}_i^{1/2}\right]_{P_j} &= \begin{pmatrix} \mathbf{H}_{i_0} & \mathbf{H}_{i_J} & \mathbf{H}_{i_K} \end{pmatrix} \begin{pmatrix} \mathbf{0} \\ \mathbf{J} \\ \mathbf{K} \end{pmatrix} \\ &= \mathbf{H}_{i_J}\mathbf{J} + \mathbf{H}_{i_K}\mathbf{K}\end{aligned}$$

La multiplicación $\mathbf{H}_{i_J}\mathbf{J}$ involucra a la submatriz triangular inferior $\mathbf{J}$, por lo que el coste será qn_j^2 flops.

La segunda multiplicación matricial $\mathbf{H}_{i_K}\mathbf{K}$ involucra a matrices generales, luego el coste será:

$$q[2(n - c_{0_j} - n_j + 1) - 1]n_j = 2qn_jn - 2qc_{0_j}n_j - 2qn_j^2 + qn_j \quad \text{flops}$$

Deberemos añadir el coste de la suma de ambas multiplicaciones, es decir qn_j flops, luego el coste total de $\mathbf{H}_i \left[\mathbf{P}_i^{1/2}\right]_{P_j}$ será:

$$\begin{aligned}qn_j^2 + (2qn_jn - 2qc_{0_j}n_j - 2qn_j^2 + qn_j) + qn_j &= \\ 2qn_jn - qn_j^2 - 2qc_{0_j}n_j + 2qn_j \quad \text{flops}\end{aligned}$$

- El escalado de la matriz $\lambda^{-1/2}\left[\mathbf{P}_i^{1/2}\right]_{P_j}$ (si $\lambda \neq 1$). La columna a-ésima, $a = 1, \ldots, n$ de $\mathbf{P}_i^{1/2}$ tiene $n - a + 1$ elementos no nulos, por lo que el coste del escalado seá:

$$\begin{aligned}\sum_{a=c_{0_j}}^{c_{0_j}+n_j-1} n - a + 1 &= n_j(n+1) - \sum_{a=c_{0_j}}^{c_{0_j}+n_j-1} a \\ &= nn_j - \frac{1}{2}n_j^2 + \left(\frac{3}{2} - c_{0_j}\right)n_j \quad \text{flops}\end{aligned}$$

Por lo que el coste total de la formación de $[\mathbf{C}_i]_{P_j}$ es

$$(2q+1)n_j n - \left(q+\frac{1}{2}\right)n_j^2 - \left(2qc_{0_j} - 2q + c_{0_j} - \frac{3}{2}\right)n_j \quad \text{flops}$$

- Denotemos por w_{CG} el coste de calcular una rotaciones de Givens.

- La aportación al coste de la línea 12 se debe a la aplicación de la rotación de Givens calculada al par de subvectores $(\mathbf{D}_i(z,r))_{P_j}$ y $[\mathbf{C}_i(z,c)]_{P_j}$ donde z es el intervalo de índices $z = [r : q, c_{0_j} + q + c - 1 : q + n]$

  La cantidad total de pares de elementos es:

$$q - r + 1 + q + n - (c_{0_j} + q + c - 1) + 1 = q - r + 3 + n - c_{0_j} - c$$

  Luego la aplicación de la rotación de Givens tendrá un coste de

$$w_{AG}(q - r + 3 + n - c_{0_j} - c) \quad \text{flops}$$

- El bucle de la línea 10 conllevará, por tanto, un coste de:

$$\begin{aligned}
\sum_{r=1}^{q} w_{CG} + w_{AG}(q - r + 3 + n - c_{0_j} - c) \quad &= \\
w_{CG}q + w_{AG}(q + 3 + n - c_{0_j} - c)q - w_{AG}\sum_{r=1}^{q} r \quad &= \\
w_{CG}q + w_{AG}\left(\frac{1}{2}q + \frac{5}{2} + n - c_{0_j} - c\right)q \quad &\text{flops}
\end{aligned}$$

- El bucle de la línea 9 implicará un coste de

$$\begin{aligned}
\sum_{c=1}^{n_j} w_{CG}q + w_{AG}\left(\frac{1}{2}q + \frac{5}{2} + n - c_{0_j} - c\right)q &= \\
w_{CG}qn_j + w_{AG}\left(\frac{1}{2}q + \frac{5}{2} + n - c_{0_j}\right)qn_j - w_{AG}q\sum_{c=1}^{n_j} c &= \\
w_{CG}qn_j + w_{AG}\left(\frac{1}{2}q + 2 + n - c_{0_j} - \frac{1}{2}n_j\right)qn_j \quad \text{flops}
\end{aligned}$$

- Sólo en P_{p-1} se realiza la actualización del estado y de la solución LS (líneas 21 y 22), cuyo coste se muestra en (4.6) y es de orden $\Theta(qn)$.

Por lo que el coste aritmético de la iteración paralela i-ésima en el procesador P_j es de:

$$\begin{aligned}
W_{P_j,i}(z)\big|_{\lambda=1} &= \\
2qn_jn - qn_j^2 - 2qc_{0_j}n_j + 2qn_j + & \\
w_{CG}qn_j + w_{AG}\left(\frac{1}{2}q + 2 + n - c_{0_j} - \frac{1}{2}n_j\right)qn_j &= \\
(2q + w_{AG}q)nn_j - \frac{1}{2}(2q + w_{AG}q)n_j^2 + & \\
-\left[(2q + w_{AG}q)c_{0_j} - w_{CG}q - (2q + w_{AG}q) - w_{AG}q - \frac{1}{2}w_{AG}q^2\right]n_j & \quad \text{flops}
\end{aligned} \tag{4.12}$$

O bien

$$
\begin{aligned}
W_{P_j,i}(z)\big|_{\lambda\neq 1} &= \\
W_{P_j,i}(z)\big|_{\lambda=1} + nn_j - \frac{1}{2}n_j^2 + \left(\frac{3}{2} - c_{0_j}\right) n_j &= \\
\left[(2q + w_{AG}q) + 1\right] nn_j - \frac{1}{2}\left[(2q + w_{AG}q) + 1\right] n_j^2 & \\
- \left[(2q + w_{AG}q + 1)c_{0_j} - w_{CG}q - (2q + w_{AG}q + 1) - w_{AG}q - \frac{1}{2}w_{AG}q^2 - \frac{1}{2}\right] n_j & \quad \text{flops}
\end{aligned}
\tag{4.13}
$$

En estos costes, no hemos incluido el coste de la actualización de la solución, ya que implica una pérdida de regularidad en las expresiones, debido a que sólo lo realiza P_{p-1}. Esto implicará, a su vez, cierta sobrecarga aritmética. Al ser de coste $\Theta(qn)$, en principio, no la consideraremos.

4.2.6. Equilibrado de la carga

Hemos paralelizado la parte del algoritmo secuencial comprendida entre las líneas 4 y 10, que son las que aportan el coste de orden superior. Hemos obtenido en el subapartado anterior el coste por iteración en cada procesador, según su cantidad de columnas asignadas, n_j. A continuación deberemos explicitar qué cantidad de columnas n_j debe poseer cada procesador P_j para que todos finalicen la iteración, idealmente, en la misma cantidad de tiempo.

Si reescribimos la condición de equilibrado (2.8) para este algoritmo, utilizando (4.12) y (4.7) obtenemos:

$$
\begin{aligned}
&(2q + w_{AG}q)nn_j - \frac{1}{2}(2q + w_{AG}q)\, n_j^2 + \\
-&\left[(2q + w_{AG}q)c_{0_j} - w_{CG}q - (2q + w_{AG}q) - w_{AG}q - \frac{1}{2}w_{AG}q^2\right] n_j \;\; = \\
&\alpha_j \left[qn^2 + w_{CG}qn + w_{AG}\left(\frac{1}{2}q + \frac{1}{2}n + 1\right)qn\right]
\end{aligned}
\tag{4.14}
$$

O bien, si $\lambda \neq 1$ utilizando (4.13) y (4.8)

$$
\begin{aligned}
&[(2q + w_{AG}q) + 1]\, nn_j - \frac{1}{2}[(2q + w_{AG}q) + 1]\, n_j^2 \\
-&\left[(2q + w_{AG}q + 1)c_{0_j} - w_{CG}q - (2q + w_{AG}q + 1) - w_{AG}q - \frac{1}{2}w_{AG}q^2 - \frac{1}{2}\right] n_j \;\; = \\
&\alpha_j \left[qn^2 + \left(\frac{1}{2}n^2 + \frac{1}{2}n\right) + w_{CG}qn + w_{AG}\left(\frac{1}{2}q + \frac{1}{2}n + 1\right)qn\right]
\end{aligned}
\tag{4.15}
$$

De estas expresiones podemos obtener progresivamente los distintos n_j, comenzando por n_{p-1}, siendo $c_{0_{p-1}} = 1$, y siguiendo hasta n_0, siendo los distintos $c_{0_j} = c_{0_{j+1}} + n_{j+1}$, resolviendo una ecuación de segundo grado. Al tener que redondear o truncar estos valores reales, se provocará cierto desequilibrio residual imposible de eliminar. La forma en la que el factor c_{0_j} influye en el coste, nos indica claramente que, conforme mayor es dicho valor, menor es el coste, es decir, si $n_j = n_k$, $\forall\, k$, entonces los procesadores con mayor índice tendrían un valor inferior para c_{0_j}, por lo que estarían más cargados.

La forma expuesta de realizar el equilibrado se fundamenta en la verosimilitud del modelo de tiempo de ejecución deducido. Por ser una expresión teórica, puede que tenga ciertas discrepancias respecto al resultado empírico, por lo que es probable que los valores de n_j deducidos no sean los óptimos. Además, no hemos tenido en cuenta el tiempo de actualización de la solución ya que es de complejidad $\Theta(qn)$ y supone la pérdida de regularidad en las expresiones.

Línea segmentada bidireccional

Podemos plantear una alternativa que se fundamente sólo en la hipótesis de que la carga aritmética, en cualquier partición o conjuntos de particiones j que vayan a un procesador, pueda modelarse como:

$$W_j(s) = \left[\rho c_{0_j} + f(q,n)\right] n_j + \frac{\beta}{2} n_j^2,\ \forall\, 0 \leq j \leq p-1 \tag{4.16}$$

donde

- β y ρ son constantes
- $f(n,q)$ es cierta función polinómica
- $\sum_{j=0}^{t-1} W_j(z) = W_{\text{sec}}(z)$
- $c_{0_j} + c_{0'_j} = f_1(n) - \frac{\beta}{\rho} n_j$

Podemos comprobar cómo el modelo analítico deducido cumple de manera bastante aproximada estas condiciones, con el matiz de la actualización del estado la solución LS.

El método alternativo se basa en asignar a un mismo procesador dos particiones de manera simétrica o especular. Supongamos que hemos realizado t particiones. Dos particiones, representadas por su carga aritmética, $W_j(z)$ y $W_{j'}(z)$, se considerarán especulares o simétricas cuando sus índices cumplan la relación

$$j = t - j' - 1, \quad 0 \leq j, j' \leq t-1 \tag{4.17}$$

Impondremos como condición adicional que $n_j = n_{j'}$, de modo que

$$\sum_{k=0}^{t-1} n_k = n, \quad \text{y} \quad \sum_{k=0}^{t/2-1} n_k = \frac{n}{2}$$

entonces la carga total de la unión de las particiones j y j' será:

$$\begin{aligned} W_{jj'}(z) &= W_j(z) + W_{j'}(z) \\ &= \left[\rho(c_{0_j} + c_{0_{j'}}) + 2f(n,q)\right] n_j + \beta n_j^2 \\ &= 2f(q,n)n_j + \rho\left(c_{0_j} + c_{0_{j'}}\right) n_j + \beta n_j^2 \\ &= 2f(q,n)n_j + \rho\left(f_1(n) - \frac{\beta}{\rho} n_j\right) n_j + \beta n_j^2 \\ &= 2f(q,n)n_j + \rho f_1(n) n_j - \beta n_j^2 + \beta n_j^2 \\ &= (2f(q,n) + \rho f_1(n))\, n_j \end{aligned}$$

la cual no depende de c_{0_j} ni de $c_{0'_j}$, siendo, además, proporcional a n_j. Definitivamente

$$W_{P_j}(z) = W_{jj'}(z)$$

Por lo que aplicando (2.8):

$$\begin{aligned} W_{P_j}(z) &= \alpha_j W_{\text{sec}}(z) \\ W_{jj'}(z) &= \alpha_j \sum_{k=0}^{t/2-1} W_{P_k}(z) \\ &= \alpha_j \sum_{k=0}^{t/2-1} W_{kk'}(z) \\ 2f(q,n)n_j + \rho f_1(n) n_j &= \alpha_j \sum_{k=0}^{t/2-1} 2f(q,n)n_k + \rho f_1(n) n_k \\ &= \alpha_j (2f(q,n) + \rho f_1(n)) \sum_{k=0}^{t/2-1} n_k \\ n_j &= \alpha_j \frac{n}{2} \end{aligned}$$

por lo que $n_j + n_{j'} = \alpha_j n$.

Para este algoritmo podemos comprobar que se cumplen estas hipótesis. Según (4.11):

$$c_{0_{j'}} = n + 1 - \sum_{k'=0}^{j'} n_{k'}$$

por lo que haciendo un cambio de variable según (4.17): si $k' = 0$, entonces $k = t - 1$ y si $k' = j'$, entonces $k = j$, por lo tanto

$$\left(c_{0_j}\right) + \left(c_{0_{j'}}\right) \;=\; \left(1 + \sum_{k=j+1}^{t-1} n_k\right) + \left(n + 1 - \sum_{k=t-1}^{j} n_k\right)$$

y ya que:

$$\sum_{k=j}^{t-1} n_k = n - \sum_{k=0}^{j-1} n_k$$

entonces

$$c_{0_j} + c_{0_{j'}} = n + 2 - n_j = f_1(n) - \frac{\beta}{\rho} n_j$$

con $f_1(n) = n + 2$ y siendo por tanto

- $\beta = \rho = -(2q + w_{AG}q + 1)$
- $f(q, n) = \left[(2 + w_{AG})q + 1\right] n + \frac{1}{2} w_{AG} q^2 + (2 + 2w_{AG})q + \frac{3}{2}$

A continuación nos planteamos cuál es el desequilibrio resultante si el modelo supuesto no coincidiera con la realidad (seguimos despreciando el coste de la actualización de la solución y su influencia en el desequilibrio). Supongamos que realmente

$$W_j(z) = \left[\rho c_{0_j} + f(n, q)\right] n_j + \frac{\beta + \Delta\beta}{2} n_j^2$$

En tal caso

$$\begin{aligned} T_{P_j}(z) &= [W_j(z) + W_{j'}(z)]\, t_{w_j} \\ &= \left[\rho(c_{0_j} + c_{0_{j'}}) + 2f(n,q)\right] n_j t_{w_j} + \beta n_j^2 t_{w_j} + \Delta\beta n_j^2 t_{w_j} \\ &= [\rho f_1(n) + 2f(n,q)]\, n_j t_{w_j} + \Delta\beta n_j^2 t_{w_j} \\ &= [\rho f_1(n) + 2f(n,q)]\, n\alpha_j t_{w_j} + \Delta\beta n^2 \alpha_j^2 t_{w_j} \end{aligned}$$

Por lo que el desequilibrio absoluto entre dos procesadores cualesquiera $j \neq k$ sería:

$$\begin{aligned} T_{P_j}(z) - T_{P_k}(z) &= [\rho f_1(n) + 2f(n,q)]\, n\alpha_j t_{w_j} - \Delta\beta n^2 \alpha_j^2 t_{w_j} \\ &\quad - [\rho f_1(n) + 2f(n,q)]\, n\alpha_k t_{w_k} + \Delta\beta n^2 \alpha_k^2 t_{w_k} \\ &= \Delta\beta n^2(\alpha_j \alpha_j t_{w_j} - \alpha_k \alpha_k t_{w_k}) \\ &= \Delta\beta n^2(\alpha_j - \alpha_k)\alpha_f t_{w_f} \end{aligned} \tag{4.18}$$

ya que $\alpha_j t_{w_j} = \alpha_k t_{w_k} = \alpha_f t_{w_f}$. Por lo tanto, la conclusión es que si el sistema es idealmente homogéneo ($\alpha_j = \alpha_k$), no importa realmente el ajuste del modelo a la realidad, ya que el desequilibrio temporal sería nulo. Esto es debido a que la carga estará siempre equilibrada. Por el contrario, un sistema heterogéneo sería notablemente más sensible a esta diferencia respecto al modelo de referencia, en la cantidad (4.18)

La figura 4.7 muestra gráficamente cuál sería la evolución de una iteración a lo largo de la línea segmentada bidireccional, hasta conseguir la actualización de la solución para dicha iteración.

4.2.7. Análisis de las comunicaciones

Podemos clasificar a la información que se debe transferir en la red de interconexión en tres tipos: datos de entrada ($\mathbf{H}_i$ e $\mathbf{y}_i$), datos de salida ($\mathbf{x}_{\mathrm{LS}_i}$) y datos de proceso ($(\mathbf{D}_i)_{P_j}$).

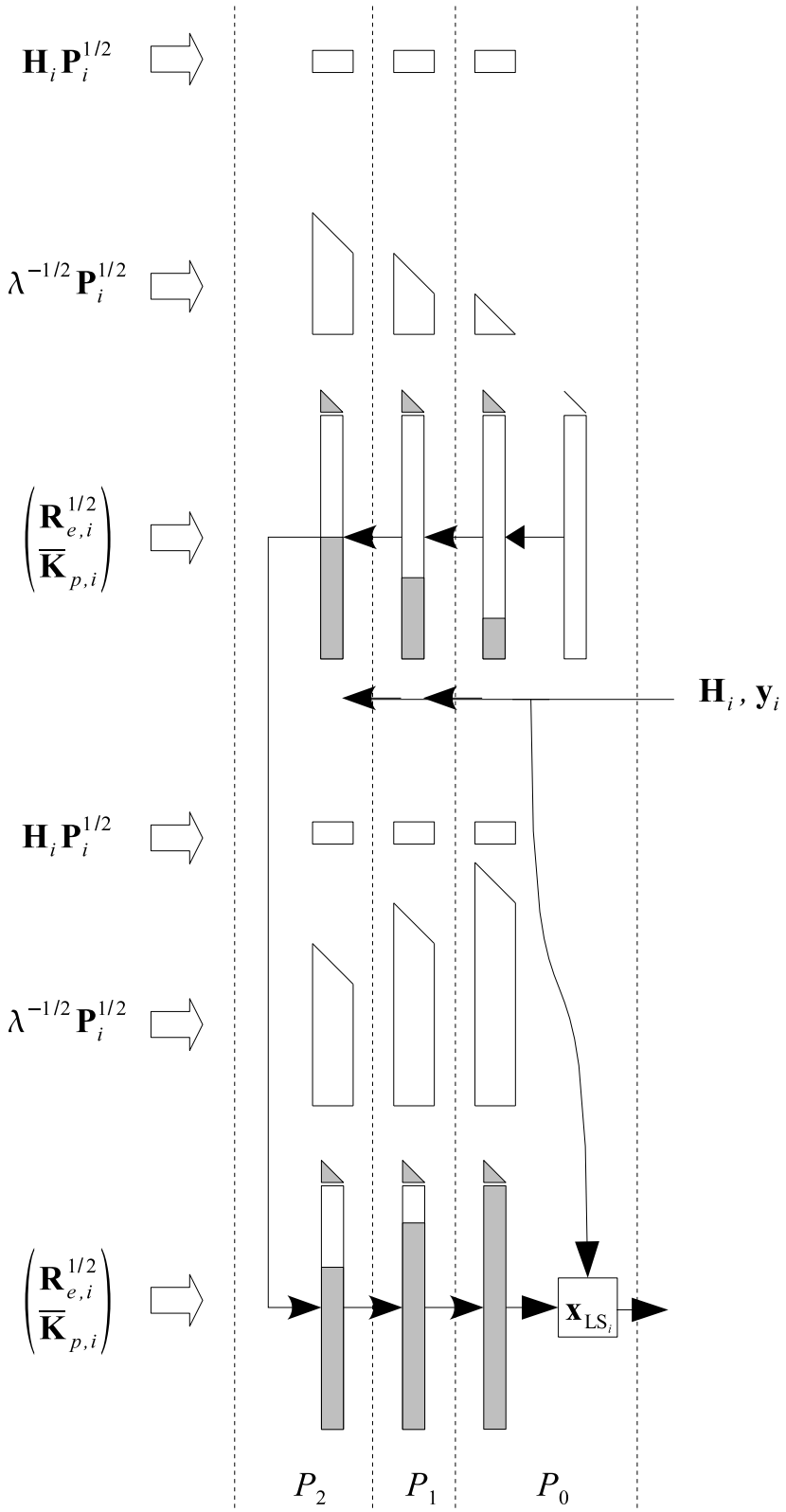

Figura 4.7: Evolución de una iteración del algoritmo SRKF en una línea segmentada bidireccional.

Todos los procesadores deben transferir el dato de proceso $(\mathbf{D}_i)_{P_j}$ al siguiente procesador de la cadena segmentada, así como poseer el dato de entrada $\mathbf{H}_i$. Con el esquema de equilibrado de carga propuesto, el procesador P_0 es el que se encarga de actualizar la solución $\mathbf{x}_{\mathrm{LS}_i}$ y enviarla a la entidad oportuna, luego es el único que requiere $\mathbf{y}_i$. La transferencia que más posibilidades plantea es $\mathbf{H}_i$ y las soluciones que planteamos son las siguientes:

- Que todos los procesadores posean $\mathbf{H}_i$, $\forall i$ antes de arrancar la ejecución del algoritmo.
- Que un proceso *difunda* dichos datos cada iteración i a todos los procesadores.
- Que se transfiera junto con $(\mathbf{D}_i)_{P_j}$

Pueden existir aplicaciones en tiempo real que no permitan aplicar la primera solución. La segunda solución exigirá una red de interconexión punto-multipunto o en su defecto, aprovechar la red de interconexión actual para emularla; además. Debido al carácter segmentado del algoritmo, cada procesador deberá almacenar cierta cantidad de datos de entrada $\mathbf{H}_i$ hasta que pueda consumirlos. Consideramos que la tercera solución es la menos exigente, más genérica y, la que en cualquier caso, se adapta más fácilmente a la situación real. Ésta será, por tanto, la hipótesis de trabajo con la que realizaremos los cálculos teóricos del coste de las comunicaciones. Posteriormente codificaremos el algoritmo paralelo.

Las dimensiones de las informaciones a transmitir son:

- $\mathbf{H}_i$: qn elementos
- Los datos no nulos de $(\mathbf{D}_i)_{P_j}$: la submatriz $\mathbf{L}_i$ $(\frac{q(q+1)}{2})$ elementos, y la submatriz $\mathbf{M}_i$ con un total de $q\sum_{k=0}^{j} n_k$ elementos no nulos.

El tiempo de comunicar datos del procesador P_j hasta el P_{j+1}, $j = 0, \ldots, p-2$ en cualquier iteración i será:

$$T_{C,P_j,i}(z) = \beta + \tau \left[qn + \frac{1}{2}q(q+1) + q\sum_{k=0}^{j} n_k \right]$$

Podemos plantear dos modelos extremos en cuanto al subsistema de comunicaciones: aquél que permite simultanear todas las comunicaciones (modelo A) y aquél en el que deben realizarse en serie (modelo B). Para el primero de ellos, el tiempo de comunicación será:

$$\begin{aligned} T_{C,i}^{(A)}(z,p) &= \max_{j=0,\ldots p-2}\{T_{C,P_j,i}(z)\} \\ &= T_{C,P_{p-2}}(z) \\ &= \beta + \tau \left[qn + \frac{1}{2}q(q+1) + q(n - n_{p-1}) \right] \end{aligned}$$

Si obviamos el llenado o vaciado de la línea segmentada, el tiempo de comunicación lo podemos aproximar por

$$\begin{aligned} T_{C}^{(A)}(z,p) &= \sum_{i=0}^{m/q-1} T_{C,i}(z,p) \\ &= \beta\frac{m}{q} + m\tau \left[n + \frac{1}{2}(q+1) + (n - n_{p-1}) \right] \\ &= \Theta\left(mn\right) \end{aligned} \tag{4.19}$$

Para el segundo modelo, (B):

$$\begin{aligned} T_{C,i}^{(B)}(z,p) &= \sum_{j=0}^{p-2} T_{C,P_j,i}(z) \\ &= \sum_{j=0}^{p-2} \left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) + q\sum_{k=0}^{j} n_k \right] \right\} \\ &= (p-1)\left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) \right] \right\} + \tau q \sum_{j=0}^{p-2}\sum_{k=0}^{j} n_k \end{aligned}$$

A continuación, intentaremos hallar una cota para $\sum_{j=0}^{p-2}\sum_{k=0}^{j} n_k$. El caso más desfavorable para este sumatorio es que los valores n_k estén ordenados en orden decreciente respecto al índice, y un caso extremo sería que $n_0 = n$ y $n_k = 0$, $\forall\, k > 0$; en tal caso, la cota superior del sumatorio doble sería $(p-1)n$, por lo tanto:

$$T_{C,i}^{(B)}(z,p) < (p-1)\left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) \right] \right\} + \tau q(p-1)n$$

Para todas las iteraciones, el resultado será:

$$\begin{aligned} T_{C}^{(B)}(z,p) &< \sum_{i=0}^{m/q-1} \left\{ (p-1)\left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) \right] \right\} + \tau q(p-1)n \right\} \\ &< \frac{m}{q}\left\{ (p-1)\left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) \right] \right\} + \tau q(p-1)n \right\} \\ &< \frac{m}{q}(p-1)\beta + m(p-1)\tau \left[n + \frac{1}{2}(q+1) \right] + \tau m(p-1)n \\ &= \Theta\left(mpn\right) \end{aligned} \tag{4.20}$$

Las consideraciones especiales que requiere el planteamiento con la línea segmentada bidireccional son que existen $2p-2$ transferencias en vez de $p-1$ y sólo en las primeras $p-1$ es necesario transferir $\mathbf{H}_i$. Estos cambios implican prácticamente un tiempo doble en

las comunicaciones, aunque no una variación en la complejidad asintótica de las mismas.

4.2.8. Escalabilidad

La escalabilidad por isoeficiencia puede expresarse viendo en qué orden de magnitud debe aumentarse la carga en función del aumento en la cantidad de procesadores. Ello se puede deducir forzando a que el tiempo de ejecución del algoritmo secuencial sea proporcional al tiempo de sobrecarga, [71]. En nuestro caso, a partir de (4.9), $T_{\text{sec}}(z) = \Theta\left(mn^2\right)$

En este algoritmo existen dos fuentes de sobrecarga: por cálculo (la actualización de la solución sólo es realizada por P_{p-1} con un coste de $\Theta\left(nq\right)$ por iteración, luego la sobrecarga total será $\Theta\left(mn\right)$) y por comunicaciones que puede oscilar entre $T_C(z,p) = \Theta\left(mn\right)$ y $T_C(z,p) = \Theta\left(mpn\right)$, luego el tiempo de sobrecarga según [71], $T_O(z,p) = pT_C(z,p)$ oscilará entre $\Theta\left(pmn\right)$ y $\Theta\left(p^2mn\right)$, por lo que la escalabilidad lo hará entre $n = \Theta\left(p\right)$ y $n = \Theta\left(p^2\right)$, siendo, evidentemente independiente de m.

4.2.9. Resultados experimentales

Los siguientes resultados han sido obtenidos mediante tests realizados en la máquina *Aldebarán*. Al igual que en el caso secuencial, todas las pruebas se han realizado con matrices con un número de filas $m = 6000$ y $\lambda = 1$.

Equilibrado de carga

Las figuras 4.8 y 4.9 muestran los tiempos aritméticos de cada procesador y el tiempo secuencial dividido por el número de procesadores para la línea segmentada unidireccional y bidireccional respectivamente. Para el caso unidireccional, observamos cierto desequilibrio, ya que el procesador P_0 está ligeramente más sobrecargado que el resto, mientras que en el caso bidireccional el equilibrado es notablemente mejor.

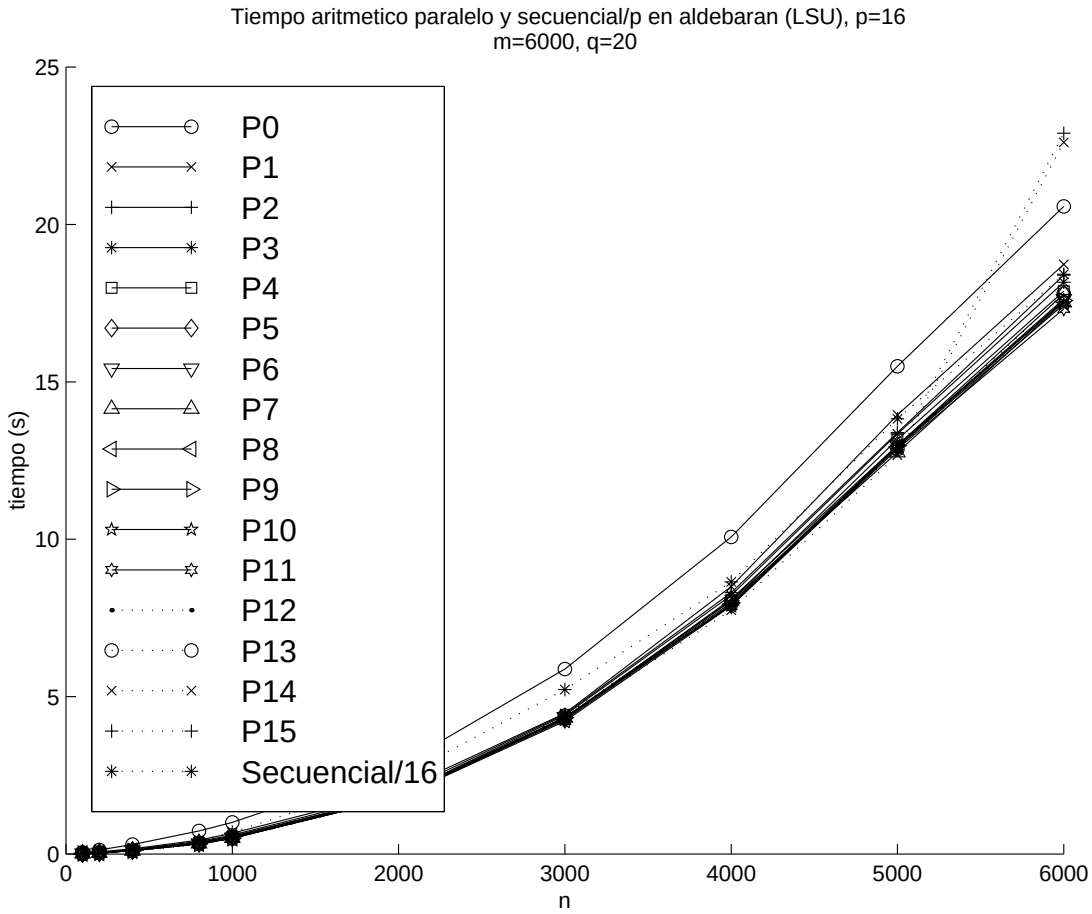

Figura 4.8: Tiempos aritméticos paralelos de cada procesador y secuencial dividido por el número de procesadores para la línea segmentada unidireccional (LSU)

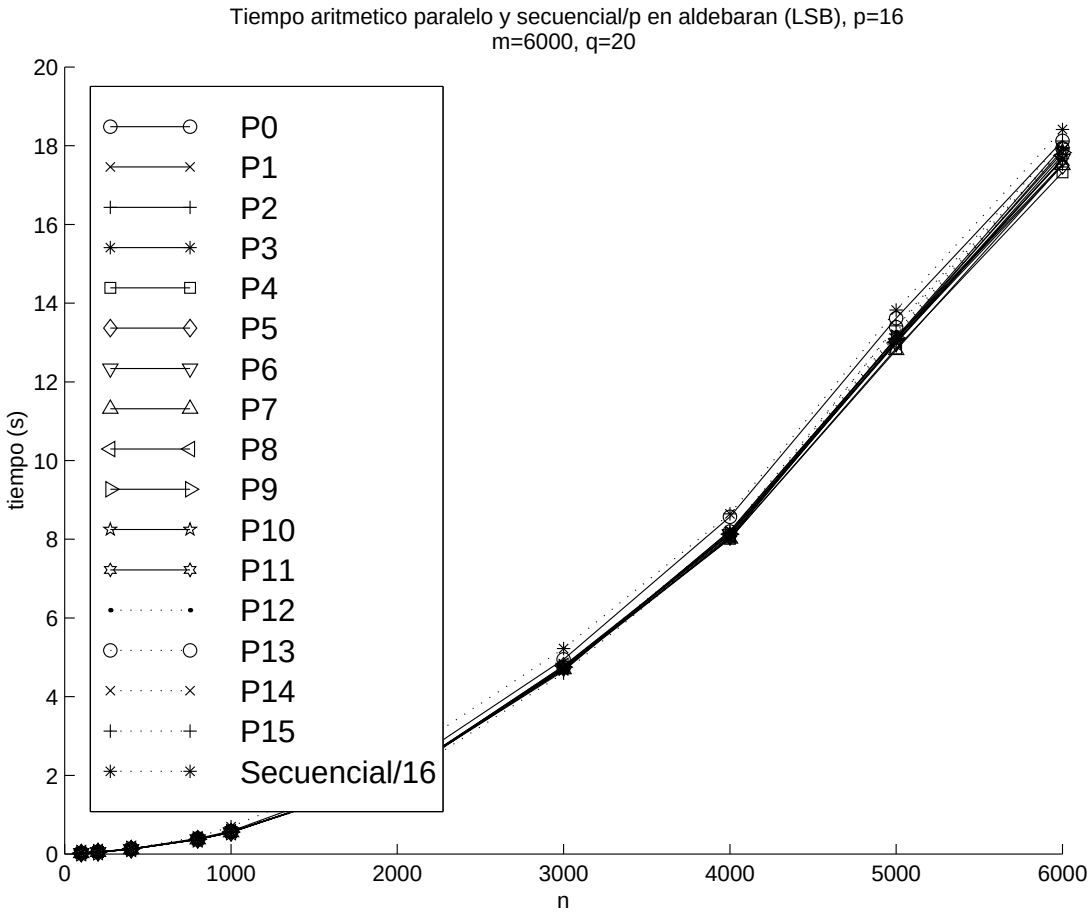

Figura 4.9: Tiempos aritméticos paralelos de cada procesador y secuencial dividido por el número de procesadores para la línea segmentada bidireccional (LSB)

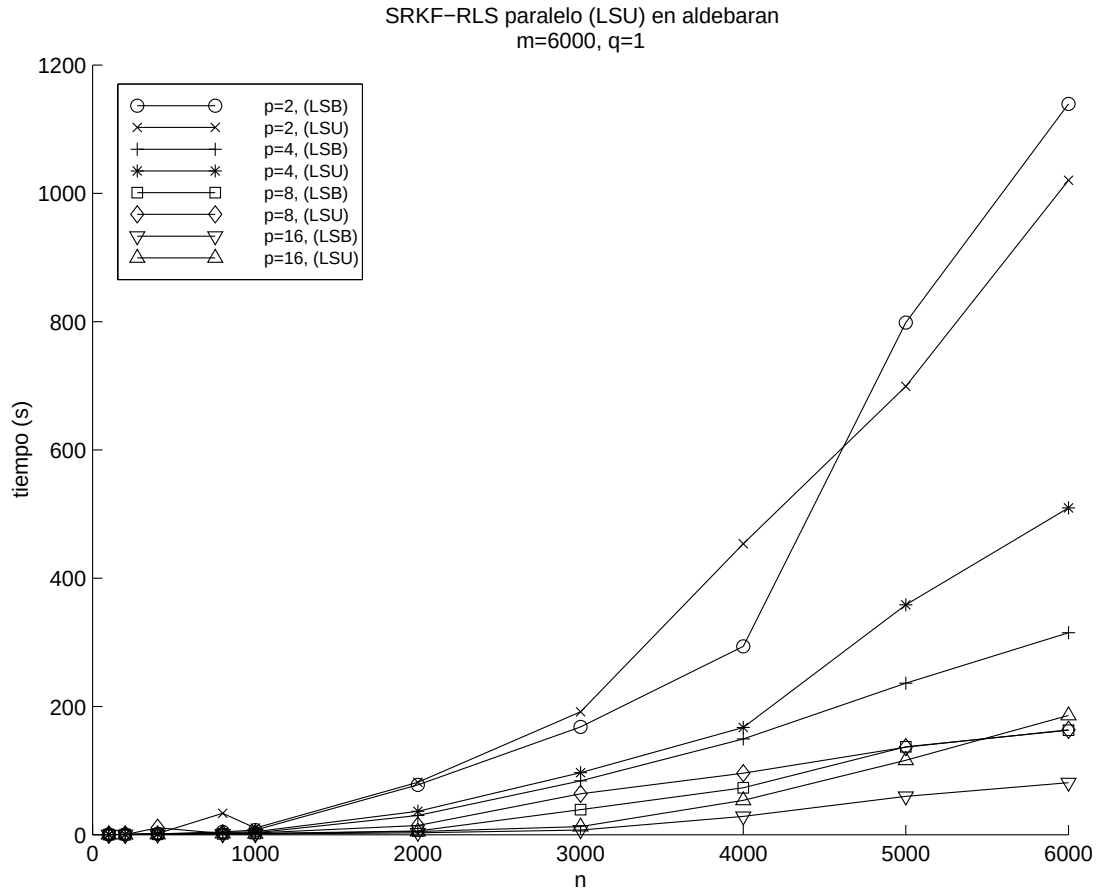

Figura 4.10: Tiempos de ejecución paralelo para el algoritmo SRKF-RLS en la línea segmentada unidireccional (LSU) y bidireccional (LSB), con $q = 1$

Tiempos de ejecución paralelos

Las figuras 4.10 y 4.11 muestran los tiempos de ejecución comparados del caso unidireccional y bidireccional del algoritmo para $q = 1$ y $q = 50$ respectivamente.

Podemos observar que el caso bidireccional tiene un mejor comportamiento para valores de q bajos que se invierte para valores de q altos.

Tiempos de sobrecarga

La figura 4.12 muestra el tiempo de sobrecarga estimado como la diferencia entre el tiempo paralelo y el tiempo secuencial, dividida por el número de procesadores.

Podemos observar cómo dicho tiempo es mucho más notable para el caso bidireccional que en el unidireccional para este valor relativamente elevado de q, siendo además

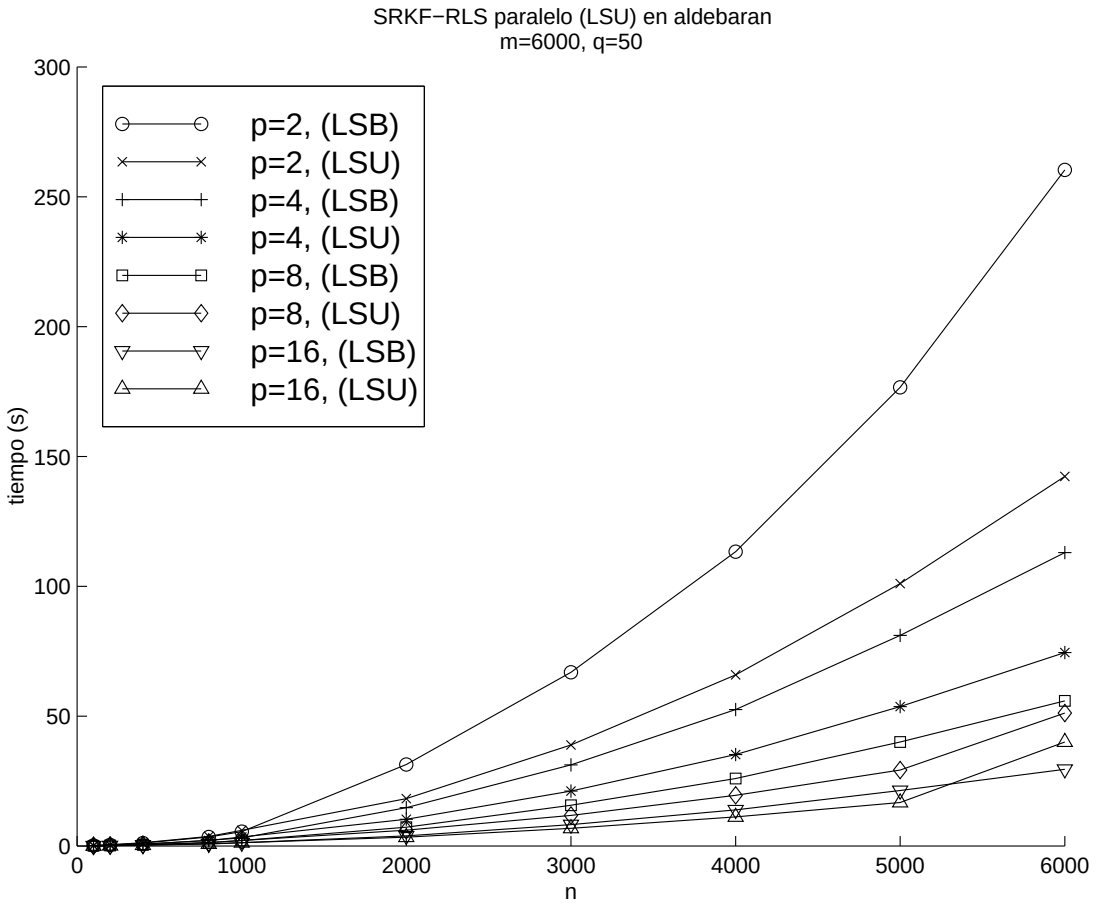

Figura 4.11: Tiempos de ejecución paralelo para el algoritmo SRKF-RLS en la línea segmentada unidireccional (LSU) y bidireccional (LSB), con $q = 50$

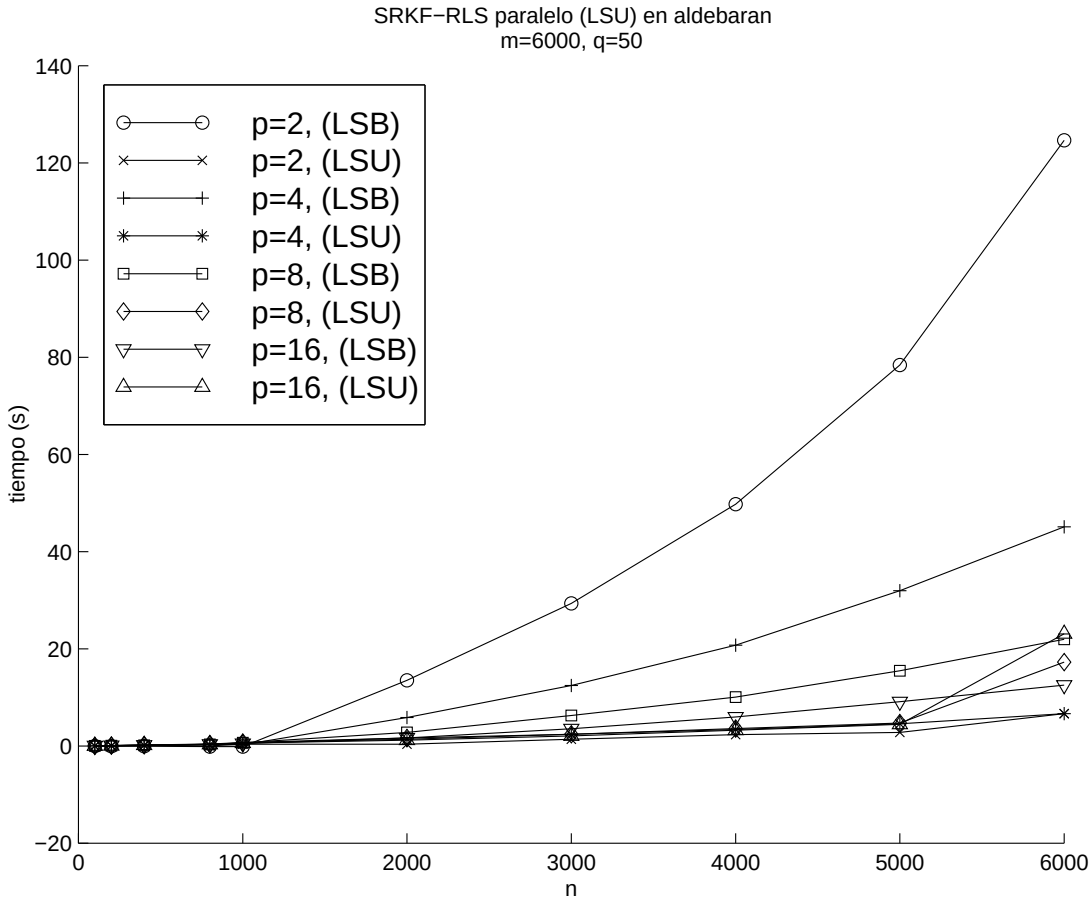

Figura 4.12: Tiempos de sobrecarga para el algoritmo SRKF-RLS en la línea segmentada unidireccional (LSU) y bidireccional (LSB), con $q = 50$

claramente cuadrático. Este carácter cuadrático no puede ser debido al tiempo de comunicación, ya que éste resultó ser lineal con n, expresiones (4.19) ó (4.20), ni a la sobrecarga aritmética por la actualización del estado, por lo que la causa más verosímil es cierto *reflejo* de la carga procesada por cada procesador, debido a retardos acumulados por la atención a sendos flujos de información. La carga es proporcional al valor del parámetro q, por lo que será mayor la influencia cuanto mayor sea dicho parámetro. Adicionalmente, con el esquema de equilibrado bidireccional, cualquier desequilibrio computacional en los procesadores tiene como consecuencia una diferencia en el tiempo de ejecución de tipo cuadrático tal y como se observó en (4.18).

Eficiencia

En las figuras 4.13 y 4.14 se comparan las eficiencias para la línea segmentada unidireccional y bidireccional para $q = 1$ y $q = 50$ respectivamente, donde se observa que para valores de q pequeños como $q = 1$ la eficiencia es similar, aunque sutílmente mejor para el caso bidireccional, y se invierte para valores de q relativamente altos, por lo motivos expuestos anteriormente.

4.3 Conclusiones y posibles mejoras

Hemos llevado a cabo la paralelización del algoritmo SRKF con una notable eficiencia para dos esquemas de equilibrado de carga. Ambas esquemas (unidireccional y bidireccional) pueden ser equilibrados de manera heurística mediante test previos con sucesivos ajustes hasta lograr el equilibrio deseado. Adicionalmente, podemos plantear esquemas de equilibrado adaptativos con los que la cantidad de columnas asignadas a cada procesador cambie en función de la capacidad computacional potencialmente variante con el tiempo, siempre y cuando la sobrecarga debido a estos esquemas no vaya en detrimento del tiempo de ejecución.

Para el esquema bidireccional, podemos plantear un esquema de atención a las comuni-

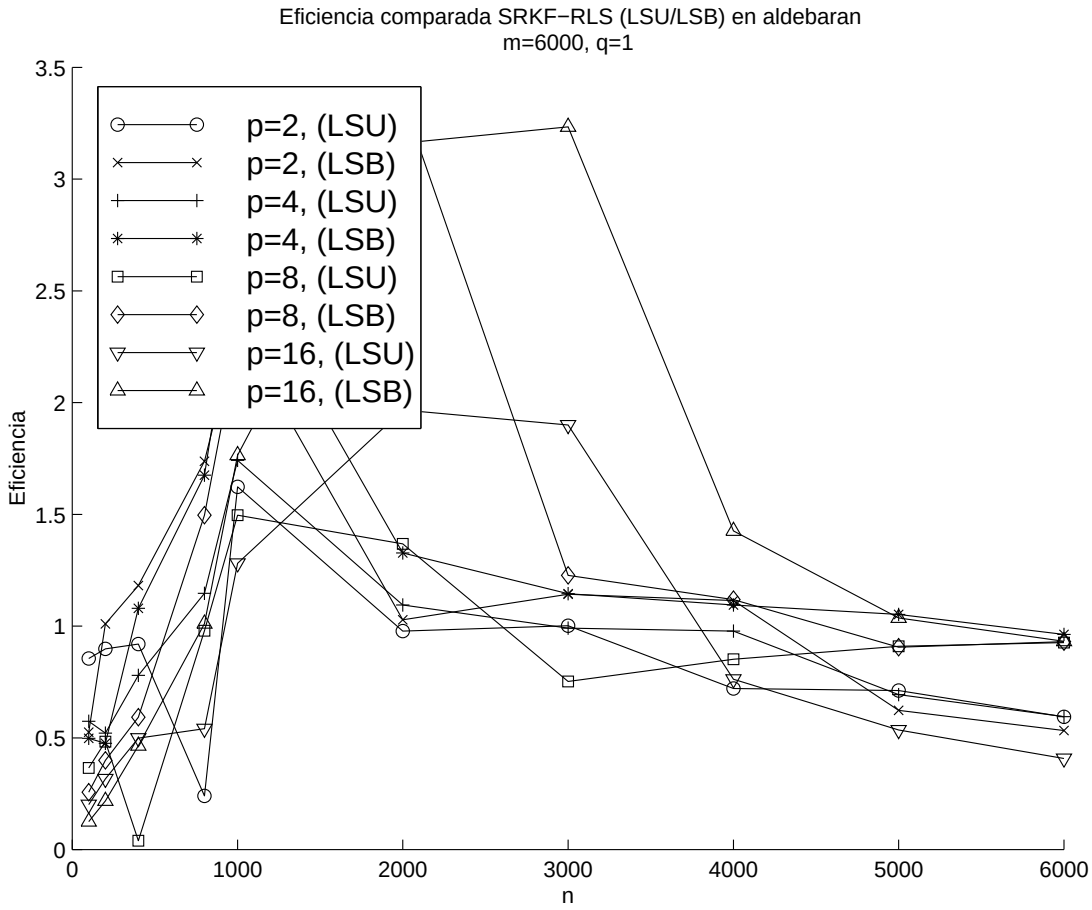

Figura 4.13: Eficiencia comparada para el algoritmo SRKF-RLS en la línea segmentada unidireccional (LSU) y bidireccional (LSB), con $q = 1$

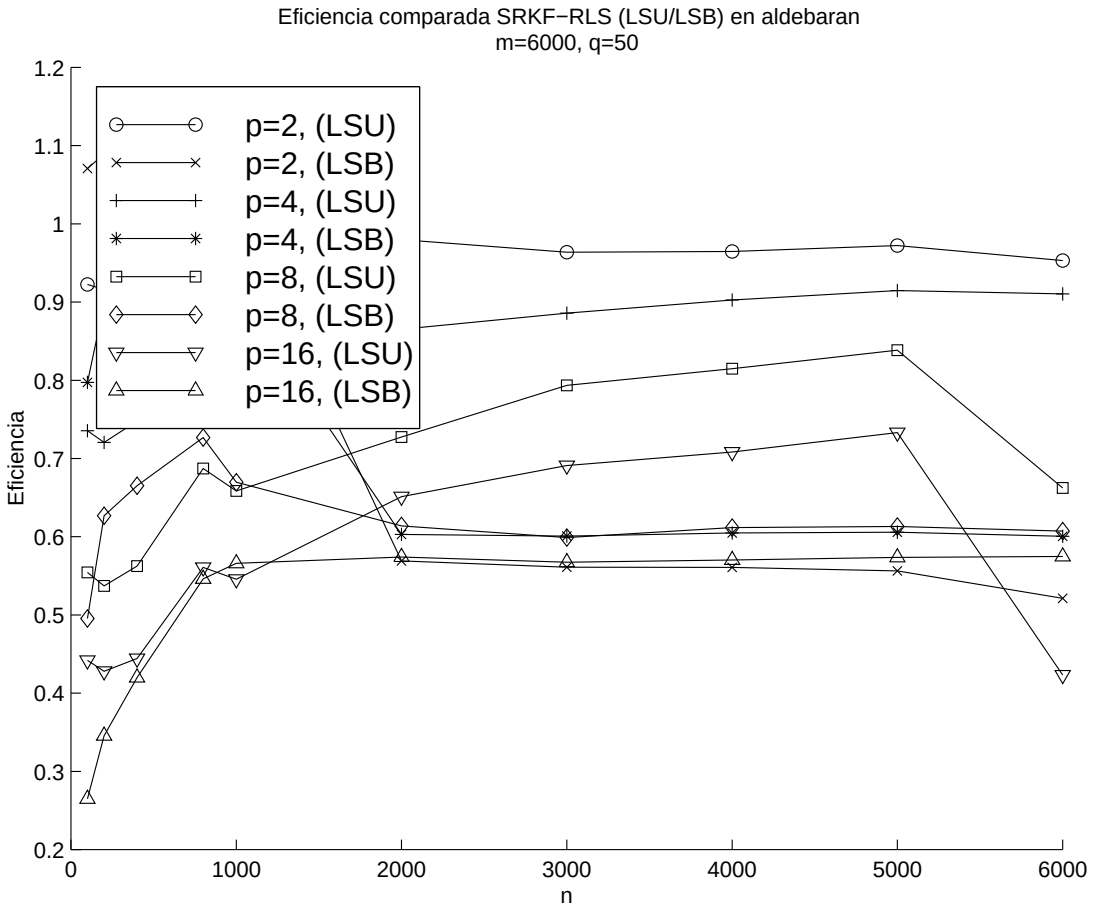

Figura 4.14: Eficiencia comparada para el algoritmo SRKF-RLS en la línea segmentada unidireccional (LSU) y bidireccional (LSB), con $q = 50$

caciones basado en hilos o *threads*, con los cuales la equidad en la atención sea notablemente superior, intentando atenuar las causas que motivan una menor eficiencia respecto al caso unidireccional conforme el parámetro q va creciendo.

Una posible ampliación del estudio del algoritmo paralelo en sus dos versiones es contrastar el comportamiento de los mismos en función del modo de comunicación: en las implementaciones se han utilizado las versiones asíncronas de los envíos, pudiendo ser interesante la observación del comportamiento cuando los envíos son síncronos.

Tanto para el algoritmo secuencial como para las versiones paralelas, puede resultar de interés el conocer el valor óptimo del parámetro q para obtener el mejor tiempo de ejecución posible, aunque la aplicación general que pudiera utilizar este algoritmo y el contexto de la misma sean realmente quienes definan dicho valor (períodos de muestreo de la señal, velocidad en la actualización de la solución, etc.). Asimismo, puede resultar interesante el observar el comportamiento de los algoritmos con una versión *empaquetada* de la matriz triangular inferior $\mathbf{P}_i^{1/2}$.

5

Algoritmo RLS basado en la versión raíz cuadrada del Filtro de Información

En este capítulo mostramos un segundo algoritmo para resolver el problema RLS, basado en el Filtro de Información. Realizaremos la paralelización del mismo y obtendremos las características más importantes de su comportamiento.

5.1 Algoritmo secuencial

La siguiente variante del *Filtro de Kalman*, que se muestra en [13], la denotaremos por el acrónimo SRIF (*Square Root Information Filter*). Este algoritmo ya fue introducido en el subapartado 3.7.2 y repetimos a continuación por claridad en la exposición.

Entrada: $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$, $\mathbf{y} = (\mathbf{y}_0^*, \mathbf{y}_1^*, \ldots)^*$, $\hat{\mathbf{x}}_0$, $\mathbf{P}_0^{1/2}$, λ

Salida: Actualización de $\mathbf{x}_{\mathrm{LS}}$ cada iteración i-ésima

para $i = 0, \ldots, m/q - 1$ **hacer**

Calcular y aplicar $\mathbf{\Theta}_i$ unitaria tal que:

$$\mathbf{E}_i\mathbf{\Theta}_i = \begin{pmatrix} \lambda^{1/2}\mathbf{P}_i^{-*/2} & \lambda^{1/2}\mathbf{H}_i^* \\ \hat{\mathbf{x}}_i^*\mathbf{P}_i^{-*/2} & \mathbf{y}_i^* \\ \lambda^{-1/2}\mathbf{P}_i^{1/2} & \mathbf{0} \end{pmatrix} \mathbf{\Theta}_i = \begin{pmatrix} \mathbf{P}_{i+1}^{-*/2} & \mathbf{0} \\ \hat{\mathbf{x}}_{i+1}^*\mathbf{P}_{i+1}^{-*/2} & \mathbf{e}_i^*\mathbf{R}_{e,i}^{-*/2} \\ \mathbf{P}_{i+1}^{1/2} & -\overline{\mathbf{K}}_{p,i} \end{pmatrix} = \mathbf{F}_i \tag{5.1}$$

$$\hat{\mathbf{x}}_{i+1} = \lambda^{-1/2}\hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i}\mathbf{R}_{e,i}^{-1/2}\mathbf{e}_i \tag{5.2}$$

$$\mathbf{x}_{\mathrm{LS}_i} = \left(\lambda^{1/2}\right)^{i+1}\hat{\mathbf{x}}_{i+1}$$

fin para

Tal y como ocurría con el algoritmo SRKF del capítulo anterior, en las iteraciones del algoritmo se propagan ciertos factores raíz cuadrada, en este caso de la matriz $\mathbf{P}_i^{-1}$ que denotaremos por $\mathbf{P}_i^{-1} = \mathbf{P}_i^{-*/2}\mathbf{P}^{-1/2}$. De nuevo, estos factores no tienen una expresión única, pero conviene, con el ánimo de intentar ahorrar costes aritméticos, que posean cierta estructura. Si empleamos la factorización de Cholesky, $\mathbf{P}_i^{-*/2}$ será una matriz triangular inferior y $\mathbf{P}_i^{-1/2}$ una triangular superior, así como su inversa $\mathbf{P}_i^{1/2}$, que es la que realmente se propaga.

Tal y como podríamos comprobar, esta variante del Filtro de Kalman realiza en el fondo, las mismas operaciones que el filtro de Kalman original. Al reestructurar las operaciones surgen nuevas variables con un doble objetivo: claridad en la expresión algorítmica y descripción de algunos conceptos importantes en el campo del Control o Teoría de la Señal; así pues la matriz

$$\mathbf{R}_{e,i} = \mathbf{I} + \mathbf{H}_i\mathbf{P}_i\mathbf{H}_i^* \in \mathbb{C}^{q\times q} \tag{5.3}$$

se puede interpretar como la matriz de covarianza del error en el instante i-ésimo: $\mathbf{R}_{e,i} = \mathrm{cov}(\mathbf{e}_i)$; la matriz

$$\overline{\mathbf{K}}_{p,i} = \lambda^{-1/2}\mathbf{P}_i\mathbf{H}_i^*\mathbf{R}_{e,i}^{-*/2} \in \mathbb{C}^{n\times q} \tag{5.4}$$

resulta ser la matriz de covarianza cruzada entre el error de la estimación en el instante i-ésimo y el estado en el instante $(i+1)$-ésimo: $\overline{\mathbf{K}}_{p,i} = \mathrm{cov}(\mathbf{x}_{i+1}, \mathbf{e}_i)$

Posteriormente nos será útil una expresión alternativa para $\mathbf{P}_{i+1}^{-1}$: de (3.20) y el lema de inversión de matrices[1]

$$\begin{aligned}
\mathbf{P}_{i+1}^{-1} &= \left[\lambda^{-1}\left(\mathbf{P}_i - \mathbf{P}_i\mathbf{H}_i^*\left[\mathbf{H}_i\mathbf{P}_i\mathbf{H}_i^* + \mathbf{I}\right]^{-1}\mathbf{H}_i\mathbf{P}_i\right)\right]^{-1} \\
&= \lambda\left[\mathbf{P}_i^{-1} + \mathbf{P}_i^{-1}\mathbf{P}_i\mathbf{H}_i^*\left(-\mathbf{H}_i\mathbf{P}_i\mathbf{P}_i^{-1}\mathbf{P}_i\mathbf{H}_i^* + \left(\mathbf{H}_i\mathbf{P}_i\mathbf{H}_i^* + \mathbf{I}\right)\right)^{-1}\mathbf{H}_i\mathbf{P}_i\mathbf{P}_i^{-1}\right] \\
&= \lambda\left[\mathbf{P}_i^{-1} + \mathbf{H}_i^*\mathbf{H}_i\right]
\end{aligned} \tag{5.5}$$

La matriz $\mathbf{\Theta}_i$ es una transformación unitaria que obtiene ceros en las entradas apropiadas de $\mathbf{E}_i$; el resto de vectores y matrices se definen tal y como se hizo en la subsección 3.7. Como comentábamos anteriormente, esta forma de realizar la iteración para la obtención de la estimación del estado es equivalente a la iteración del *filtro de Kalman*, ecuaciones (3.19) y (3.20), [13]. Ello puede verificarse comprobando que $\mathbf{E}_i\mathbf{E}_i^* = \mathbf{F}_i\mathbf{F}_i^*$.

Para cada iteración del algoritmo se construye $\mathbf{E}_i \in \mathbb{C}^{(2n+1)\times(n+q)}$, entonces se obtienen ciertos ceros en esta matriz y finalmente la solución se actualiza. De nuevo, $\mathbf{E}_{i+1}$ se vuelve a construir con nuevos datos y la solución se actualiza otra vez, y así sucesivamente. Cada iteración del algoritmo utiliza q filas consecutivas de $\mathbf{H}$ (o $\mathbf{A}$) y de $\mathbf{y}$. Por simplicidad, supondremos que m es un múltiplo entero de q, $(m = Mq)$. Sea:

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda^{1/2}\mathbf{I}_n & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \lambda^{-1/2}\mathbf{I}_n \end{pmatrix} \quad \mathbf{C}_i = \left(\begin{array}{c} \mathbf{P}_i^{-*/2} \\ \hline \hat{\mathbf{x}}_i^*\mathbf{P}_i^{-*/2} \\ \hline \mathbf{P}_i^{1/2} \end{array}\right) \quad \mathbf{D}_i = \left(\begin{array}{c} \mathbf{H}_i^* \\ \hline \mathbf{y}_i^* \\ \hline \mathbf{0} \end{array}\right)$$

entonces $\mathbf{E}_i$ puede expresarse como $\mathbf{E}_i = \mathbf{\Lambda}\left(\mathbf{C}_i \;\; \mathbf{D}_i\right) = \left(\mathbf{\Lambda}\mathbf{C}_i \;\; \mathbf{\Lambda}\mathbf{D}_i\right)$.

[1] $(\mathbf{A}+\mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{DA}^{-1}\mathbf{B} + \mathbf{C}^{-1})^{-1}\mathbf{DA}^{-1}$

El principal objetivo es conseguir zeros de manera eficiente en las primeras n filas de $\mathbf{\Lambda D}_i$, aplicando una secuencia de transformaciones unitarias, $\mathbf{\Theta}_i$. Un ejemplo con $q = 2$ y $n = 3$ se muestra a continuación:

$$\mathbf{E}_i\mathbf{\Theta}_i = \mathbf{\Lambda}\left(\mathbf{C}_i \;||\; \mathbf{D}_i\right)\mathbf{\Theta}_i = \left(\begin{array}{ccc||cc} x & 0 & 0 & x & x \\ x & x & 0 & x & x \\ x & x & x & x & x \\ \hline x & x & x & x & x \\ \hline x & x & x & 0 & 0 \\ 0 & x & x & 0 & 0 \\ 0 & 0 & x & 0 & 0 \end{array}\right)\mathbf{\Theta}_i = \left(\begin{array}{ccc||cc} * & 0 & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \\ \hline * & * & * & * & * \\ \hline * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \end{array}\right) = \mathbf{F}_i$$

Estas operaciones podemos llevarlas a cabo de múltiples maneras: podemos utilizar rotaciones de Givens para obtener ceros en cada fila r-ésima, $r = 1, \ldots, n$ de $\mathbf{\Lambda D}_i$, rotando cada elemento de dicha fila respecto a la r-ésima entrada diagonal de la submatriz $\mathbf{\Lambda C}_i$ para conseguir el objetivo indicado. Otra forma más rápida teóricamente puede ser utilizando transformaciones de Householder para reflejar cada r-ésima fila, $r = 1, \ldots n$, de $\mathbf{\Lambda D}_i$ respecto a la r-ésima entrada diagonal de $\mathbf{\Lambda C}_i$. La disminución prevista en el tiempo de ejecución de la opción basada en las transformaciones de Householder respecto a la basada en las rotaciones de Givens puede truncarse debido a ciertos detalles de implementación: si utilizamos un paquete de álgebra lineal numérica estándar, como pudiera ser `LAPACK`, la aplicación de las transformaciones de Householder requiere que todos los elementos del vector en cuestión sean contiguos (esto no ocurre en esta situación si queremos aprovechar la estructura triangular de las primeras n filas de $\mathbf{\Lambda C}_i$), por lo que sería necesario mover o permutar(parte de) la r-ésima columna de $\mathbf{C}_i$ para conseguir cumplir la condición citada, empleando cierta cantidad de tiempo en ello. Para paliar este problema, podemos concebir un método híbrido que consigue mejores resultados que los dos métodos planteados hasta ahora y que consiste en utilizar una transformación de Householder $\mathbf{J}_r$ para reflejar las

últimas $q-1$ componentes de la r-ésima fila de $\mathbf{\Lambda D}_i$ respecto al primer elemento de ella, y entonces rotar dicho elemento respecto de la r-ésima entrada diagonal de $\mathbf{\Lambda C}_i$, mediante una rotación de Givens $\mathbf{G}_r$.

Podemos aprovechar la estructura de $\mathbf{F}_i$ para comenzar la siguiente iteración con el mínimo movimiento posible de datos entre las distintas variables, ya que:

$$\mathbf{E}_i\mathbf{\Theta}_i = (\mathbf{\Lambda C}_i, \mathbf{\Lambda D}_i)\,\mathbf{\Theta}_i = \mathbf{F}_i = \begin{pmatrix} \mathbf{C}_{i+1} & \begin{pmatrix} \mathbf{0} \\ \mathbf{e}_i^* \mathbf{R}_{e,i}^{-*/2} \\ -\overline{\mathbf{K}}_{p,i} \end{pmatrix} \end{pmatrix},$$

por lo que la submatriz $\mathbf{\Lambda C}_i$ procesada se convierte automáticamente en $\mathbf{C}_{i+1}$. El esquema de almacenamiento con una mayor velocidad es por columnas en la implementación Fortran. El pseudocódigo del algoritmo se muestra en **Algoritmo 4**, utilizando una notación similar a Matlab para las submatrices.

5.1.1. Costes aritméticos

Los costes aritméticos asociados a la versión híbrida (método que conjuga tanto las transformaciones de Householder como las rotaciones de Givens) del **Algoritmo 4** son:

- el escalado de $\mathbf{\Lambda C}_i$ y $\mathbf{\Lambda D}_i$, aprovechando la estructura de las submatrices: $qn+n^2+n$ flops, si $\lambda \neq 1$,
- Cada fila a anular de las n primeras de $\mathbf{\Lambda D}_i$ requiere:
  - El cálculo de una transformación de Householder de orden q: $w_{CH}(q)$ flops.
  - La aplicación de la transformación de Householder a un total de $n+1$ filas: $w_{AH}(q)(n+1)$ flops, donde $w_{AH}(q)$ es el coste de aplicar la transformación de Householder de orden q a una sola fila.
  - El cálculo de una rotación de Givens: w_{CG} flops

Algoritmo 4 RLS secuencial: algoritmo raíz cuadrada del Filtro de Información

Entrada: $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$, $\mathbf{y} = (\mathbf{y}_0^*, \mathbf{y}_1^*, \ldots)^*$, λ, $\mathbf{P}_0^{-*/2}$, $\mathbf{P}_0^{1/2}$, $\hat{\mathbf{x}}_0$
Salida: Actualización de $\mathbf{x}_{\mathrm{LS}}$ cada iteración i-ésima

$\mathbf{\Lambda} = \begin{pmatrix} \lambda^{1/2}\mathbf{I}_n & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \lambda^{-1/2}\mathbf{I}_n \end{pmatrix}$

$\mathbf{C}_0 = \begin{pmatrix} \mathbf{P}_0^{-*/2} \\ \hat{\mathbf{x}}_0\mathbf{P}_0^{-*/2} \\ \mathbf{P}_0^{1/2} \end{pmatrix}$

para $i = 0, \ldots, m/q - 1$ **hacer**

  $\mathbf{D}_i \leftarrow \begin{pmatrix} \mathbf{H}_i^* \\ \mathbf{y}_i^* \\ \mathbf{0} \end{pmatrix}$

  para $r = 1 : n$ **hacer**

    Calcular la transformación de Householder $\mathbf{J}_r$ a partir de $\mathbf{\Lambda D}_i(r, [1:q])$

    Aplicar $\mathbf{J}_r$:

$$\mathbf{\Lambda D}_i([r : r+n], [1:q]) \leftarrow \mathbf{\Lambda D}_i([r : r+n], [1:q])\mathbf{J}_r$$

    Calcular la rotación de Givens $\mathbf{G}_r$ a partir de $\mathbf{\Lambda C}_i(r, r)$ y $\mathbf{\Lambda D}_i(r, 1)$

    Aplicar $\mathbf{G}_r$:

$$[\mathbf{\Lambda C}_i([z], r), \quad \mathbf{\Lambda D}_i([z], 1)] \leftarrow [\mathbf{\Lambda C}_i([z], r), \quad \mathbf{\Lambda D}_i([z], 1)]\,\mathbf{G}_r$$

    donde z es el intervalo de filas $r : r + n + 1$

  fin para

  $\mathbf{C}_{i+1} \leftarrow \mathbf{\Lambda C}_i$

  $\begin{pmatrix} \mathbf{e}_i^*\mathbf{R}_{e,i}^{-*/2} \\ -\overline{\mathbf{K}}_{p,i} \end{pmatrix} \leftarrow \mathbf{\Lambda D}_i([n+1 : 2n+1], [1:q])$

  Actualizar la estimación del estado $\hat{\mathbf{x}}_{i+1} \leftarrow \lambda^{-1/2}\hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i}\mathbf{R}_{e,i}^{-1/2}\mathbf{e}_i$

  Actualizar la solución LS $\mathbf{x}_{\mathrm{LS}_i} \leftarrow \left(\lambda^{1/2}\right)^{i+1}\hat{\mathbf{x}}_{i+1}$

fin para

- La aplicación de una rotación de Givens a un par de vectores con un total de $n+2$ elementos: $w_{AG}(n+2)$ flops, donde w_{AG} es el tiempo de aplicar la rotación a un par de elmentos.

luego el coste de anular una fila es:

$$w_{CH}(q) + w_{AH}(q)(n+1) + w_{CG} + w_{AG}(n+2) \quad \text{flops}$$

- Actualización del estado y la solución LS: podemos basarnos en la expresión (4.6) de la versión raíz cuadrada del filtro de Kalman

$$4qn + q^2 \quad \text{flops}$$

más $2n+1$ flops si $\lambda \neq 1$.

Por lo que para anular las n primeras filas de $\mathbf{\Lambda D}_i$, es decir, completar una iteración del algoritmo, serán necesarios

$$\left[w_{CH}(q) + w_{AH}(q)(n+1) + w_{CG} + w_{AG}(n+2)\right] n + 4qn + q^2 \quad \text{flops}$$

más $qn + n^2 + n + 2n + 1$ flops si $\lambda \neq 1$.

$\mathbf{H}$ es una matriz $m \times n$, pero sus filas serán procesadas en grupos de q filas, por lo que el número total de iteraciones será de m/q. Por simplicidad, supondremos que m es un múltiplo de q. Consecuentemente, el término de mayor peso en el coste total será:

$$\begin{aligned} W_{\text{sec,SRIF}}(z)|_{\lambda=1} &\approx (w_{AH}(q) + w_{AG})n^2\frac{m}{q} \quad \text{flops} \\ W_{\text{sec,SRIF}}(z)|_{\lambda\neq1} &\approx (w_{AH}(q) + w_{AG} + 1)n^2\frac{m}{q} \quad \text{flops} \end{aligned} \tag{5.6}$$

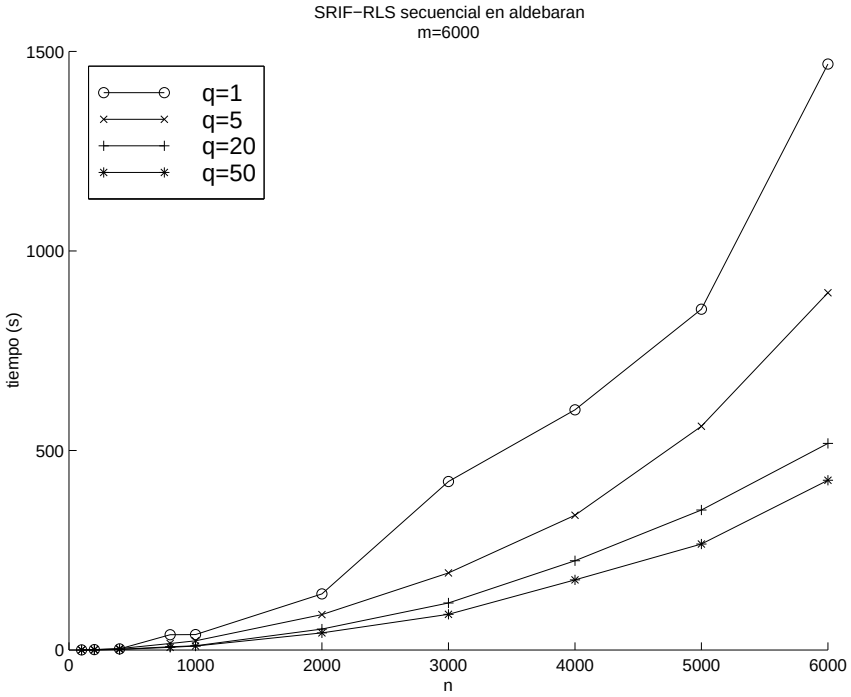

Figura 5.1: Tiempo de ejecución del algoritmo secuencial *raíz cuadrada para el filtro de información.*

5.1.2. Resultados experimentales

La figura 5.1 muesta el tiempo de ejecución en segundos del *algoritmo secuencial raíz cuadrada para el Filtro de Información* en función de n, y para varios valores de q, siendo $m = 6000$, en la máquina *Aldebarán*. Podemos observar un comportamiento asintótico respecto al parámetro q, que atribuimos a dos factores: un factor arquitectónico (memoria *cache*), tal y como ocurrió en el algoritmo SRKF del capítulo anterior, y un factor algorítmico (coste (5.6)), de donde se desprende el comportamiento observado.

Versión basada en rotaciones de Givens exclusivamente

Descartamos a priori la versión basada exclusivamente en rotaciones de Givens debido a su previsible mayor coste. En cualquier caso, vamos a considerar esta versión con sus peculiaridades para realizar una comparación con la versión híbrida. Considerando los términos más significativos, la versión basada en Givens, que denotaremos por SRIFG, tiene un coste aproximado de:

$$\begin{aligned} W_{\text{sec,SRIFG}}(z) &\approx \sum_{i=0}^{m/q-1}\sum_{r=1}^{n}\left\{\left(\sum_{c=1}^{q-1} w_{AG}(n+1)\right) + w_{AG}(n+2)\right\} \\ &\approx \sum_{i=0}^{m/q-1}\sum_{r=1}^{n} q w_{AG}(n+1) \\ &\approx mn^2 w_{AG} \end{aligned} \tag{5.7}$$

donde el índice c recorre los $q-1$ elementos a anular de la fila r-ésima, teniendo que aplicar las rotaciones a un par de vectores de longitud $n+1$; adicionalmente hay que aplicar una rotación más, para anular el elemento no nulo que nos queda, sobre la diagonal de $\mathbf{P}_i^{-*/2}$.

Si consideramos un coste para la aplicación de las transformaciones de Householder de $w_{AH}(q) = 4q$ y $w_{AG} = 6$ para las rotaciones de Givens, [87], entonces de (5.6) y (5.7):

$$\begin{aligned} W_{\text{sec,SRIF}}(z) &\approx \left(4 + \frac{6}{q}\right) mn^2 \quad \text{flops} \\ W_{\text{sec,SRIFG}}(z) &\approx 6mn^2 \quad \text{flops} \end{aligned}$$

de lo que se deduce que para valores relativamente pequeños de q, puede ser más ventajoso el emplear el método basado exclusivamente en rotaciones de Givens, y viceversa. Esta conclusión viene corroborada por los resultados de la gráfica que aparece en la figura 5.2. Para el caso en el que $q = 1$, el ratio es aproximadamente la unidad, debido a que son realmente el mismo algoritmo, ya que no se utilizan las transformaciones de Householder.

5.2 Algoritmo paralelo

En la presente sección describiremos los detalles de la paralelización del algoritmo SRIF, [97], basado en la aplicación de transformaciones de Householder y de rotaciones de Givens. Igualmente que en el algoritmo SRKF mostrado en el capítulo anterior, hemos centrado el esfuerzo en la paralelización de la parte más costosa del algoritmo secuencial,

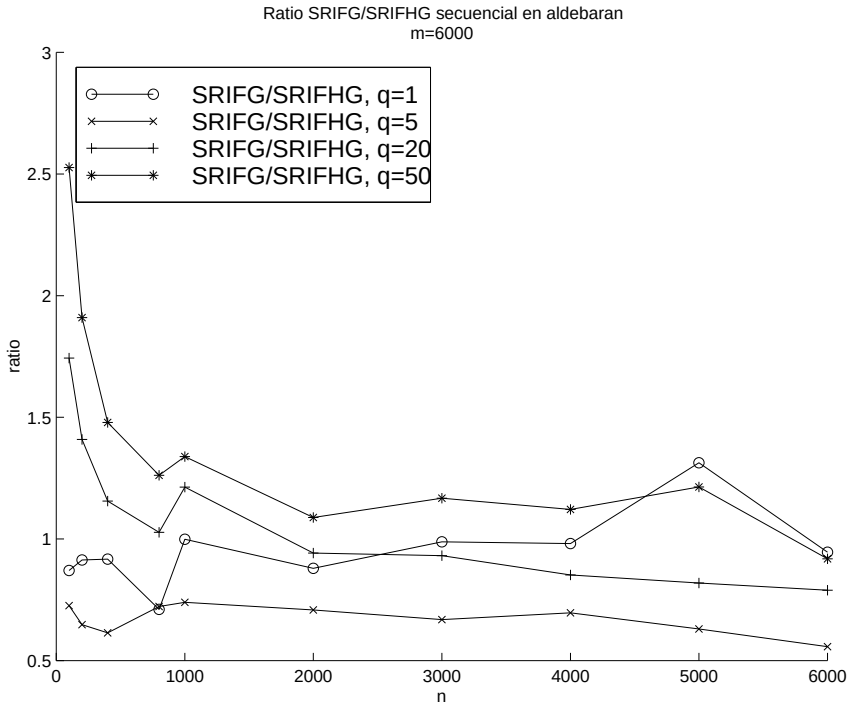

Figura 5.2: Ratio entre el tiempo de ejecución de la versión basada en Givens (SRIFG) y la versión híbrida (Householder + Givens, SRIFHG), para varios valores de q

es decir, hemos excluido la parte de la actualización del estado y la solución por tener un coste despreciable frente al resto del código. Tal y como ocurrió en el algoritmo SRKF, observaremos que la paralelización del algoritmo desde una perspectiva segmentada, prevé un mejor comportamiento del mismo.

5.2.1. Descomposición de los datos, tareas y dependencias

Con el objetivo de simplificar la notación, la matriz problema $\mathbf{E}_i$ o la matriz final $\mathbf{F}_i$ la descompondremos como sigue:

$$\left(\begin{array}{c|c} \lambda^{1/2}\mathbf{P}_i^{-*/2} & \lambda^{1/2}\mathbf{H}_i^* \\ \hline \hat{\mathbf{x}}_i^*\mathbf{P}_i^{-*/2} & \mathbf{y}_i^* \\ \hline \lambda^{-1/2}\mathbf{P}_i^{1/2} & \mathbf{0} \end{array}\right) \Rightarrow \left(\begin{array}{c|c} \mathbf{L}_i & \mathbf{\Gamma}_i \\ \hline \alpha_i^* & \beta_i^* \\ \hline \mathbf{U}_i & \mathbf{\Omega}_i \end{array}\right) \Leftarrow \left(\begin{array}{c|c} \mathbf{P}_{i+1}^{-*/2} & \mathbf{0} \\ \hline \hat{\mathbf{x}}_{i+1}^*\mathbf{P}_{i+1}^{-*/2} & \mathbf{e}_i^*\mathbf{R}_{e,i}^{-*/2} \\ \hline \mathbf{P}_{i+1}^{1/2} & -\overline{\mathbf{K}}_{p,i} \end{array}\right)$$

Un esquema particular se muestra en la figura 5.3

A continuación comentaremos las características geométricas, los requisitos, operacio-

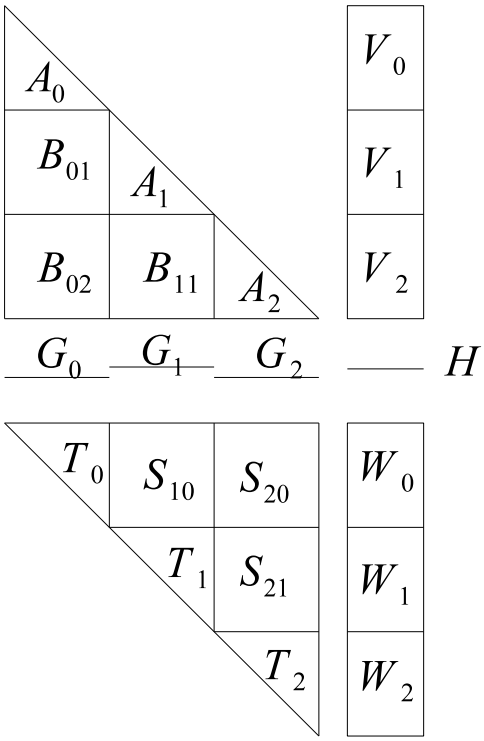

Figura 5.3: Esquema particular de particiones.

nes y datos que generan cada uno de los tipos de particiones propuestos. Identificaremos cada submatriz por la letra que corresponde a su tipo y por uno o varios subíndices:

- Submatriz de tipo A: triangular inferior cuyos vértices de los ángulos no rectos coinciden con elementos diagonales de $\mathbf{L}_i$. Posee $r_{A_j} = c_{A_j}$ filas y columnas. La fila de $\mathbf{L}_i$ que se corresponde con la primera de la submatriz A_j será denotada por $r_{0_A,j}$.

- Submatriz de tipo B: es rectangular y está ubicada en la vertical inferior de una submatriz de tipo A. Si esta submatriz es la A_j, la submatriz B será denotada por B_{jk} donde $k > j$ indicará el orden vertical que ocupa. Poseerá $r_{B_{jk}} = r_{A_{j+k}}$ filas y $c_{B_{jk}} = c_{A_j}$ columnas . La fila de $\mathbf{L}_i$ que se corresponde con la primera de la submatriz B_j será denotada por $r_{0_B,j}$.

- Submatriz de tipo V: rectangular y compuesta por r_{V_j} filas consecutivas de $\mathbf{\Gamma}_i$, comenzando por la $r_{0_V,j}$. Tanto r_{V_j} como $r_{0_V,j}$ coincidirán con los de la submatriz A_j que se encuentra en su horizontal.

- Submatriz de tipo T: triangular superior cuyos vértices de los ángulos no rectos coinciden con elementos diagonales de $\mathbf{U}_i$. Posee $r_{T_j} = c_{T_j} = r_{A_j}$ filas y columnas. La fila de $\mathbf{U}_i$ que se corresponde con la primera de la partición T_j será denotada por $r_{0_T,j}$. Existirán tantas submatrices T como A, por lo que emplearemos los mismos subíndices para las que se encuentren alineadas verticalmente.

- Submatriz de tipo S: rectangular y ubicada en la vertical superior de una partición de tipo T. Si esta submatriz es la T_j, la partición S será denotada por S_{jk} donde $k < j$ indicará el orden vertical que ocupa. Poseerá $r_{S_{jk}} = r_{T_k}$ filas y $c_{S_{jk}} = c_{T_j}$ columnas . La fila de $\mathbf{U}_i$ que se corresponde con la primera de la submatriz S_j será denotada por $r_{0_S,j}$.

- Submatriz de tipo W: partición rectangular compuesta por r_{W_j} filas consecutivas de $\mathbf{\Omega}_i$, comenzando por $r_{0_W,j}$. Tanto r_{W_j} como $r_{0_W,j}$ coincidirán con los de alguna partición T o S que se encuentre en su horizontal.

Las dependencias y co-dependencias entre las operaciones a realizar sobre la descomposición de los datos son las siguientes:

- Submatrices de tipo A. La submatriz A_j presenta co-dependencia con la submatriz V_j para el cálculo de transformaciones ortogonales.

- Submatrices de tipo B: la submatriz B_{jk} presenta co-dependencia con la submatriz V_{j+k} por la aplicación de las transformaciones generadas por la co-dependencia $A_j - V_j$.

- Submatrices de tipo V: una submatriz V_j presenta dependencia de la co-dependencia $A_k - V_k, \forall\, k < j$ y co-dependencia con las submatrices $B_{t,j-u}, \forall\, t < j$ y $\forall\, 1 \leq j-u \leq j$.

- Submatrices de tipo T: la subamtriz T_j presenta dependencia de la co-dependencia $A_j - V_j$ y co-dependencia con W_j.

- Submatriz de tipo S: una submatriz S_{jk} tiene dependencia de la co-dependencia $A_j - V_j$ y co-dependencia con W_k.

- Submatriz de tipo F: una submatriz W_j presenta co-dependencia con todas las submatrices S_{lj}, con $j + 1 \leq l \leq t - 1$, donde t es la cantidad total de submatrices A que existen.

- Las particiones G_j tiene un tratamiento análogo a las B_{jk}

- La partición H tiene un tratamiento análogo a las V.

Gráficamente se muestra en la figura 5.4

Inicialmente hay que resolver la co-dependencia $A_0 - V_0$. Una vez resuelta, se liberan las dependencias con ella de $B_{01} - V_1$, $B_{02} - V2$, $G_0 - H$ y $T_0 - W_0$. El siguiente segmento no puede continuar debido a que la partición V_1 todavía no está actualizada, y así sucesivamente. El máximo número de tareas concurrentes es de cuatro en este ejemplo ($t + 1$ en general, siendo t la cantidad de submatrices de tipo A que hemos dispuesto), aunque la co-dependencia $G_i - H$ tiene un coste pequeño en comparación con las restantes de su mismo nivel. Pero debido a la dependencia de las submatrices C_j con la co-dependencia $B_{kj} - C_j$ del segmento anterior, por el carácter intrínsecamente secuencial del algoritmo, tendremos desocupados a todos los procesadores, salvo uno, durante cierta cantidad de tiempo y además de forma periódica. Por ello, optamos por llevar a cabo una paralelización de manera segmentada, en la que un mismo segmento de cada iteración es asignado al mismo procesador.

El *mapeo* de tareas a procesadores será como sigue:

- Realizaremos tantas particiones generales como procesadores y segmentos de iteración existan.

- Un procesador P_j *ocupará* o *procesará* de manera continua las submatrices A_j, B_{jk}, G_j, T_j y S_{jl}, es decir, un conjunto de columnas consecutivas.

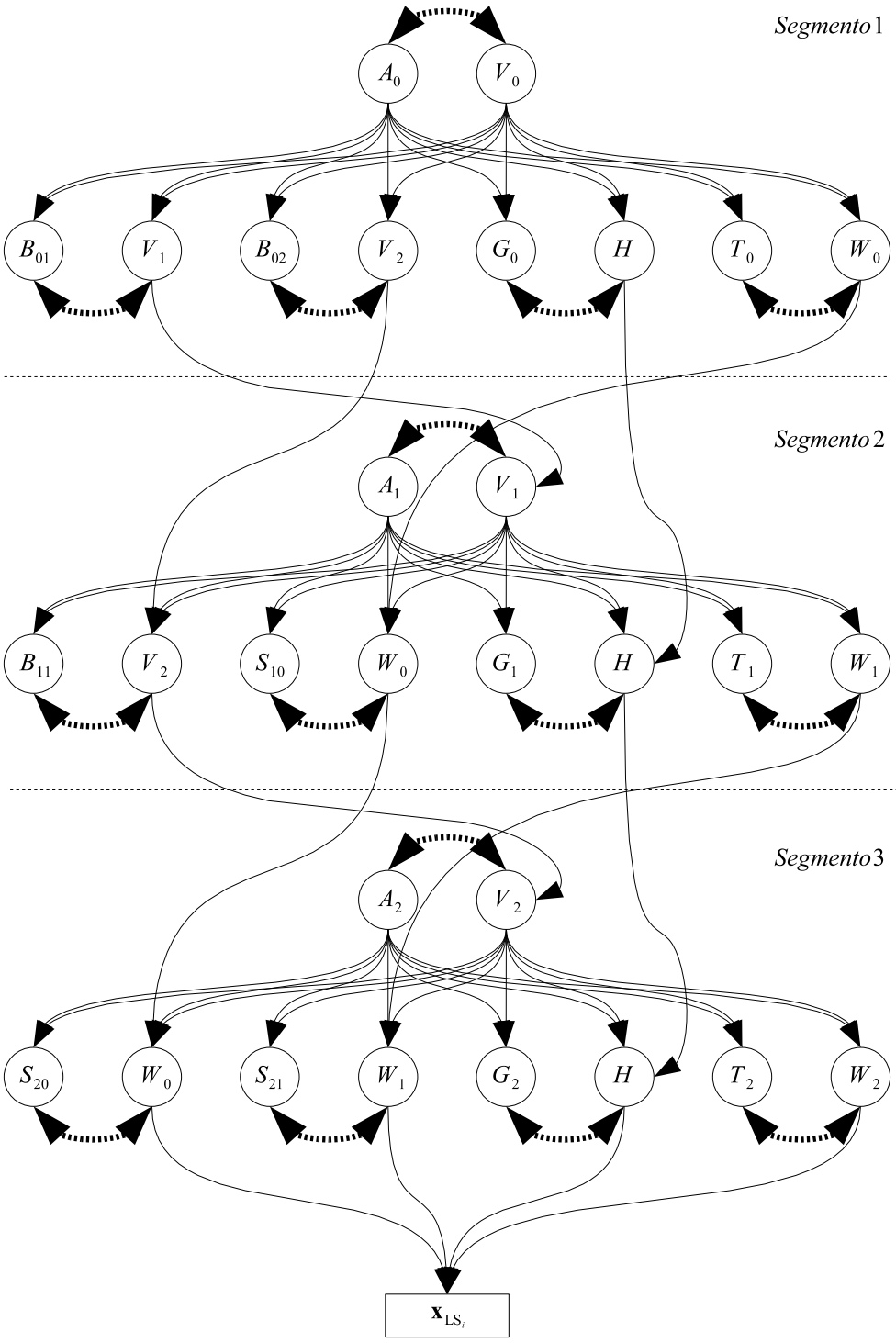

Figura 5.4: Grafo de dependencias.

- Las particiones V, H y W serán ocupadas o procesadas de manera ordenada por todos los procesadores, de modo, que en términos reales un procesador P_j procesará dichas particiones, enviando la parte no nula del resultado de todas estas particiones al procesador P_{j+1}.

En [98] se muestra este algoritmo y en [99] una implementación híbrida `OpenMP+MPI` en un cluster de dos SMP apta para redes heterogéneas.

Por simplicidad y claridad en la exposición, describiremos este algoritmo mediante un ejemplo en el que disponemos de $p = 3$ procesadores: P_0, P_1 y P_2.

Una matriz encerrada entre corchetes con un procesador como subíndice denotará que parte de la matriz en cuestión reside en tal procesador. Si está encerrada entre paréntesis, denotará que la matriz por completo reside en dicho procesador. Las distintas submatrices que aparecerán seguirán la misma notación que en el caso secuencial.

El vector $\hat{\mathbf{x}}_i$ estará (siempre) ubicado en el último procesador (P_2 en este ejemplo). Las n columnas de $\mathbf{\Lambda C}_i$ serán distribuidas entre los procesadores (n_0 columnas residirán en P_0, n_1 columnas en P_1 y n_2 columnas en P_2, con $n_0+n_1+n_2 = n$) y esta distribución de datos no cambiará durante la ejecución del algoritmo paralelo (podría cambiar en una verisón con equilibrado adaptativo de la carga). $\mathbf{\Lambda D}_i$ será manipulada de manera segmentada por todos los procesadores, por lo que el subíndice cambiará de manera oportuna conforme vaya evolucionando la ejecución (estará inicialmente en P_0). Para mostrar más claramente la ejecución del algoritmo, dividiremos $(\mathbf{\Lambda D}_i)_{P_j}$ por filas: las primeras n filas serán divididas en p grupos, así como las n últimas. Un superíndice situado a la izquierda indicará el número de filas de la agrupación. Por lo tanto, la partición inicial de los datos será:

$$\mathbf{E}_i \;=\; \left(\begin{array}{cccc} [\mathbf{\Lambda C}_i]_{P_0} & [\mathbf{\Lambda C}_i]_{P_1} & [\mathbf{\Lambda C}_i]_{P_2} & (\mathbf{\Lambda D}_i)_{P_0} \end{array} \right)$$

$$= \left(\begin{array}{ccc} [\mathbf{\Lambda C}_i]_{P_0} & [\mathbf{\Lambda C}_i]_{P_1} & [\mathbf{\Lambda C}_i]_{P_2} \end{array} \left(\begin{array}{c} {}^{n_0}\left[\lambda^{1/2}\mathbf{H}_i^*\right] \\ {}^{n_1}\left[\lambda^{1/2}\mathbf{H}_i^*\right] \\ {}^{n_2}\left[\lambda^{1/2}\mathbf{H}_i^*\right] \\ \mathbf{y}_i^* \\ {}^{n_0}[\mathbf{0}] \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_2}[\mathbf{0}] \end{array} \right)_{P_0} \right)$$

Más adelante observaremos que el número de elementos no nulos de $(\mathbf{\Lambda D}_i)_{P_j}$ será siempre $q(n+1)$ en cualquier procesador P_j, tanto antes como después del proceso aplicado.

5.2.2. Tareas de los procesadores

Supongamos que P_0 obtiene ceros en las n_0 filas de ${}^{n_0}\left[\lambda^{1/2}\mathbf{H}_i^*\right]$ aplicando una serie de transformaciones unitarias denotadas por $\mathbf{\Theta}_{i,P_0}$:

$$\mathbf{E}_i' = \quad \mathbf{E}_i\mathbf{\Theta}_{i,P_0} = \left(\begin{array}{ccc} [\mathbf{\Lambda C}_i]_{P_0} & [\mathbf{\Lambda C}_i]_{P_1} & [\mathbf{\Lambda C}_i]_{P_2} \end{array} \left(\begin{array}{c} {}^{n_0}\left[\lambda^{1/2}\mathbf{H}_i^*\right] \\ {}^{n_1}\left[\lambda^{1/2}\mathbf{H}_i^*\right] \\ {}^{n_2}\left[\lambda^{1/2}\mathbf{H}_i^*\right] \\ \mathbf{y}_i^* \\ {}^{n_0}[\mathbf{0}] \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_2}[\mathbf{0}] \end{array} \right)_{P_0} \right) \mathbf{\Theta}_{i,P_0}$$

$$= \quad \left(\begin{array}{ccc} [\mathbf{C}_{i+1}]_{P_0} & [\mathbf{\Lambda C}_i]_{P_1} & [\mathbf{\Lambda C}_i]_{P_2} \end{array} \left(\begin{array}{c} {}^{n_0}[\mathbf{0}] \\ {}^{n_1}\left[\lambda^{1/2}\mathbf{H}_i^*\right]' \\ {}^{n_2}\left[\lambda^{1/2}\mathbf{H}_i^*\right]' \\ \mathbf{y}_i^{*'} \\ {}^{n_0}(\mathbf{L}_i) \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_2}[\mathbf{0}] \end{array} \right)_{P_0} \right)$$

Puede observarse que los datos que no pertenezcan a P_0 no están involucrados en los cálculos. n_0 filas de elementos no nulos, ${}^{n_0}(\mathbf{L}_i)$, aparecen por debajo de $\mathbf{y}_i^{*'}$ como consecuencia de la aplicación de $\mathbf{\Theta}_{i,P_0}$. Es relevante el hecho de que $[\mathbf{C}_{i+1}]_{P_0}$ (las primeras n_0 columnas del resultado $\mathbf{E}_i'$) son precisamente las primeras n_0 columnas de la matriz $\mathbf{E}_{i+1}$ (excepto el factor $\mathbf{\Lambda}$). Esto es determinante para obtener un comportamiento segmentado en el trabajo de los procesadores con un mínimo de movimiento de datos de una iteración a la siguiente.

Ahora, si P_0 transfiere ${}^{n_1}\left[\lambda^{1/2}\mathbf{H}_i^*\right]'$, ${}^{n_2}\left[\lambda^{1/2}\mathbf{H}_i^*\right]'$, $\mathbf{y}_i^{*'}$, y ${}^{n_0}(\mathbf{L}_i)$ hacia P_1, entonces en P_1:

$$\mathbf{E}_i'' = \quad \mathbf{E}_i'\mathbf{\Theta}_{i,P_1} = \quad \left(\begin{array}{ccc} [\mathbf{C}_{i+1}]_{P_0} & [\mathbf{\Lambda C}_i]_{P_1} & [\mathbf{\Lambda C}_i]_{P_2} \end{array} \left(\begin{array}{c} {}^{n_0}[\mathbf{0}] \\ {}^{n_1}\left[\lambda^{1/2}\mathbf{H}_i^*\right]' \\ {}^{n_2}\left[\lambda^{1/2}\mathbf{H}_i^*\right]' \\ \mathbf{y}_i^{*'} \\ {}^{n_0}(\mathbf{L}_i) \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_2}[\mathbf{0}] \end{array} \right)_{P_1} \right) \mathbf{\Theta}_{i,P_1}$$

$$= \left(\begin{array}{ccc} [\mathbf{C}_{i+1}]_{P_0} & [\mathbf{C}_{i+1}]_{P_1} & [\mathbf{\Lambda C}_i]_{P_2} \end{array} \right. \left. \begin{pmatrix} {}^{n_0}[\mathbf{0}] \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_2}\left[\lambda^{1/2}\mathbf{H}_i^*\right]'' \\ \mathbf{y}_i^{*''} \\ {}^{n_0}(\mathbf{L}_i)' \\ {}^{n_1}(\mathbf{M}_i) \\ {}^{n_2}[\mathbf{0}] \end{pmatrix}_{P_1} \right)$$

Mientras P_1 obtiene ceros en las n_1 filas de ${}^{n_1}\left[\lambda^{1/2}\mathbf{H}_i^*\right]'$ con la aplicación de la secuencia de transformaciones unitarias $\mathbf{\Theta}_{i,P_1}$, P_0 puede ya trabajar con $[\mathbf{C}_{i+1}]_{P_0}$, suponiendo que nuevos datos de entrada ($\mathbf{H}_{i+1}$ y $\mathbf{y}_{i+1}$) están ya disponibles (algoritmo segmentado). De nuevo, si P_1 transfiere ${}^{n_2}\left[\lambda^{1/2}\mathbf{H}_i^*\right]''$, $\mathbf{y}_i^{*''}$, ${}^{n_0}(\mathbf{L}_i)'$ y ${}^{n_1}(\mathbf{M}_i)$ (la parte no nula de la matriz $(\mathbf{\Lambda D}_i)_{P_1}$ procesada —$q(n+1)$ números en coma flotante—) hacia P_2, entonces P_2 puede obtener ceros en las n_2 filas de ${}^{n_2}\left[\lambda^{1/2}\mathbf{H}_i^*\right]''$ con la aplicación de la secuencia de transformaciones unitarias $\mathbf{\Theta}_{i,P_2}$. Simultáneamente, P_1 podría trabajar con $[\mathbf{C}_{i+1}]_{P_1}$ y $(\mathbf{\Lambda D}_{i+1})_{P_1}$, y P_0 podría trabajar con $[\mathbf{C}_{i+2}]_{P_0}$ suponiendo que nuevos datos de entrada están ya disponibles (comportamiento segmentado de nuevo):

$$\mathbf{E}_i''' = \mathbf{F}_i = \mathbf{E}_i''\mathbf{\Theta}_{i,P_2}$$

$$= \left(\begin{array}{ccc} [\mathbf{C}_{i+1}]_{P_0} & [\mathbf{C}_{i+1}]_{P_1} & [\mathbf{\Lambda C}_i]_{P_2} \end{array} \right. \left. \begin{pmatrix} {}^{n_0}[\mathbf{0}] \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_2}\left[\lambda^{1/2}\mathbf{H}_i^*\right]'' \\ \mathbf{y}_i^{*''} \\ {}^{n_0}(\mathbf{L}_i)' \\ {}^{n_1}(\mathbf{M}_i) \\ {}^{n_2}[\mathbf{0}] \end{pmatrix}_{P_2} \right) \mathbf{\Theta}_{i,P_2}$$

$$
\begin{aligned}
&= \left(\begin{array}{ccc} [\mathbf{C}_{i+1}]_{P_0} & [\mathbf{C}_{i+1}]_{P_1} & [\mathbf{C}_{i+1}]_{P_2} \end{array} \left(\begin{array}{c} {}^{n_0}[\mathbf{0}] \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_2}[\mathbf{0}] \\ \mathbf{y}_i^{*'''} \\ {}^{n_0}(\mathbf{L}_i)'' \\ {}^{n_1}(\mathbf{M}_i)' \\ {}^{n_2}(\mathbf{N}_i) \end{array} \right)_{P_2} \right) \\
&= \left(\begin{array}{ccc} [\mathbf{C}_{i+1}]_{P_0} & [\mathbf{C}_{i+1}]_{P_1} & [\mathbf{C}_{i+1}]_{P_2} \end{array} \left(\begin{array}{c} {}^{n_0+n_1+n_2=n}(\mathbf{0}) \\ \mathbf{e}_i^* \mathbf{R}_{e,i}^{-*/2} \\ -\overline{\mathbf{K}}_{p,i} \end{array} \right)_{P_2} \right)
\end{aligned}
$$

Consecuentemente, la expresión (5.1) es entonces obtenida mediante

$$\mathbf{E}_i \mathbf{\Theta}_i = \mathbf{E}_i \mathbf{\Theta}_{i,P_0} \mathbf{\Theta}_{i,P_1} \mathbf{\Theta}_{i,P_2} = \mathbf{F}_i$$

de forma segmentada. En este momento, P_2 puede actualizar la solución $\hat{\mathbf{x}}_0$ (o bien $\mathbf{x}_{LS}$) como:

$$\hat{\mathbf{x}}_{i+1} = \lambda^{-1/2} \hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i} \left[\mathbf{R}_{e,i}^{-1/2} \mathbf{e}_i \right] \quad \text{and} \quad \mathbf{x}_{\mathrm{LS}_i} = (\lambda^{1/2})^{i+1} \hat{\mathbf{x}}_{i+1}$$

5.2.3. Pseudocódigo

El **Algoritmo 5** muestra el pseudocódigo del algoritmo paralelo propuesto. Sea n_j el número de columnas de $\mathbf{C}_i$ asignadas al procesador P_j y c_{0_j} el índice de la primera de ellas. Una flecha simple indicará una operación de asignación y una flecha doble, una transferencia de información hacia/desde el siguiente/previo procesador. Las transferencias de datos de P_j hacia P_{j+1} ìmplica sólo la parte no nula de la matriz y se supone que en el destino es ubicada en las posiciones oportunas. La figura 5.5 ilustra gráficamente el algoritmo paralelo segmentado propuesto.

Algoritmo 5 RLS paralelo: raíz cuadrada para el Filtro de Información

Entrada: $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$ e $\mathbf{y} = (\mathbf{y}_0^*, \mathbf{y}_1^*, \ldots)^*$ en P_0, $\hat{\mathbf{x}}_0$ en P_{p-1}, y $\mathbf{P}_0^{1/2}$
Salida: $\mathbf{x}_{\text{LS}}$ cada iteración i-ésima, en P_{p-1}

1: **para todo** $P_j, j = 0, \ldots, p-1:$ en P_j **hacer**
2: $\quad \mathbf{\Lambda} = \begin{pmatrix} \lambda^{1/2}\mathbf{I}_n & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \lambda^{-1/2}\mathbf{I}_n \end{pmatrix}$
3: $\quad \mathbf{C}_0 = \begin{pmatrix} \mathbf{P}_0^{-*/2} \\ \hat{\mathbf{x}}_0^*\mathbf{P}_0^{-*/2} \\ \mathbf{P}_0^{1/2} \end{pmatrix}_{P_j}$
4: $\quad$ **para** $i = 0, \ldots, m/q - 1$ **hacer**
5: $\quad\quad$ **si** $P_j == P_0$ **entonces**
6: $\quad\quad\quad \left(\mathbf{\Lambda}\mathbf{D}_i^-\right)_{P_j} \leftarrow \mathbf{\Lambda} \begin{pmatrix} \mathbf{H}_i^* \\ \mathbf{y}_i^* \\ \mathbf{0} \end{pmatrix}$
7: $\quad\quad$ **sino**
8: $\quad\quad\quad \left(\mathbf{\Lambda}\mathbf{D}_i^-\right)_{P_j} \Leftarrow \left(\mathbf{\Lambda}\mathbf{D}_i^+\right)_{P_{j-1}}$
9: $\quad\quad$ **fin si**
10: $\quad\quad$ **para** $r = c_{0_j} : c_{0_j} + n_j - 1$ **hacer**
11: $\quad\quad\quad$ Calcular $\mathbf{\Theta}_{i,P_j}$
12: $\quad\quad\quad$ Aplicar $\mathbf{\Theta}_{i,P_j}$ to $\left([\mathbf{\Lambda}\mathbf{C}_i]_{P_j}, \ \left(\mathbf{\Lambda}\mathbf{D}_i^-\right)_{P_j}\right)$
13: $\quad\quad$ **fin para**
14: $\quad\quad$ $[\mathbf{C}_{i+1}]_{P_j} \leftarrow [\mathbf{\Lambda}\mathbf{C}_i]_{P_j}$
15: $\quad\quad$ $\left(\mathbf{\Lambda}\mathbf{D}_i^+\right)_{P_j} \leftarrow \left(\mathbf{\Lambda}\mathbf{D}_i^-\right)_{P_j}$
16: $\quad\quad$ **si** $P_j \neq P_{p-1}$ **entonces**
17: $\quad\quad\quad \left(\mathbf{\Lambda}\mathbf{D}_i^+\right)_{P_j} \Rightarrow \left(\mathbf{\Lambda}\mathbf{D}_i^-\right)_{P_{j+1}}$
18: $\quad\quad$ **sino**
19: $\quad\quad\quad \begin{pmatrix} \mathbf{e}_i^*\mathbf{R}_{e,i}^{-*/2} \\ -\overline{\mathbf{K}}_{p,i} \end{pmatrix} \leftarrow \left(\mathbf{\Lambda}\mathbf{D}_i^+\right)_{P_j} ([n+1 : 2n+1], :)$
20: $\quad\quad\quad$ Actualizar la estimación del estado $\hat{\mathbf{x}}_{i+1} \leftarrow \lambda^{-1/2}\hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i}\mathbf{R}_{e,i}^{-1/2}\mathbf{e}_i$
21: $\quad\quad\quad$ Actualizar a solución LS $\mathbf{x}_{\text{LS}_i} \leftarrow \left(\lambda^{1/2}\right)^{i+1}\hat{\mathbf{x}}_{i+1}$
22: $\quad\quad$ **fin si**
23: $\quad$ **fin para**
24: **fin para**

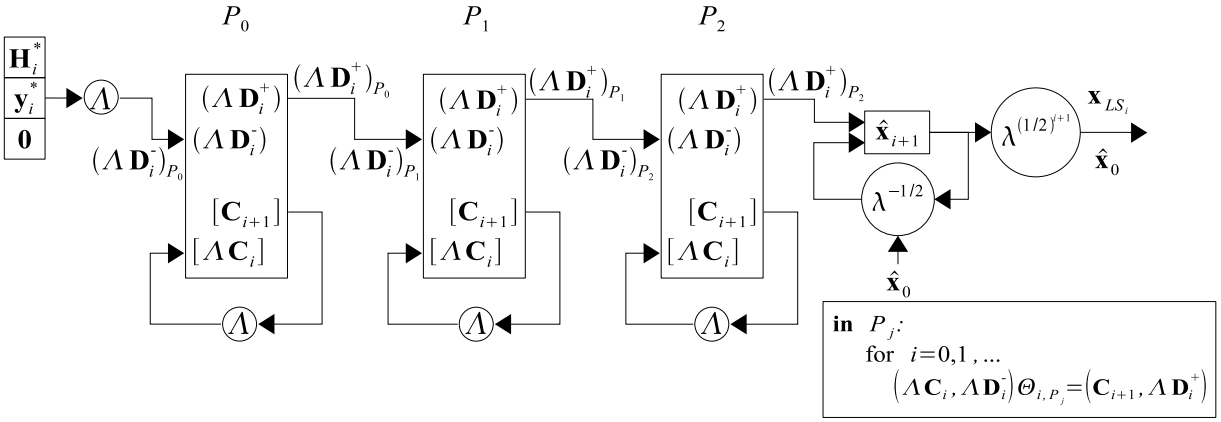

Figura 5.5: Algoritmo paralelo segmentado raíz cuadrada para el Filtro de Información con $p = 3$.

La figura 5.6 muestra el estado del algoritmo paralelo y su evolución cuando se está extrayendo la solución correspondiente al instante i-ésimo.

5.2.4. Coste aritmético

Para calcular el coste debido exclusivamente a operaciones aritméticas, primeramente evaluaremos el coste aritmético por iteración, para después multiplicarlo por el número de iteraciones.

Las contribuciones en el coste aritmético por iteración son las siguientes:

- El escalado de $[\mathbf{\Lambda C}_i]_{P_j}$ requiere $n_j n + n$ flops, sólo si $\lambda \neq 1$.
- El procesador P_j debe aplicar la misma secuencia de transformaciones de Givens y Householder que el algoritmo secuencial, pero sólo a las columnas $1, \ldots n_j$ de $[\mathbf{\Lambda C}_i]_{P_j}$ y a las filas $c_{0_j}, \ldots, c_{0_j} + n_j - 1$ de $(\mathbf{\Lambda D}_i)_{P_j}$ respectivamente, por lo que el coste de las líneas 10–13 del pseudocódigo es

$$\left[w_{CH}(q) + w_{CG} + w_{AH}(q)(n+1) + w_{AG}(n+2)\right] n_j \quad \text{flops}$$

- Sólo el procesador P_{p-1} realiza la actualización del estado, luego exclusivamente para él este coste es según (4.6)

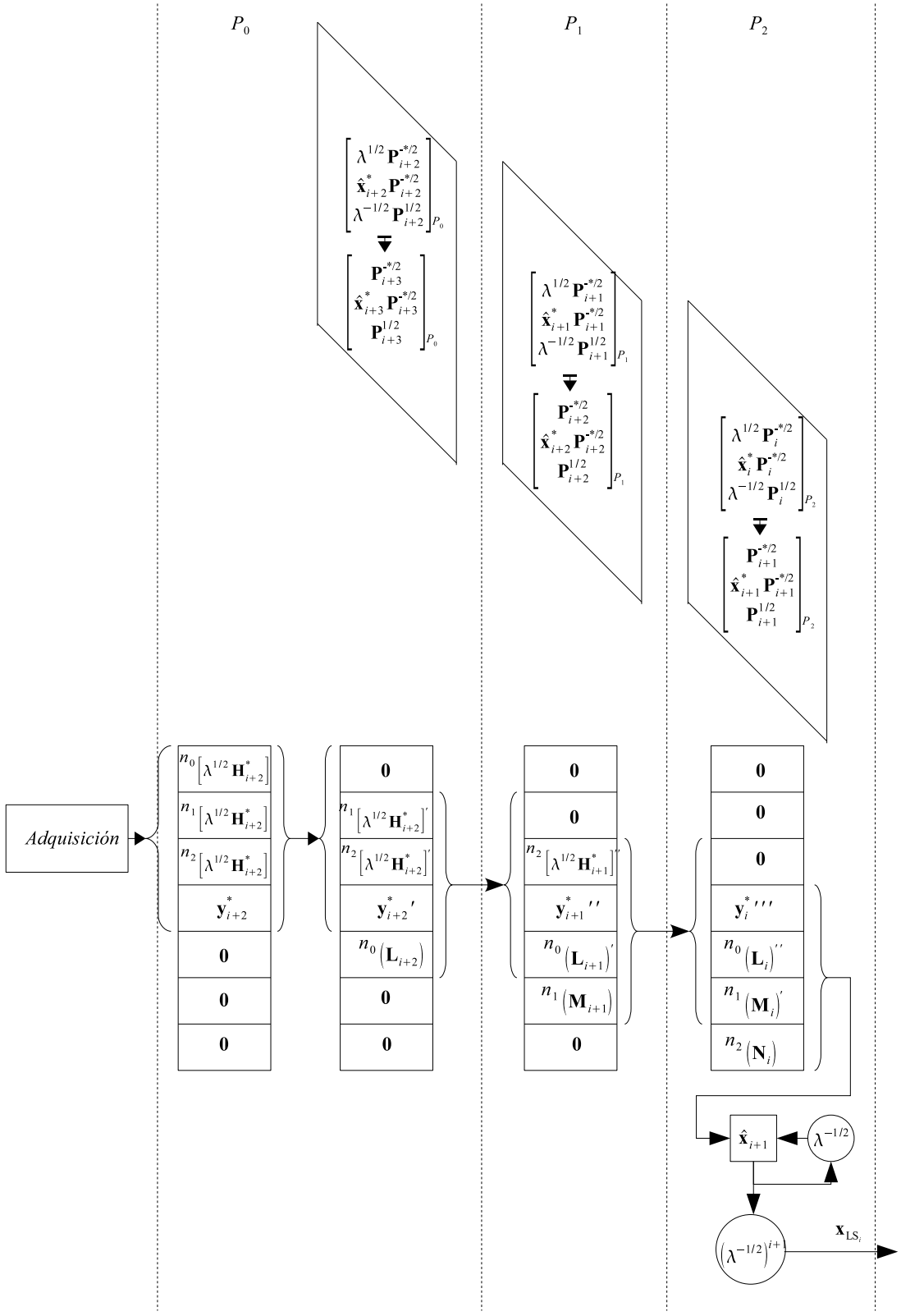

Figura 5.6: Estado y evolución del algoritmo SRIF para la solución $\mathbf{x}_{\mathrm{LS}_i}$, con $p = 3$.

$$4qn + q^2 \quad \text{flops} \tag{5.8}$$

más $2n + 1$ flops si $\lambda \neq 1$.

Por lo que el coste por iteración en el procesador P_j será:

$$\begin{aligned} W_{P_j,i}(z)\big|_{\lambda=1} &= [w_{CH}(q) + w_{CG} + w_{AH}(q)(n+1) + w_{AG}(n+2)]\, n_j \\ W_{P_j,i}(z)\big|_{\lambda\neq 1} &= n_j n + n + W_{P_j,i}(z)\big|_{\lambda=1} \end{aligned} \tag{5.9}$$

Si $P_j = P_{p-1}$ hay que añadir el coste de la actualización del estado y la solución (5.8). De estas expresiones, y obviando el tiempo de llenado o vaciado de la línea segmentada, se puede suponer que el tiempo aritmético por iteración es independiente del índice de esta, por lo que el tiempo aritmético total del algoritmo paralelo es:

$$T_A(p,z) \approx \frac{m}{q} \max_{j=0,\ldots,p-1} \{W_{P_j,i}(z) t_{w_j}\} \tag{5.10}$$

5.2.5. Equilibrado de la carga

Nos centraremos en el caso en el que $\lambda = 1$. Si despreciamos el coste de la actualización del estado/solución (5.8), entonces a partir de (2.8) y (5.9) :

$$\begin{aligned} [w_{CH}(q) + w_{AH}(q)(n+1) + w_{CG} + w_{AG}(n+2)]\, n_j &= \\ [w_{CH}(q) + w_{AH}(q)(n+1) + w_{CG} + w_{AG}(n+2)]\, n\alpha_j & \end{aligned}$$

por lo que

$$n_j = n\alpha_j$$

quedando como sobrecarga aritmética el coste de la actualización en el último procesador (5.8).

Para esta situación de este algoritmo, vemos que no es necesario plantear una potencial mejora en el equilibrado al utilizar una línea segmentada bidireccional.

5.2.6. Coste de las comunicaciones

Si $P_j \neq P_{p-1}$, entonces $q(n+1)$ elementos deben ser transferidos al siguiente procesador P_{j+1} cada iteración del algoritmo paralelo. Supongamos que un modelo afín ajusta el tiempo de comunicación punto a punto de un procesador al siguiente, independientemente de la arquitectura que subyazca. Supongamos que esta *comunicación al siguiente procesador* $T_{\text{NPC}}(z)$ conlleva un tiempo, según el modelo lineal, de

$$T_{\text{NPC}}(z) = \beta + \tau[q(n+1)] \approx \beta + \tau qn$$

segundos, independientemente del índice del procesador j.

El número total de transferencias necesarias cuando una iteración finaliza depende del valor de la iteración. Tal y como hemos venido haciendo hasta ahora, obviaremos los instantes en los que la línea segmentada se está llenando o vaciando, por lo tanto el tiempo del algoritmo paralelo dedicado a la comunicación será el de una iteración multiplicado por el número total de éstas, es decir, m/q.

Para cada etapa o iteración, las transferencias de un procesador al siguiente, pueden llevarse a cabo de manera simultánea o en serie, dependiendo de las características de la red empleada. Si pueden realizarse simultáneamente (modelo de comunicación A), el tiempo total empleado será el número de etapas multiplicado por el tiempo de comunicación de cualquier transferencia, por lo que:

$$
\begin{aligned}
T_C^{(A)}(z,p) &\approx \frac{m}{q} T_{\mathrm{NPC}}(z) \\
&\approx \frac{m}{q}\beta + \tau mn \quad \mathrm{s} \qquad (5.11) \\
&= \Theta(mn) \qquad (5.12)
\end{aligned}
$$

En el caso de que deban realizarse en serie (definiremos esta situación como modelo de comunicación B), el tiempo total empleado en dichas transferencias de información es:

$$
\begin{aligned}
T_C^{(B)}(z,p) &= \frac{m}{q}(p-1) T_{\mathrm{NPC}}(z) \\
&= \frac{m}{q}(p-1)\beta + (p-1)\tau mn \quad \mathrm{s} \qquad (5.13) \\
&= \Theta(pmn)
\end{aligned}
$$

El tiempo de comunicaciones que influirá de manera efectiva en el tiempo paralelo dependerá del grado de solapamiento entre el cálculo y las comunicaciones; el caso más desfavorable sería que dicho grado de solapamiento fuera nulo y por lo tanto, de manera íntegra estos tiempos debieran incluirse en el cómputo total.

5.2.7. Escalabilidad

Existen dos fuentes de sobrecarga en el algoritmo: por una parte el tiempo destinado a las comunicaciones y por otra la actualización de la solución que sólo se lleva a cabo en el último procesador. El orden de magnitud de laa sobrecarga aritmética en la actualización, $\Theta(qn)$, (5.8), resulta despreciable frente al de las comunicaciones, tanto para el modelo A como el B, por lo que la escalabilidad estaría entre $n = \Theta(p)$ y $n = \Theta(p^2)$ en función del modelo de comunicación que mejor ajuste.

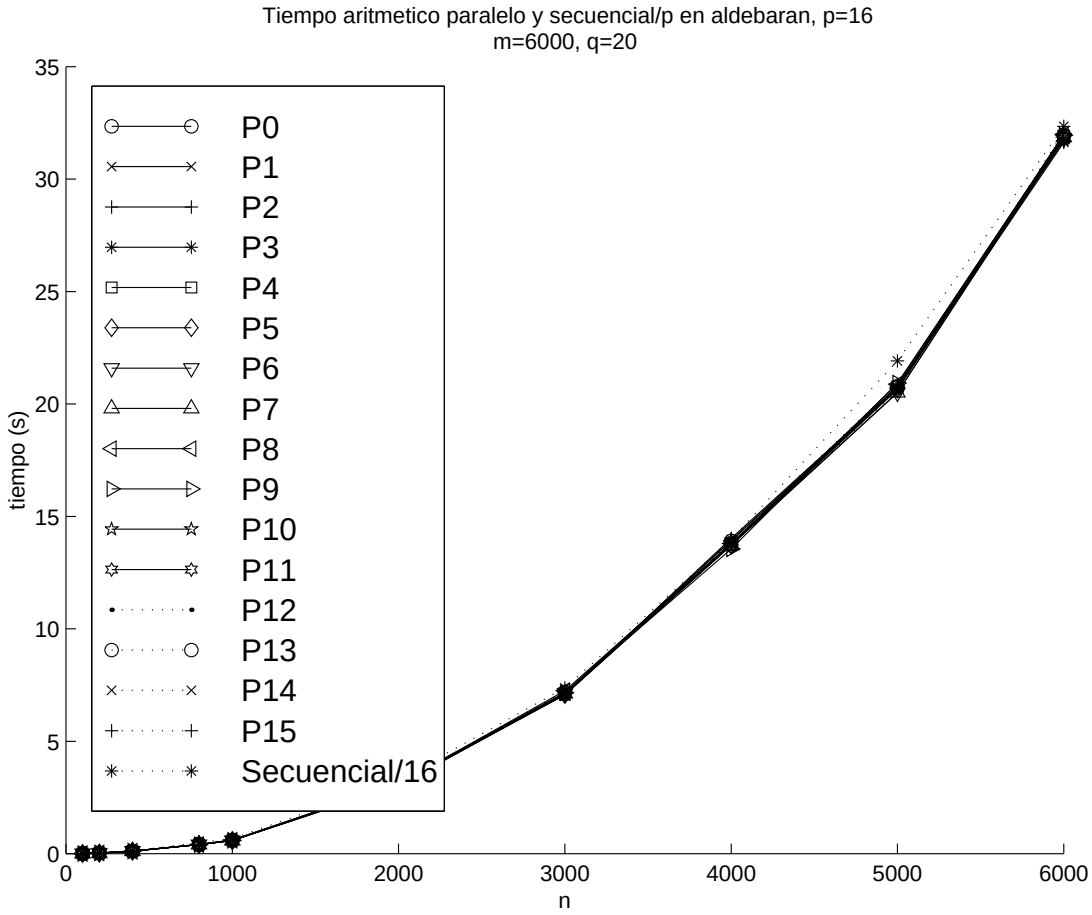

Figura 5.7: Tiempo aritmético paralelo por procesador y tiempo secuencial dividido por el número de procesadores, con $q = 20$ y $p = 16$

5.2.8. Resultados experimentales

Los siguientes resultados han sido obtenidos ejecutando los tests en la máquina ccNUMA *Aldebarán*. Los denominadores comunes a todas las pruebas, a excepción del test de una versión híbrida memoria distribuida-memoria compartida, han sido un total de filas $m = 6000$ y un valor de 1 para el parámetro λ.

Equilibrado de la carga

En la figura 5.7 se observa un buen resultado de equilibrado de carga para 16 procesadores.

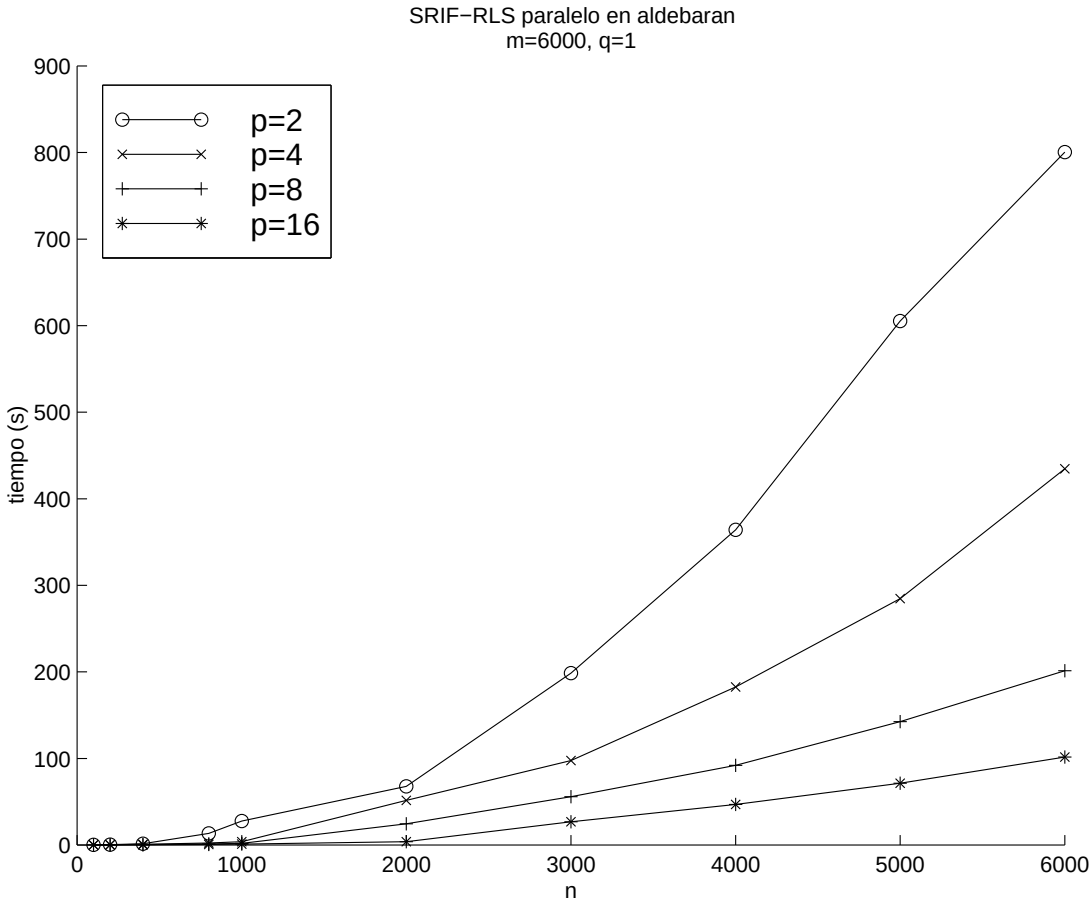

Figura 5.8: Tiempo de ejecución del algoritmo paralelo en función de n con $q = 1$.

Tiempos de ejecución paralela

La figura 5.8 muestra los tiempos de ejecución paralela del algoritmo para $q = 1$. Para el resto de valores de q no mostrados, los resultados apreciados son los esperados.

Tiempos de sobrecarga

En la figura 5.9 mostramos el tiempo de sobrecarga, estimado como la diferencia entre el tiempo de ejecución paralela y el tiempo de ejecución del algoritmo secuencial dividido por el número de procesadores.

Debido a la variabilidad de las medidas, es difícil determinar cuál es el comportamiento general de este tiempo de sobrecarga. En cualquier caso, parece razonable deducir cualitativamente que dicho comportamiento no parece ser cuadrático, por lo que podemos

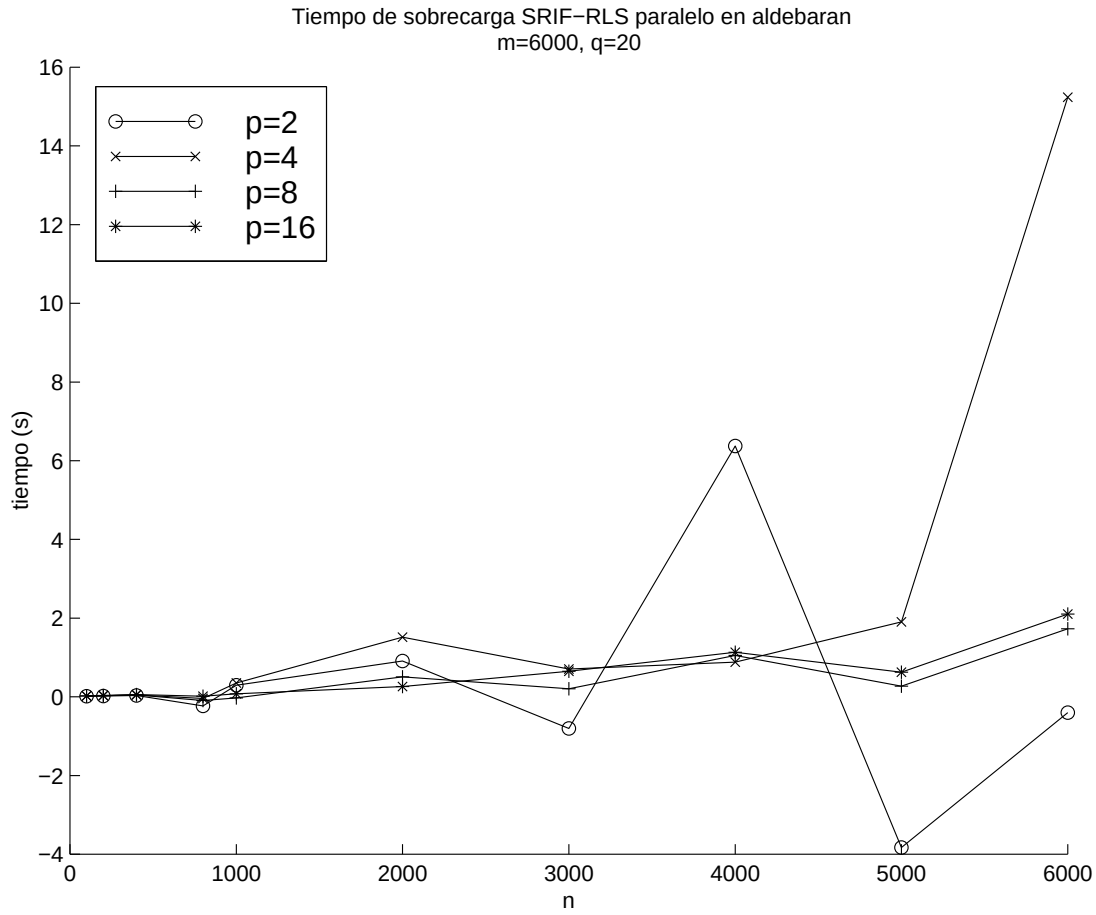

Figura 5.9: Tiempo de sobrecarga en función de n con $q = 20$.

suponerle un carácter lineal. Las fuentes de sobrecarga en el algoritmo paralelo eran la actualización de la solución (en el último procesador) y el tiempo de comunicación, luego estas son las aportaciones más versosímiles a este tiempo.

Eficiencia

Las figuras 5.10 y 5.11 muestran la eficiencia del algoritmo paralelo para $q = 1$ y $q = 20$. Primeramente se observa un gran pico de superlinealidad debido al efecto de la memoria *cache* para $q = 1$, que ya no es tan notable para $q = 20$, debido a que la cantidad de datos a tratar aumenta cuando lo hace q. Observamos que la eficiencia para $q = 1$ es bastante independiente del número de procesadores oscilando entre el 80 y 90 % aproximadamente (en nuestra opinión, la oscilación es debida a la fluctuación aleatoria en la toma de tiempos paralelos y sobre todo secuenciales). Para $q = 20$ ya observamos cómo la eficiencia es dependiente del número de procesadores, disminuyendo, en términos generales, con el aumento del número de procesadores, tal y como resulta razonable.

Eficiencia heterogénea

Un sistema heterogéneo fue emulado basándonos en el homogéneo disponible, asignando carga extra aritifical a algunos procesadores. Esta carga artifical consistió en la repetición dos veces del código dentro de la iteración del algoritmo en sólo la mitad de los procesadores. De esta forma, éstos eran dos veces más lentos para cualquier valor de n. De manera que el tiempo relativo por operación elemental en un sistema de p procesadores fue:

$$t_{w_j} = \begin{cases} 1, & j = 0, \ldots p/2 - 1 \\ 2, & j = p/2, \ldots, p - 1 \end{cases} \tag{5.14}$$

La eficiencia para el sistema paralelo heterogéneo se muestra en la figure 5.12. Esta eficiencia fue calculada como el cociente entre el incremento de velocidad y el número de procesadores. Utilizando (5.14), (2.5) y (2.6) el máximo incremento de velocidad en este

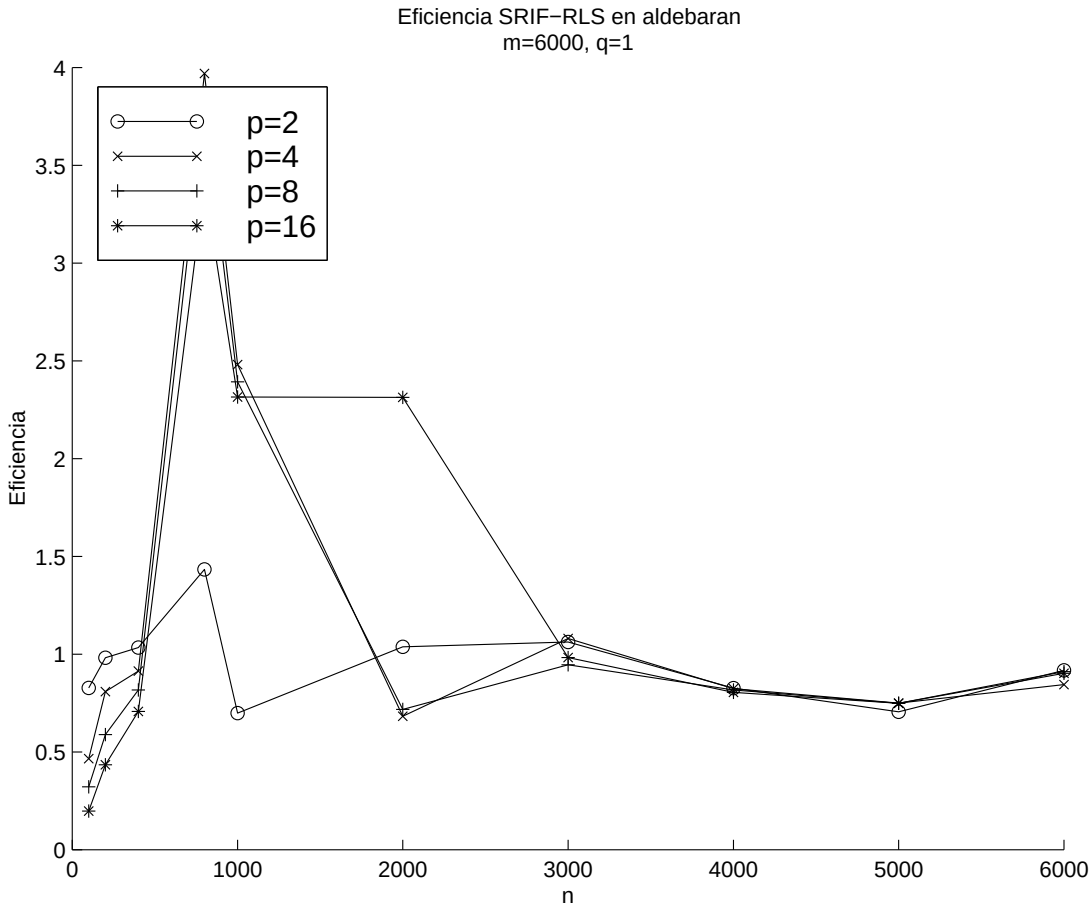

Figura 5.10: Eficiencia en función de n con $q = 1$.

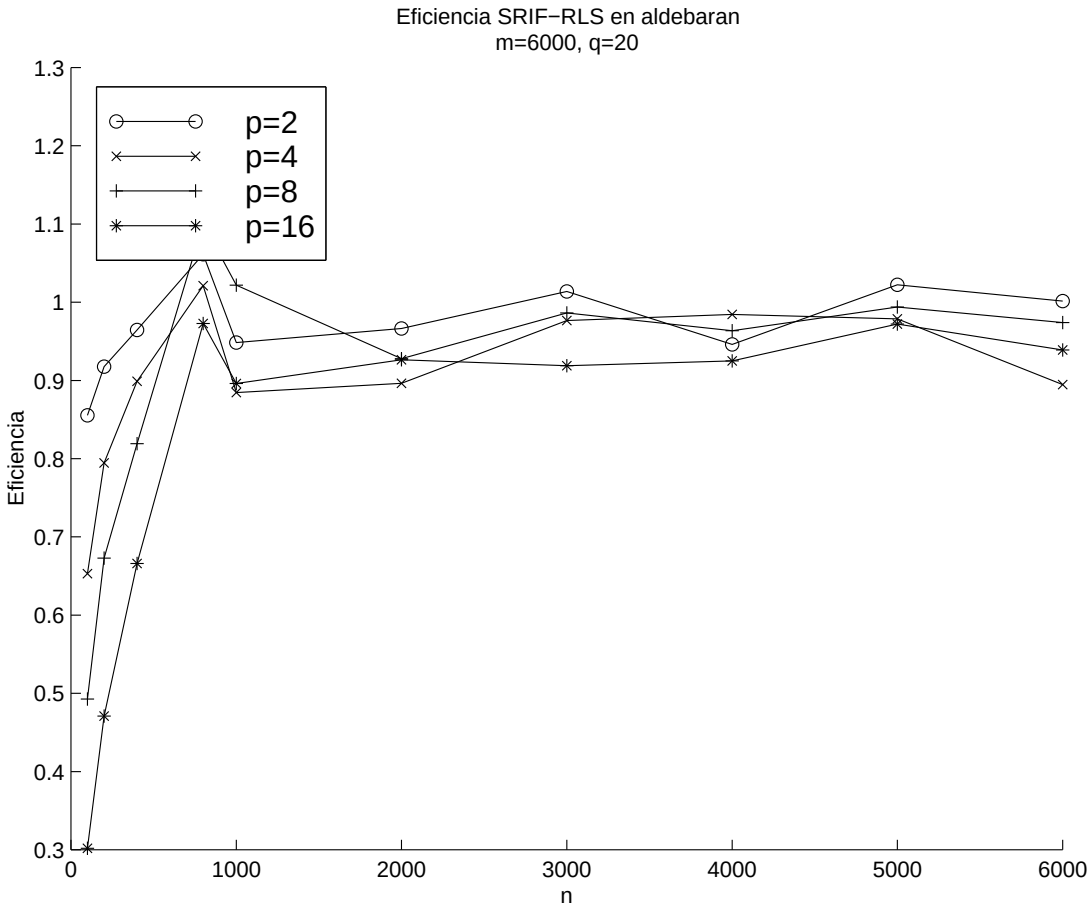

Figura 5.11: Eficiencia en función de n con $q = 20$.

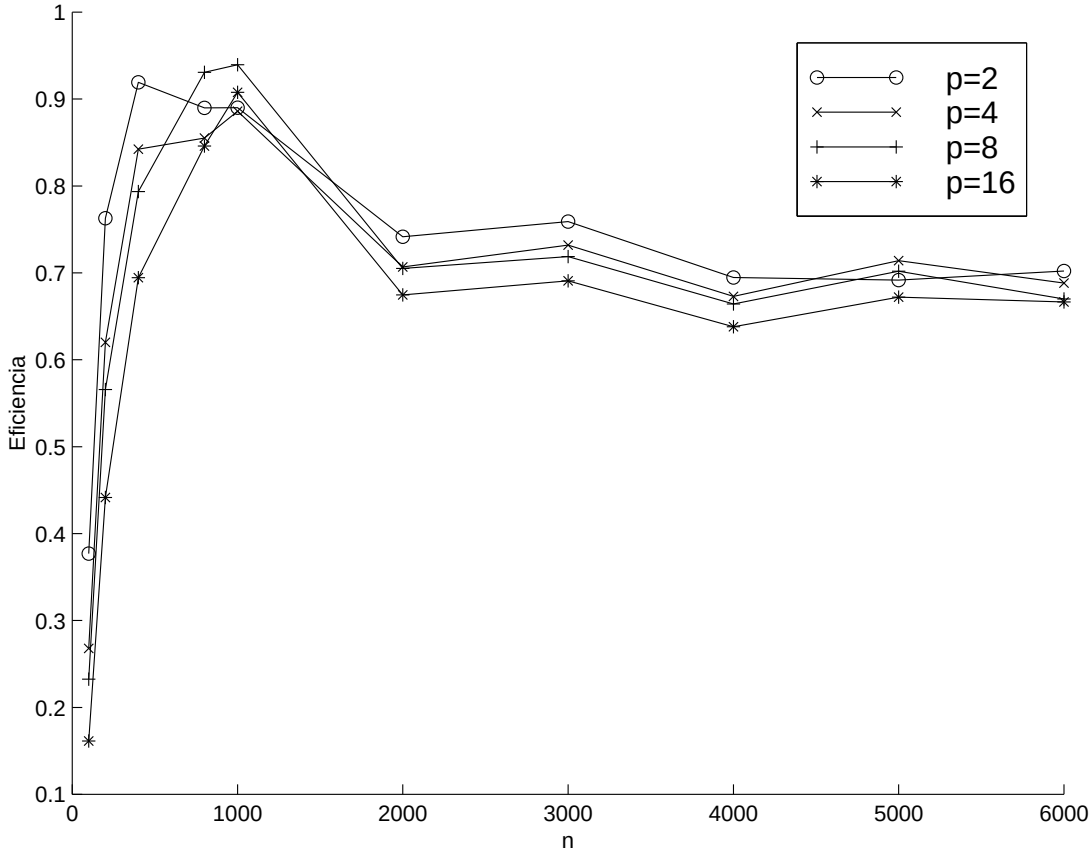

Figura 5.12: Eficiencia homogénea del sistema paralelo heterogéneo con $q = 20$.

sistema paralelo heterogéneo es:

$$S_{\text{máx}}(p, s) = \frac{3}{4}p$$

Por lo que la máxima eficiencia se sitúa en el 75 %. En esta figura, la eficiencia ronda el 70 %. Si calculamos ahora la eficiencia como el ratio entre el incremento de velocidad y el máximo incremento de velocidad (eficiencia heterogénea) entonces obtenemos resultados similares al caso del sistema paralelo homogéneo (alrededor del 90 %).

Versión híbrida memoria distribuida-memoria compartida

Realizamos una implementación mezclando el paradigma de memoria distribuida y el de memoria compartida, tomando como referencia el algoritmo diseñado para la versión distribuida y emulando la comunicación en memoria compartida, con las ideas expuestas en

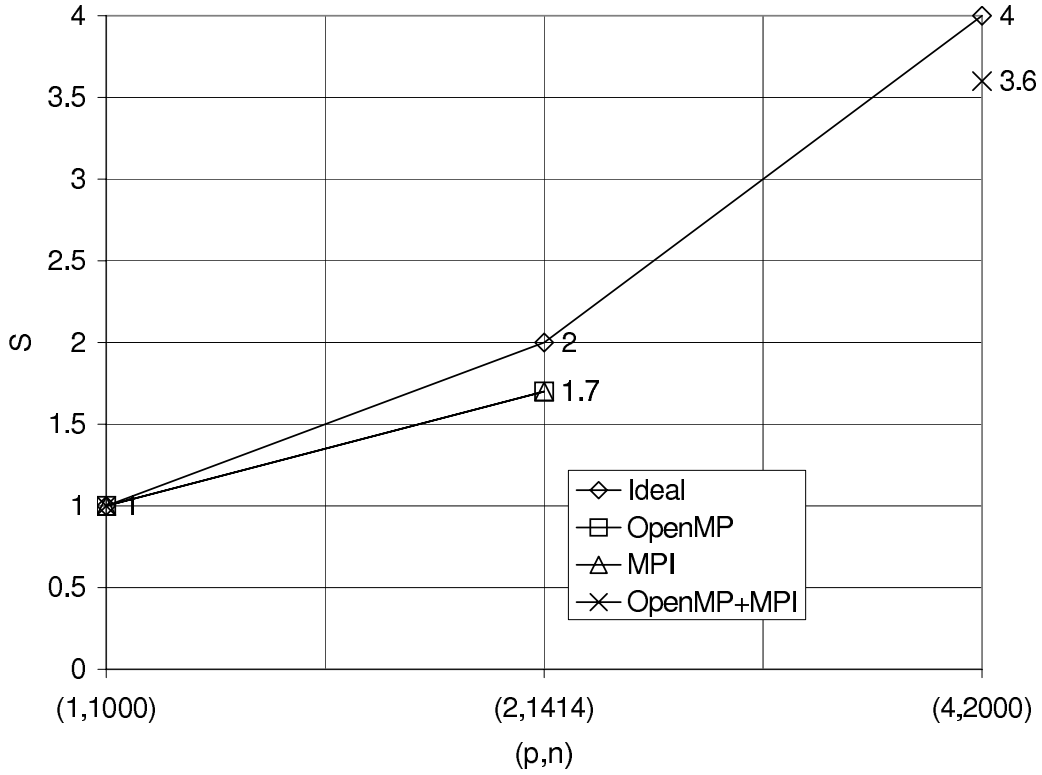

Figura 5.13: Incremento de velocidad para la versión híbrida con $q = 1$ y $\lambda = 0{,}99999$.

la subsección 2.2.2, [98]. La prueba se realizó en un cluster con dos nodos SMP (*Kubrick*), descrito en la subsección 2.1.4. La figura 5.13 muestra el incremento de velocidad para varias combinaciones. memoria distribuida-memoria compartida.

5.3 Conclusiones y posibles mejoras

Hemos desarrollado un algoritmo paralelo con una aceptable eficiencia para arquitecturas con memoria distribuida, compartida y mixta. Debido a la forma en la que tanto las operaciones aritméticas como las comunicaciones pueden ser organizadas, el equilibrado de carga propuesto tanto en un sistema homogéneo como en uno heterogéneo es extremadamente simple y eficaz, obteniendo unos rendimientos similares. No cabe, por tanto, el plantear para este algoritmo un esquema segmentado bidireccional con el que tratar de reequilibrar la carga tal y como se hizo para el algoritmo SRKF estudiado en el capítulo anterior. En cualquier caso puede ser interesante plantear un esquema de equilibrado

adaptativo con el que realizar un *reequilibrado* cuando la carga de cada procesador pueda variar con el tiempo.

Las rutinas de transmisión de información empleadas son de carácter asíncrono, pudiendo ser interesante el estudio del comportamiento de los algoritmos cuando dicho carácter es síncrono.

Debido a la versión *no thread safe* utilizada para MPI, el acceso a las comunicaciones dentro de la rutina está dentro de secciones críticas en ambos hilos. Proponemos como una futura mejora el empleo de una versión *thread safe* con la que intentar aumentar algo más la eficiencia, debido a este inconveniente.

Tanto para el algoritmo secuencial como para el paralelo, podemos plantearnos la estimación del valor óptimo del parámetro q para conseguir el mínimo tiempo de ejecución. Igualmente que en el algoritmo SRKF del capítulo anterior, esta estimación requiere bien de un modelo de tiempo de ejecución preciso o bien el empleo de métodos heurísticos.

6

Algoritmo RLS basado en la actualización de la factorización QR

En el presente capítulo mostraremos los detalles de implementación del algoritmo basado en la actualización de la factorización QR que resuelve el problema RLS. Los detalles del algoritmo ya fueron mostrados en la sección 3.6. Mostraremos también la relación existente con filtro de información cuando pretendemos resolver el mismo problema. Por último evaluaremos las prestaciones del algoritmo paralelo, justificando el comportamiento del algoritmo e interpretando los resultados; asimismo, nos plantearemos futuras líneas de estudio que intenten mejorar los problemas de los que adolezcan las implementaciones.

6.1 Algoritmo secuencial

El algoritmo secuencial se basa en repetir la iteración mostrada en el **Algoritmo** 1, página 71, hasta cubrir el proceso de la totalidad de las filas de la matriz problema. Emplearemos el acrónimo QRU-RLS (*QR Updating - Recursive Least Squares*) para nombrar a este algoritmo.

6.1.1. Detalles de implementación y costes

La anulación de la matriz $\mathbf{W}_i$ puede llevarse a cabo de múltiples formas, con distintos rendimientos. En primera instancia, debemos plantearnos si es más conveniente hacer estos ceros en $\mathbf{W}_i$ fila a fila o columna a columna. Teniendo en cuenta que las transformaciones deben aplicarse por la izquierda, el hacer ceros fila a fila obliga a emplear exclusivamente rotaciones de Givens; el hacer ceros columna a columna permite elegir entre rotaciones de Givens, transformaciones de Householder o ambas. Por ello, supondremos de momento que anularemos $\mathbf{W}_i$ columna a columna.

Podemos emplear rotaciones de Givens para ir anulando los elementos de la columna en cuestión de $\mathbf{W}_i$ y finalizar con una última rotación respecto al elemento diagonal oportuno de $\mathbf{R}_{i,a}$. Alternativamente, podemos emplear transformaciones de Householder, acumulándose las reflexiones de las columnas de $\mathbf{W}_i$ en los respectivos elementos diagonales de $\mathbf{R}_{i,a}$, pero los elementos sobre los que aplicar toda la transformación de Householder no están consecutivos. La biblioteca numérica utilizada requiere que todos ellos lo estén, por lo que habrá que realizar un movimiento de datos previo a la aplicación de la transformación, lo que a buen seguro conllevará cierta pérdida de eficiencia. Por último, podemos plantear una solución híbrida, empleando transformaciones de Householder para reflejar todos los elementos de una columna sobre el primero de ella, y luego rotar este último respecto al elemento diagonal.

Para cada posible versión podemos plantear múltiples combinaciones en cuanto a la forma de llevar a cabo el almacenamiento, bien por filas bien por columnas. La programación de los algoritmos ha sido realizada en Fortran, por lo que las matrices se almacenan por columnas, teniendo que interpretar cada matriz como tal o como su traspuesta en función del tipo de almacenamiento/representación escogido.

Emplearemos el convenio de *almacenamiento por filas*, cuando la matriz almacenada realmente sea la traspuesta, y *por columnas* en caso contrario.

La figura 6.1 muestra las cuatro posibilidades respecto al almacenamiento de la versión

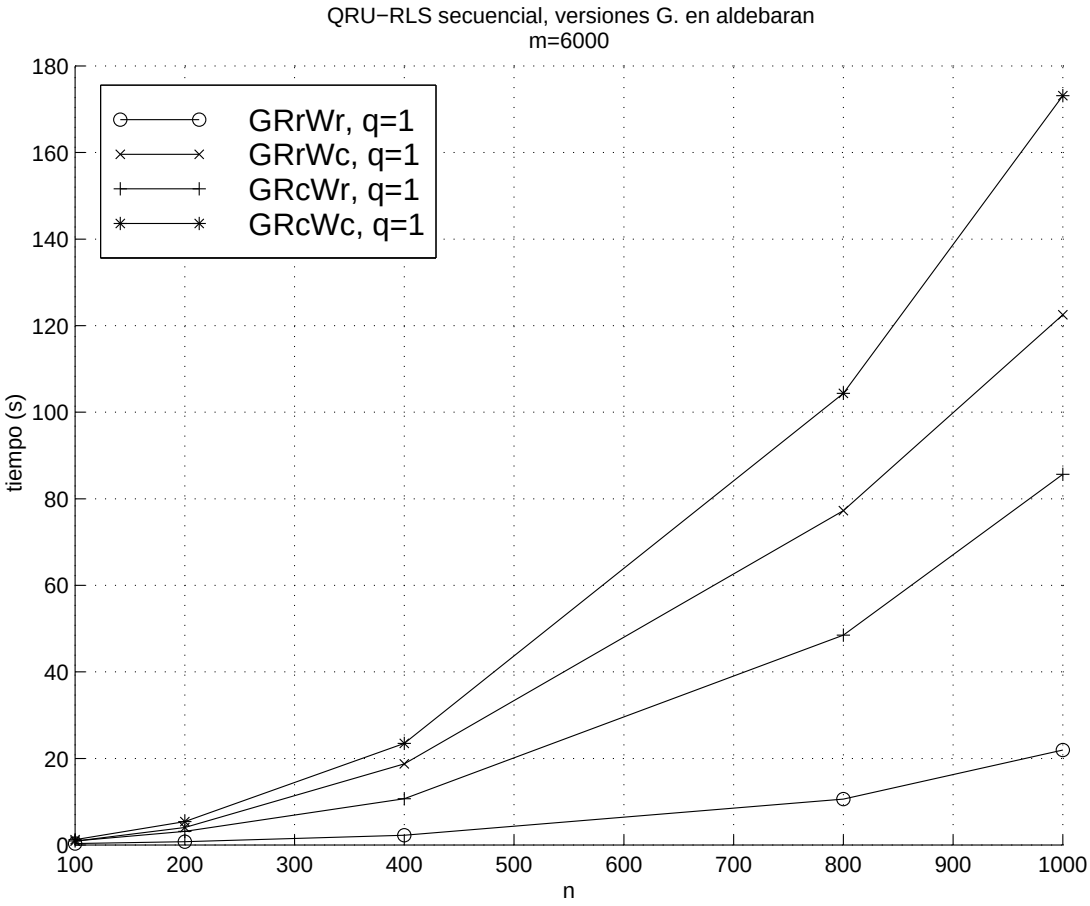

Figura 6.1: Tiempos de ejecución para la versión de Givens con distintos esquemas de almacenamiento, y $q = 1$.

basada exclusivamente en las rotaciones de Givens. La leyenda `GRxWx` denota lo siguiente: `G`, metodo de Givens, `Rx`, forma de almacenamiento de $(\mathbf{R}_{i,a}, \mathbf{Q}_{i,a}^T \mathbf{y}_i)$, donde `x` puede ser `r` (por filas) o `c` (por columnas); `Wx`, forma de almacenamiento de $(\mathbf{W}_i, \mathbf{y}_{q_i})$, con idénticas consideraciones.

Podemos observar claramente cómo el esquema de almacenamiento por filas de ambas submatrices es el que mejor tiempo de ejecución presenta, debido a que la aplicación de la rotación a un par de vectores es tal que los elementos consecutivos de cada vector son consecutivos en memoria, explotando el principio de localidad de los datos. El peor esquema de almacenamiento es el de ambas submatrices por columnas debido a que los elementos consecutivos de los vectores están separados por la dimensión principal de la matriz que los contiene. También podemos observar en la figura 6.2 que este tiempo de ejecución es

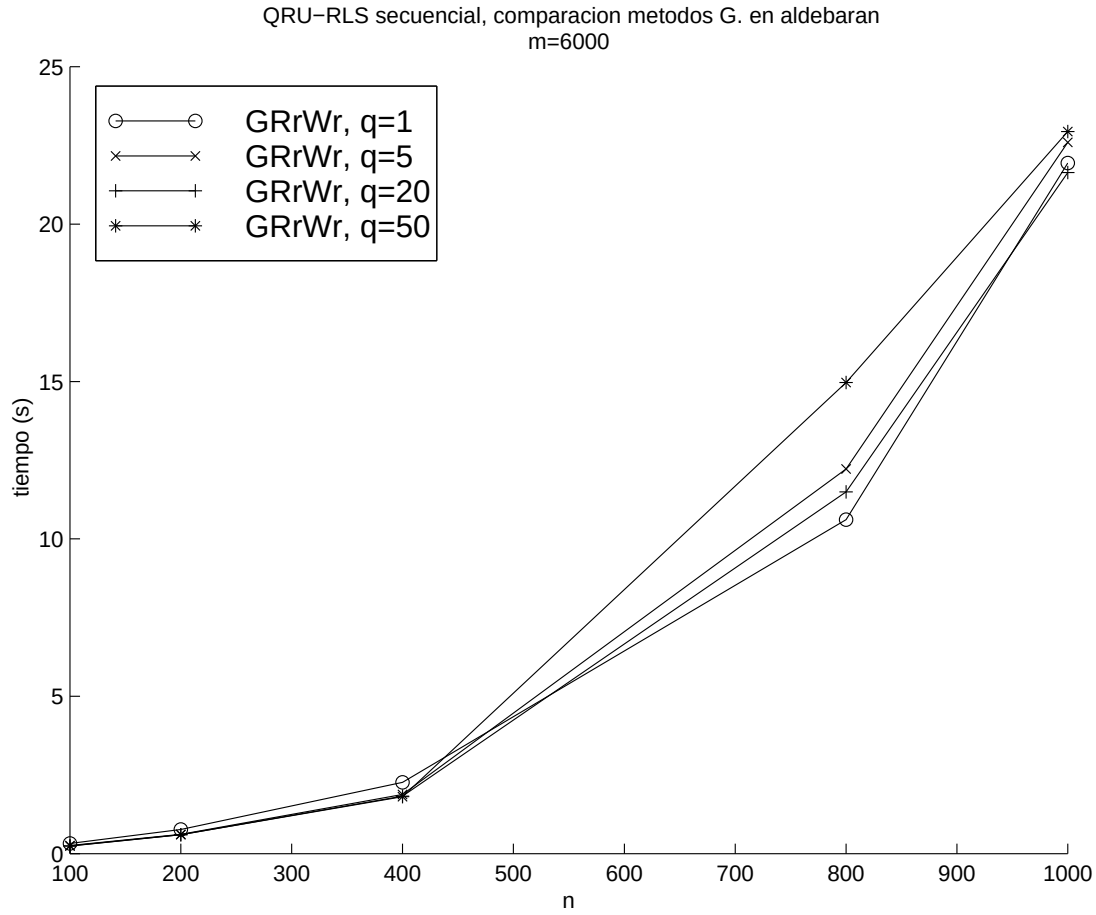

Figura 6.2: Independencia de q respecto al tiempo de ejecución de los métodos basados en Givens.

relativamente independiente de q, lo cual se tratará de justificar posteriormente.

La figura 6.3 muestra similares resultados, pero para la versión de Householder (`H`), donde se observa de nuevo que el esquema de almacenamiento por filas para ambas submatrices es el que menor tiempo de ejecución proporciona, siendo el de ambas por columnas, el peor de ellos, por los mismos motivos expuestos anteriormente.

Sin embargo, en la figura 6.4 se observa claramente la dependencia del tiempo de ejecución con el valor del parámetro q, que posteriormente trataremos de justificar.

Las figuras 6.5 y 6.6 muestran una comparación de los tiempos de ejecución de las mejores versiones de almacenamiento para el método de Givens y Householder, así como del híbrido (prefijo `X` en la leyenda), para $q = 1$ y $q = 20$ respectivamente.

Podemos apreciar cómo los métodos de Givens y el híbrido coinciden en los tiempos

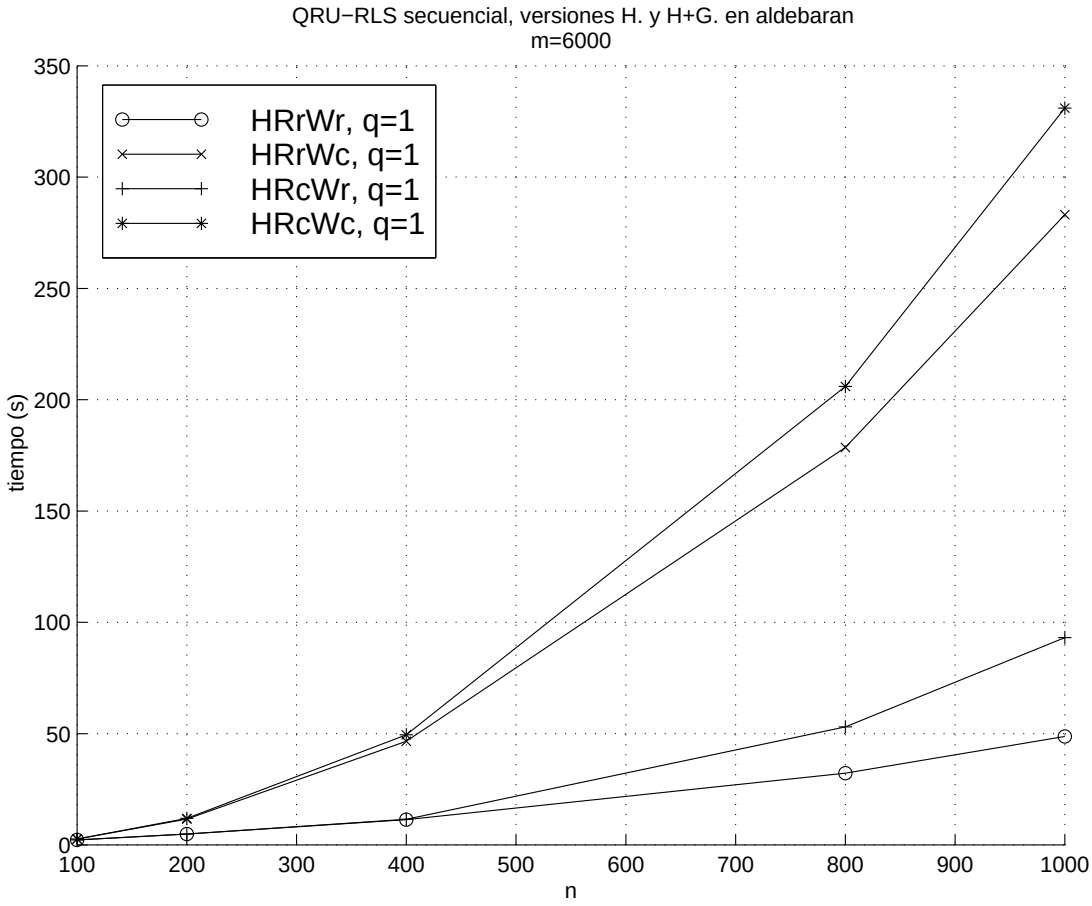

Figura 6.3: Tiempos de ejecución para la versión de Householder con distintos esquemas de almacenamiento, y $q = 1$.

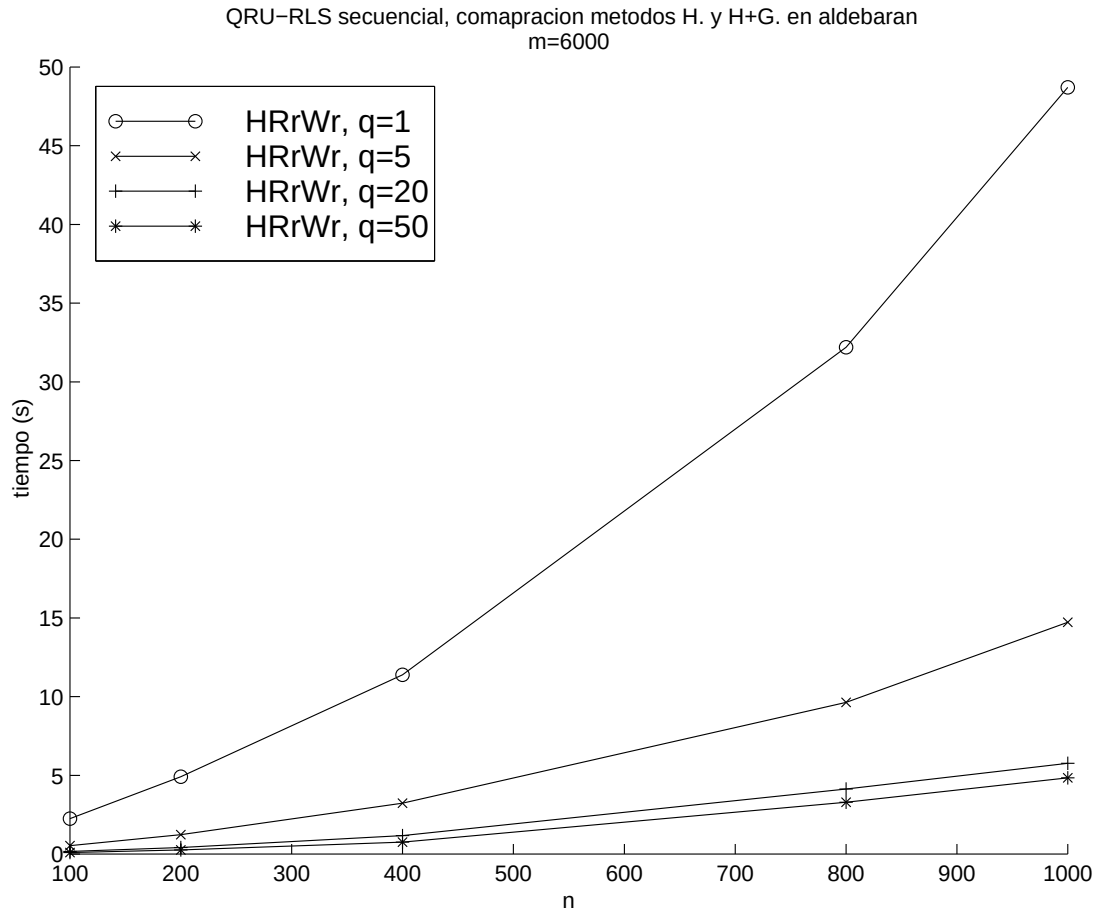

Figura 6.4: Dependencia de q respecto al tiempo de ejecución de los métodos basados en Householder.

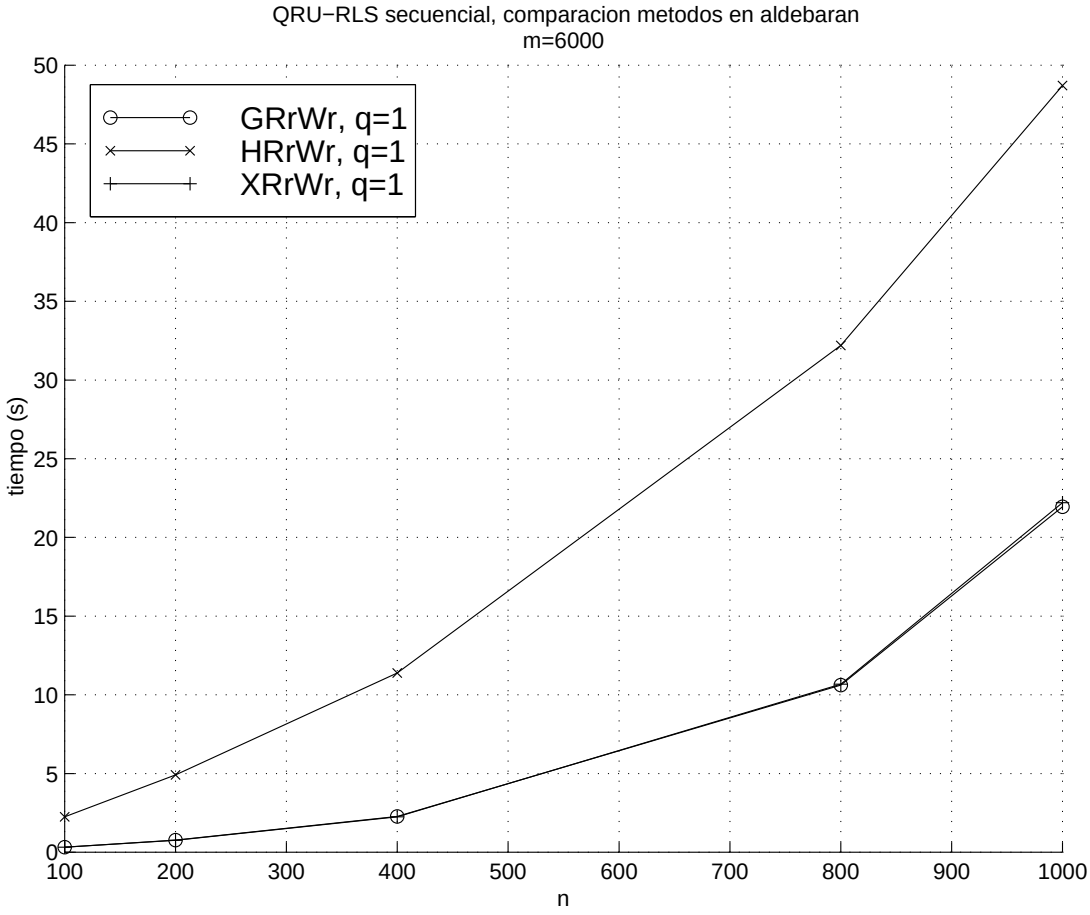

Figura 6.5: Comparación de los tres métodos para $q = 1$

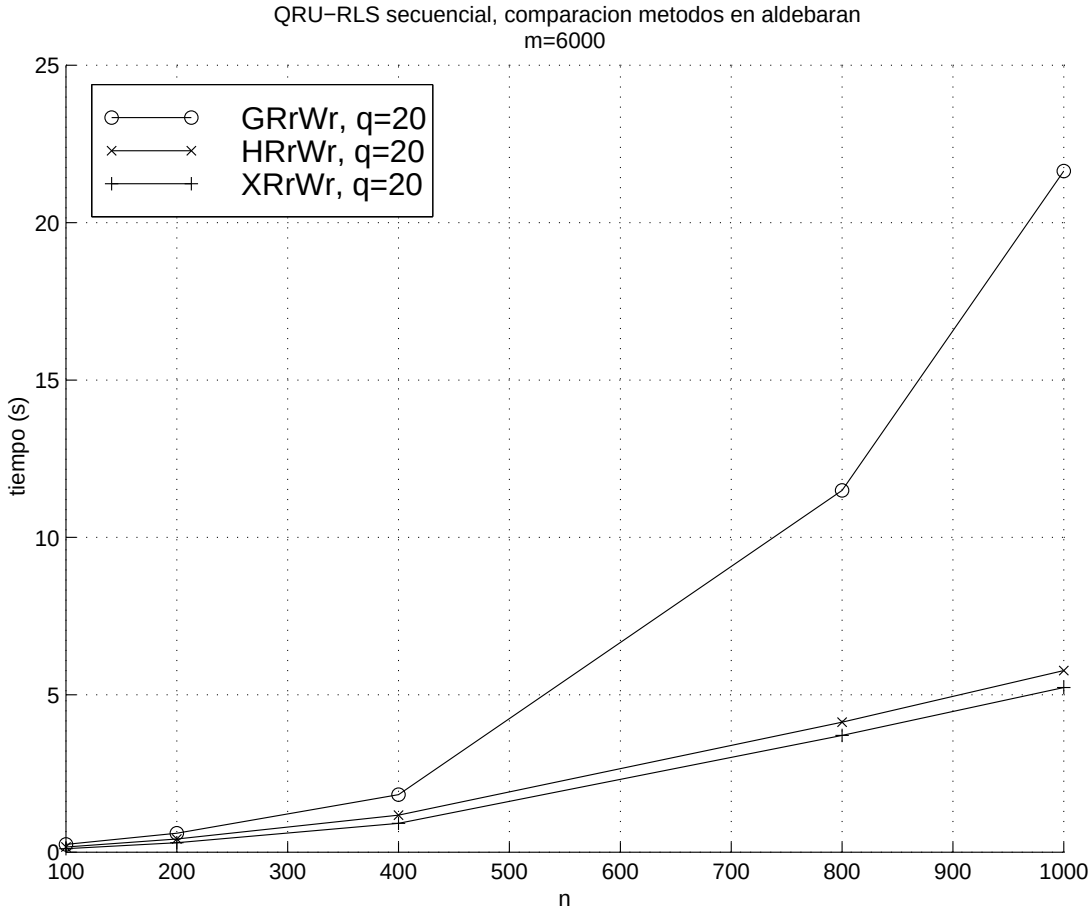

Figura 6.6: Comparación de los tres métodos para $q = 20$

de ejecución, ya que de hecho son el mismo para $q = 1$. Sin embargo, para $q > 1$, el método basado en Householder se comporta mejor que el de Givens y el híbrido mejor que el de Householder. Como posteriormente mostraremos, el método basado en Householder tiene un menor coste aritmético que el de Givens para $q > 1$; sin embargo, requiere el copiar parte de una fila de $(\mathbf{R}_{i,a}, \mathbf{Q}_{i,a}^T \mathbf{y}_i)$ a posiciones de memoria en las que se encuentre consecutivamente con $(\mathbf{W}_i, \mathbf{y}_{q_i})$ para poder aplicar la transformación de Householder a la totalidad y esto requiere cierta cantidad de tiempo. Este tiempo es contrarrestado en el método híbrido en el que se aplican las transformaciones de Householder sólo a $(\mathbf{W}_i, \mathbf{y}_{q_i})$ y por último, una rotación de Givens respecto de cierto elemento diagonal, evitando la copia.

Descartamos definitivamente la idea de hacer ceros por filas, ya que nos limita de manera exclusiva a emplear rotaciones de Givens y nos centramos en la anulación por columnas. A continuación se muestra un análisis temporal de la versión de Givens y de la híbrida. Por simplicidad en las expresiones, supondremos que $q_i = q$, $\forall\, i$, por lo que $m_i = iq$.

Independientemente del método a emplear sobre la matriz $\mathbf{W}_i$, debemos tener en cuenta que la aplicación de las transformaciones debe hacerse a las columnas que estén a la derecha de la columna tratada y además a las filas oportunas del vector columna $\left(\mathbf{y}_i^T \mathbf{Q}_{i,a}, \mathbf{y}_{q_i}^T\right)^T$, tal y como se muestra en la figura 6.7. En esta figura, también se muestra la perspectiva real que se tiene de las matrices debido al tipo de almacenamiento en Fortran, ya que la mejor ejecución se ha obtenido almacenando la matriz por filas, es decir, almacenando la transpuesta, atendiendo al esquema de almacenamiento de Fortran. Cuando toda la matriz $\mathbf{W}_i$ está anulada, tendremos que resolver un sistema triangular superior (triangular inferior desde la perspectiva Fortran).

Versión de Givens

- Para cada columna $c = 1, \ldots n$:

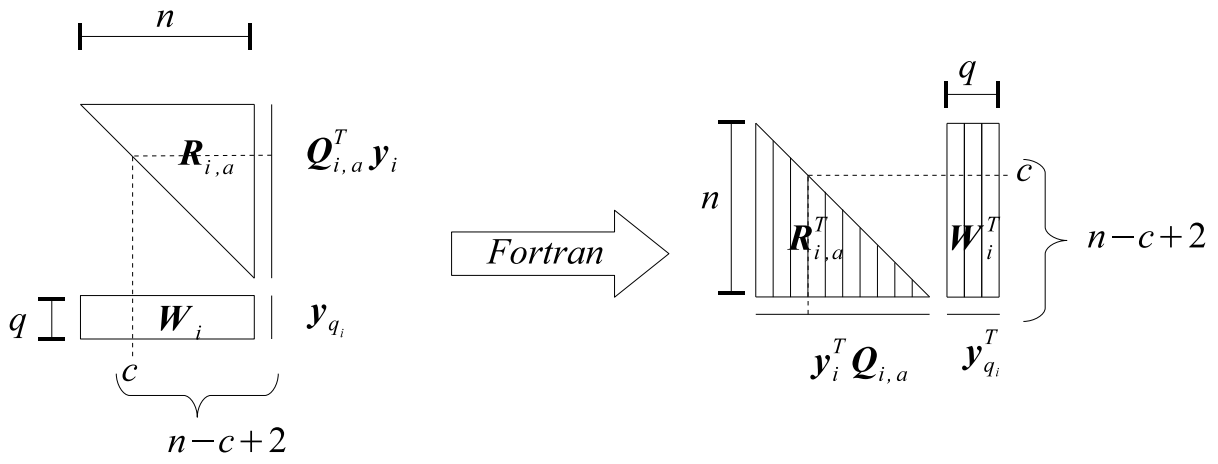

Figura 6.7: Rango de columnas a considerar en algoritmo QRU-RLS para la anulación de la columna c-ésima de $\mathbf{W}_i$, y almacenamiento en Fortran.

- Cálculo y aplicación de $q-1$ rotaciones de Givens a pares de filas de $n-c-2$ elementos, para rotar todos los elementos de la columna c sobre el primero de ella, luego el coste será $(q-1)(w_{CG}+w_{AG}(n-c+2))$ flops, donde w_{CG} es el coste de calcular una rotación de Givens y w_{AG} el de de aplicarla a un par de elementos.

- Cálculo y aplicación de una rotación de Givens a un par de filas de $n-c+2$ componentes: $w_{CG}+w_{AG}(n-c+2)$ flops, para rotar el primer elemento de la columna c respecto al elemento diagonal de $\mathbf{R}_{i,a}$ que se halla en su vertical.

Por lo que el coste de anular la columna c es $q(w_{CG}+w_{AG}(n-c+2))$ flops. El de anular las n columnas de $\mathbf{W}_i$ será:

$$\begin{aligned}\sum_{c=1}^{n}\{q(w_{CG}+w_{AG}(n-c+2))\} &= \\ q(w_{CG}+w_{AG}(n+2))n-qw_{AG}\sum_{c=1}^{n}c &= \\ q\left[w_{CG}+w_{AG}\left(\frac{1}{2}n+\frac{3}{2}\right)\right]n & \quad \text{flops}\end{aligned}$$

- Hay que resolver un sistema triangular superior de orden mín$\{m_i + q_i, n\}$, cuyo coste más desfavorable sería n^2 flops, [87].

De modo que el coste de una iteración para esta versión de Givens es:

$$q\left[w_{CG} + w_{AG}\left(\frac{1}{2}n + \frac{3}{2}\right)\right]n + n^2 \quad \text{flops}$$

Y el coste de las m/q iteraciones:

$$\sum_{i=0}^{m/q-1} q\left[w_{CG} + w_{AG}\left(\frac{1}{2}n + \frac{3}{2}\right)\right]n + n^2 \qquad = \tag{6.1}$$

$$\left[w_{CG} + w_{AG}\left(\frac{1}{2}n + \frac{3}{2}\right)\right]nm + n^2\frac{m}{q} \qquad \text{flops} \tag{6.2}$$

que podemos aproximar por

$$W_G(z) \approx \left(\frac{1}{2}w_{AG} + \frac{1}{q}\right)n^2 m = \Theta\left(mn^2\right) \quad \text{flops} \tag{6.3}$$

Versión híbrida Householder-Givens

- Para cada columna $c = 1, \ldots n$:
  - Cálculo y aplicación de una transformación de Householder de orden q a un total de $n - c + 2$ columnas : $w_{CH}(q) + w_{AH}(q)(n - c + 2)$ flops, donde $w_{CH}(q)$ y $w_{AH}(q)$ son los costes de calcular una transformación de Householder de orden q y aplicarla a una única columna, respectivamente (evitaremos la formación de la matriz de Householder para ahorrar un sustancial coste).
  - Cálculo y aplicación de una rotación de Givens a un par de filas de $n - c + 2$ componentes: $w_{CG} + w_{AG}(n - c + 2)$ flops.

  Por lo que el coste de anular una columna es $[w_{CH}(q) + w_{CG}] + [w_{AH}(q) + w_{AG}](n - c + 2)$ flops. El de anular las n columnas de $\mathbf{W}_i$ será:

$$\begin{aligned}
\sum_{c=1}^{n} \{[w_{CH}(q) + w_{CG}] + [w_{AH}(q) + w_{AG}](n - c + 2)\} &= \\
[w_{CH}(q) + w_{CG}]\, n &+ \\
[w_{AH}(q) + w_{AG}]\,(n+2)n - [w_{AH}(q) + w_{AG}] \sum_{c=1}^{n} c &= \\
[w_{CH}(q) + w_{CG}]\, n + [w_{AH}(q) + w_{AG}] \left(\frac{1}{2}n + \frac{3}{2}\right) n & \quad \text{flops}
\end{aligned} \tag{6.4}$$

- Hay que resolver un sistema triangular superior de orden mín$\{m_i + q_i, n\}$: n^2 flops.

De modo que genéricamente, el coste de una iteración para esta versión híbrida Householder-Givens es:

$$[w_{CH}(q) + w_{CG}]\, n + [w_{AH}(q) + w_{AG}] \left(\frac{1}{2}n + \frac{3}{2}\right) n + n^2 \quad \text{flops}$$

Y el coste de las m/q iteraciones:

$$\begin{aligned}
\sum_{i=0}^{m/q-1} [w_{CH}(q) + w_{CG}]\, n + [w_{AH}(q) + w_{AG}] \left(\frac{1}{2}n + \frac{3}{2}\right) n + n^2 &= \\
\left\{[w_{CH}(q) + w_{CG}]\, n + [w_{AH}(q) + w_{AG}] \left(\frac{1}{2}n + \frac{3}{2}\right) n + n^2\right\} \frac{m}{q} & \quad \text{flops}
\end{aligned}$$

que podemos aproximar por:

$$W_{HG}(z) \approx \left[\frac{1}{2q}\,(w_{AH}(q) + w_{AG}) + \frac{1}{q}\right] n^2 m \quad \text{flops} \tag{6.5}$$

Respecto a los valores de los cálculos y aplicaciones de las transformaciones de Householder y rotaciones de Givens, de la referencia [87] podemos concluir que (6.5) puede expresarse como $\Theta\,(mn^2)$.

Si empleamos los valores deducidos para el cálculo y aplicaciones de las transformaciones y rotaciones en (6.3) y 6.5) obtenemos unos costes aproximados de

$$\begin{aligned} W_G(z) &\approx \left(3+\frac{1}{q}\right)n^2m \quad \text{flops} \\ W_{HG}(z) &\approx \left(2+\frac{7}{2q}\right)n^2m \quad \text{flops} \end{aligned}$$

para la versión de Givens y la híbrida Householder-Givens respetivamente.

Cuando $q = 1$ las versiones son iguales, ya que no se llevan a cabo las aplicaciones de las transformaciones de Householder. Cuando q toma un valor grande, la versión híbrida es teóricamente 1,5 veces aproximadamente más rápida que la de Givens. Experimentalmente, estos factores han sido aproximadamente 1 y 4 respectivamente, según se deduce las gráficas de las figuras 6.5 y 6.6. Por la forma de los factores que acompañan a n^2m, la tendencia asintótica hacia 3 (con Givens) o hacia 2 (con Householder) es mucho más rápida en la versión de Givens que en la de Householder debido al factor $1/q$ frente a $7/(2q)$, lo cual hace que el método de Givens sea menos sensible al valor de q.

Adicionalmente, el algoritmo efectúa una cantidad de iteraciones igual a m/q, es decir, a mayor valor de q, menor número de iteraciones y viceversa. Con un menor número de iteraciones, y por tanto, con tamaños de los bloques, q, mayores, es razonable deducir que los algoritmos se comportarán mejor debido a que habrá un menor número de llamadas, un menor número de inicializaciones y la aplicación de las rotaciones de Givens y de las transformaciones de Householder serán hasta cierto punto más eficientes debido al efecto de la memoria *cache*. Todo ello justifica en cierta medida el comportamiento observado en las figuras 6.1–6.6. Debido a la obtención de mejores resultados en la versión híbrida, ésta será la que implementaremos en paralelo.

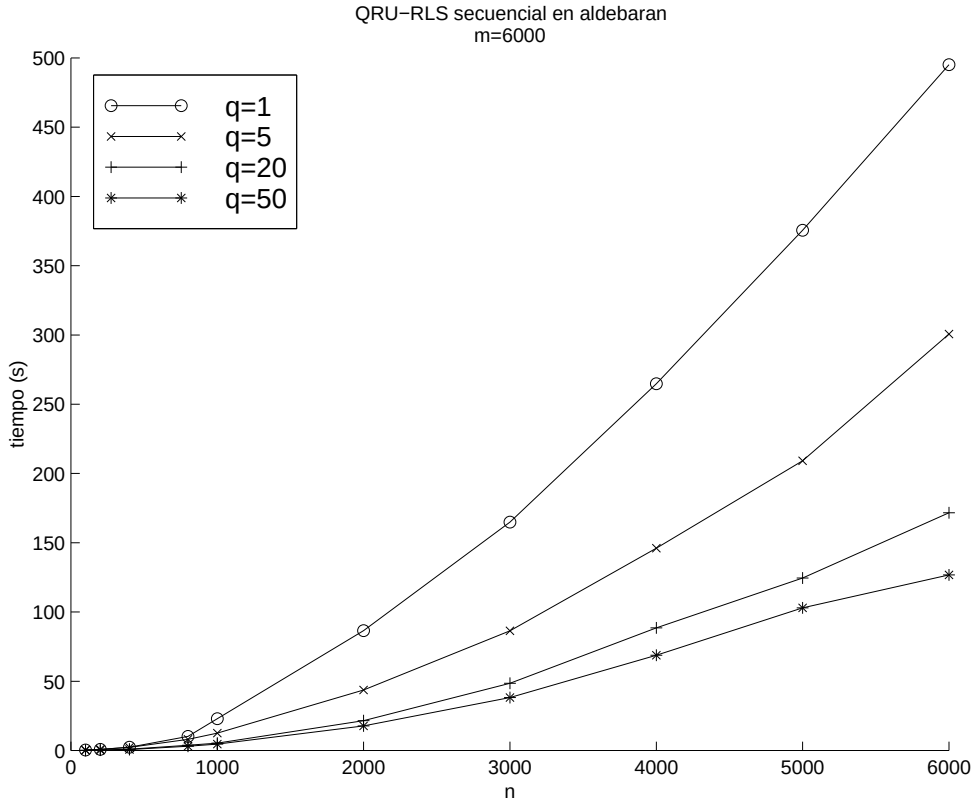

Figura 6.8: Tiempos de ejecución del algoritmo secuencial QRU-RLS, versión híbrida.

6.1.2. Resultados experimentales

Las siguientes pruebas se han llevado a cabo en la máquina ccNUMA *Aldebarán*. La figura 6.8 muestra los tiempos de ejecución de la versión híbrida del algoritmo. Podemos observar cómo los tiempos de ejecución tienden asintóticamente a cierto mínimo, conforme q va creciendo. La precisión de los resultados ha sido comparada con los resultados de la rutina `DGELSY` de `LAPACK` con una precisión del mismo orden de magnitud.

En la figura 6.9 se muestra el tiempo por iteración para distintos valores de n. Podemos observar cómo el tiempo por iteración va creciendo a medida que el índice de la iteración se va incrementando, debido a que la matriz $\mathbf{R}_{i,a}$ se va rellenando con q filas cada vez, y el trabajo es proporcional al tamaño que va poseyendo. Cuando la cantidad de filas $(i+1)q$ es igual a n la cantidad de tiempo se estabiliza, pues la matriz $\mathbf{R}_{i,a}$ ya está completa.

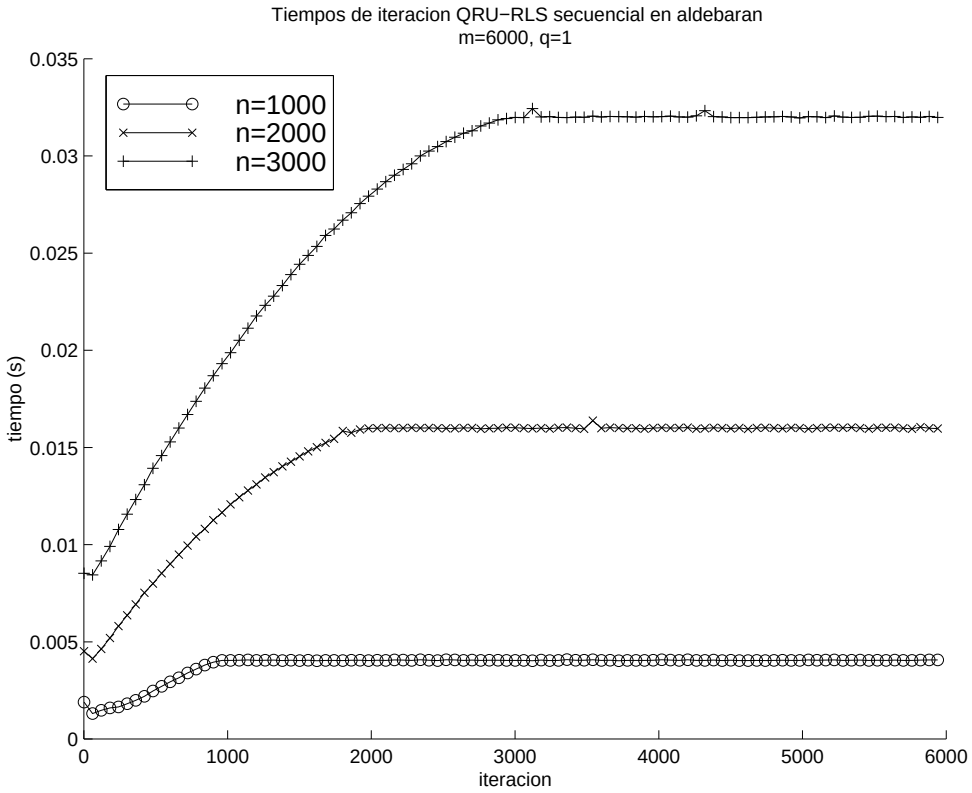

Figura 6.9: Tiempos de iteración del algoritmo secuencial QRU-RLS, versión híbrida.

6.1.3. Enlace con el filtro de información

Hemos apreciado que la actualización de la QR se lleva a cabo de manera recursiva, pero no la actualización de la solución RLS única. Veamos cómo podríamos llevarlo a cabo y sus consecuencias. Sea

$$\mathbf{H}_{i+1,a} = \begin{pmatrix} \mathbf{A}_{i+1} & \mathbf{B}_{i+1} \\ \mathbf{C}_{i+1} & \mathbf{D}_{i+1} \end{pmatrix} \tag{6.6}$$

la matriz ortogonal empleada en (3.16), donde $\mathbf{A}_{i+1} \in \mathbb{R}^{n\times n}$, $\mathbf{B}_{i+1} \in \mathbb{R}^{n\times q}$, $\mathbf{C}_{i+1} \in \mathbb{R}^{q\times n}$ y $\mathbf{D}_{i+1} \in \mathbb{R}^{q\times q}$; entonces, basándonos de nuevo en (3.16)

$$
\begin{aligned}
\begin{pmatrix} \mathbf{R}_{i,a} & \mathbf{Q}_{i,a}^T\mathbf{y}_i \\ \mathbf{W}_i & \mathbf{y}_{q_i} \end{pmatrix} &= \mathbf{H}_{i+1,a}^T \begin{pmatrix} \mathbf{R}_{i+1,a} & \mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1} \\ \mathbf{0} & \mathbf{z} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{A}_{i+1}^T & \mathbf{C}_{i+1}^T \\ \mathbf{B}_{i+1}^T & \mathbf{D}_{i+1}^T \end{pmatrix} \begin{pmatrix} \mathbf{R}_{i+1,a} & \mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1} \\ \mathbf{0} & \mathbf{z} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{A}_{i+1}^T\mathbf{R}_{i+1,a} & \mathbf{A}_{i+1}^T\mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1} + \mathbf{C}_{i+1}^T\mathbf{z} \\ \mathbf{B}_{i+1}^T\mathbf{R}_{i+1,a} & \mathbf{B}_{i+1}^T\mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1} + \mathbf{D}_{i+1}^T\mathbf{z} \end{pmatrix} \\
\mathbf{R}_{i,a} &= \mathbf{A}_{i+1}^T\mathbf{R}_{i+1,a} \\
\mathbf{R}_{i,a}^{-1} &= \mathbf{R}_{i+1,a}^{-1}\left(\mathbf{A}_{i+1}^T\right)^{-1} \\
\mathbf{Q}_{i,a}^T\mathbf{y}_i &= \mathbf{A}_{i+1}^T\mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1} + \mathbf{C}_{i+1}^T\mathbf{z}
\end{aligned}
$$

entonces

$$
\begin{aligned}
\mathbf{x}_{\mathrm{LS},i} &= \mathbf{R}_{i,a}^{-1}\mathbf{Q}_{i,a}^T\mathbf{y}_i \\
&= \mathbf{R}_{i+1,a}^{-1}\left(\mathbf{A}_{i+1}^T\right)^{-1}\left(\mathbf{A}_{i+1}^T\mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1} + \mathbf{C}_{i+1}^T\mathbf{z}\right) \\
&= \mathbf{R}_{i+1,a}^{-1}\mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1} + \mathbf{R}_{i+1,a}^{-1}\left(\mathbf{A}_{i+1}^T\right)^{-1}\mathbf{C}_{i+1}^T\mathbf{z} \\
&= \mathbf{x}_{\mathrm{LS},i+1} + \mathbf{R}_{i+1,a}^{-1}\left(\mathbf{A}_{i+1}^T\right)^{-1}\mathbf{C}_{i+1}^T\mathbf{z} \\
\mathbf{x}_{\mathrm{LS},i+1} &= \mathbf{x}_{\mathrm{LS},i} - \mathbf{R}_{i+1,a}^{-1}\left(\mathbf{A}_{i+1}^T\right)^{-1}\mathbf{C}_{i+1}^T\mathbf{z} \\
&= \mathbf{x}_{\mathrm{LS},i} - \mathbf{R}_{i,a}^{-1}\mathbf{C}_{i+1}^T\mathbf{z}
\end{aligned}
$$

Por lo que ha hemos obtenido una expresión para actualizar recursivamente la solución RLS. Si calculamos

$$\mathbf{H}_{i+1,a}\begin{pmatrix}\left(\mathbf{R}_{i,a}^{-1}\right)^T\\ \mathbf{0}\end{pmatrix} = \begin{pmatrix}\mathbf{A}_{i+1} & \mathbf{B}_{i+1}\\ \mathbf{C}_{i+1} & \mathbf{D}_{i+1}\end{pmatrix}\begin{pmatrix}\left(\mathbf{R}_{i,a}^{-1}\right)^T\\ \mathbf{0}\end{pmatrix}$$
$$= \begin{pmatrix}\mathbf{A}_{i+1}\left(\mathbf{R}_{i,a}^{-1}\right)^T\\ \mathbf{C}_{i+1}\left(\mathbf{R}_{i,a}^{-1}\right)^T\end{pmatrix}$$
$$= \begin{pmatrix}\left(\mathbf{R}_{i+1,a}^{-1}\right)^T\\ \mathbf{C}_{i+1}\left(\mathbf{R}_{i,a}^{-1}\right)^T\end{pmatrix}$$

podemos obtener de forma iterativa todos los datos necesarios para actualizar $\mathbf{x}_{\mathrm{LS},i}$. Si ampliamos (3.16), podemos conseguir toda la información de manera conjunta:

$$\mathbf{H}_{i+1,a}\left(\begin{array}{c|c|c}\mathbf{R}_{i,a} & \mathbf{Q}_{i,a}^T\mathbf{y}_i & \left(\mathbf{R}_{i,a}^{-1}\right)^T\\ \mathbf{W}_i & \mathbf{y}_{q_i} & \mathbf{0}\end{array}\right) = \left(\begin{array}{c|c|c}\mathbf{R}_{i+1,a} & \mathbf{Q}_{i+1,a}^T\mathbf{y}_{i+1} & \left(\mathbf{R}_{i+1,a}^{-1}\right)^T\\ \mathbf{0} & \mathbf{z} & \mathbf{C}_{i+1}\left(\mathbf{R}_{i,a}^{-1}\right)^T\end{array}\right) \tag{6.7}$$

con

$$\mathbf{x}_{\mathrm{LS},i+1} = \mathbf{x}_{\mathrm{LS},i} - \mathbf{R}_{i,a}^{-1}\mathbf{C}_{i+1}^T\mathbf{z}$$

llegando al mismo esquema algorítmico que el *filtro de información*, aunque con una versión traspuesta (compárense las estructuras matriciales con (5.1) y (5.2). Llegamos, por tanto, a las mismas conclusiones que en [13] respecto a la relación íntima existente entre el problema RLS y la actualización recursiva de la estimación del estado de un sistema mediante el filtro de Kalman.

6.2 Algoritmo paralelo

A continuación procederemos a realizar el diseño del algoritmo paralelo, concretamente la versión híbrida Householder-Givens, siguiendo las pautas de los algoritmos paralelos

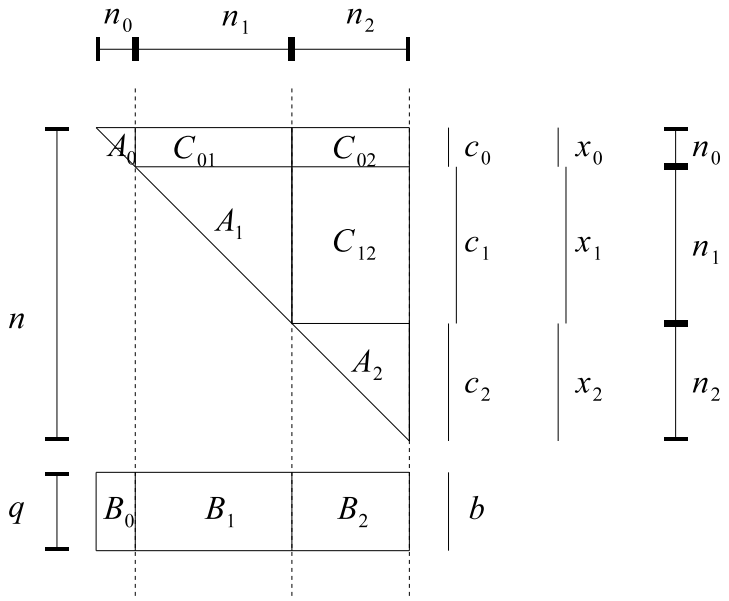

Figura 6.10: Particiones para el algoritmo paralelo QRU-RLS

descritos en los capítulos anteriores.

6.2.1. Descomposición de los datos, tareas y dependencias

En la figura 6.10 se muestra un ejemplo de la descomposición de datos de la matriz problema que podemos establecer en el algoritmo. Podemos observar que la matriz $\mathbf{R}_{i,a}$ se ha dividido en submatrices de tipo A y tipo C, el vector $\mathbf{Q}_{i,a}^T\mathbf{y}_i$, en subvectores de tipo c, $\mathbf{W}_i$ en submatrices de tipo B, el vector $\mathbf{y}_{q_i}$, en un un único vector de tipo b y por último, la solución en subvectores de tipo x.

Las tareas que requiere cada uno de estos datos son las siguientes:

- Submatrices B, de dimensiones $q \times n_j$: requieren el hacer ceros en sus n_j columnas. Ello implica que:

  - Cualquier columna a eliminar de cualquier submatriz B_j requiere el elemento diagonal correspondiente de la submatriz A_j que tiene en su vertical, por lo que surge una dependencia de A_j.

  - Cualquier submatriz B_j requiere información de las transformaciones aplicadas a las submatrices B_k, con $k < j$, para aplicarlas a sí misma.
  - Cualquier submatriz B_j requiere información de las transformaciones aplicadas a las submatrices $A_k - B_k$, con $k < j$ para aplicarlas junto con las submatrices C_{kj}, por lo que surge una nueva dependencia de C_{kj}

- Submatrices A: de dimensiones $n_j \times n_j$ requiere información sobre la aplicación de las transformaciones calculadas, por lo que surge una co-dependencia $A_j - B_j$
- Submatrices C, denotadas como C_{kj} con dimensiones $n_k \times n_j$: requiere la información de las transformaciones aplicadas en la co-dependencia $A_k - B_k$, mostrando una co-dependencia con la submatriz B_j.
- Subvectores c, de dimensiones $n_j \times 1$: tienen consideraciones análogas a las submatrices C, mostrando co-dependencia con la partición b.
- Subvectores x, de dimensiones $n_j \times 1$: el subvector x_j tiene dependencia de las submatrices A_j, $C_{j,j+1}, \ldots, C_{j,t-1}$, donde t es la cantidad de submatrices c, ó A ó B que existen, junto con subvectores x_k con $k > j$, para calcular las solución RLS por sustitución regresiva por bloques.

Estas dependencias y co-dependencias son mostradas en la figura 6.11. Parece razonable el plantear la siguiente estrategia en el algoritmo paralelo:

- La cantidad t de submatrices de cada clase (exceptuando el vector b) coincidirá con la cantidad de segmentos que se establezcan.
- La información que requiere un segmento j proviene exclusivamente del segmento anterior, y son las submatrices $B_{j+1}, \ldots, B_{t-1}$ y el vector b.
- Tras el último segmento, ya se puede hallar la solución RLS. Cada particion x_j requiere los subvectores x_k con $k > j$, las submatrices A_j y $C_{j,j+1}, \ldots, C_{j,t-1}$.

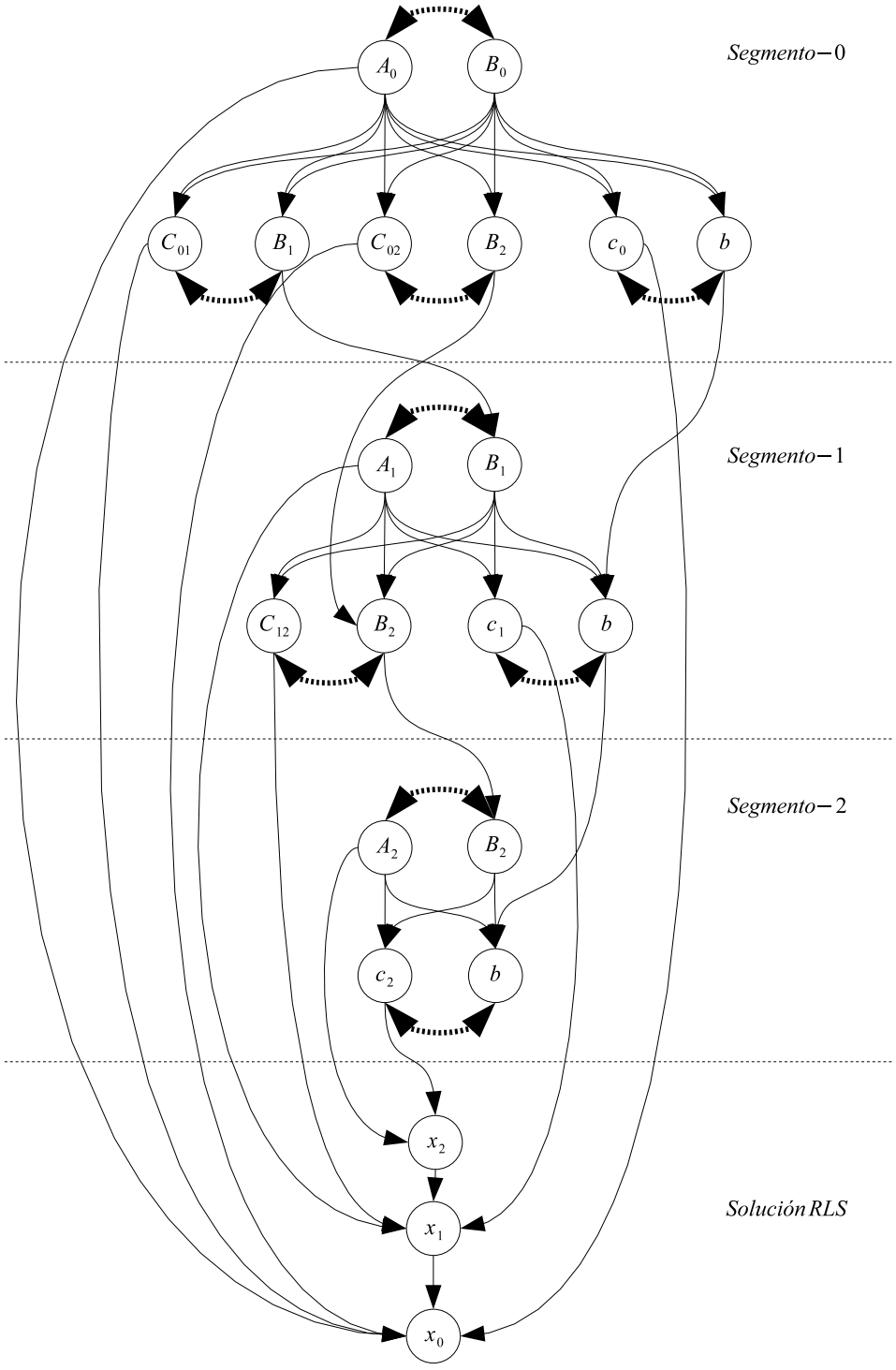

Figura 6.11: Dependencias y co-dependencias de las particiones del algoritmo QRU-RLS.

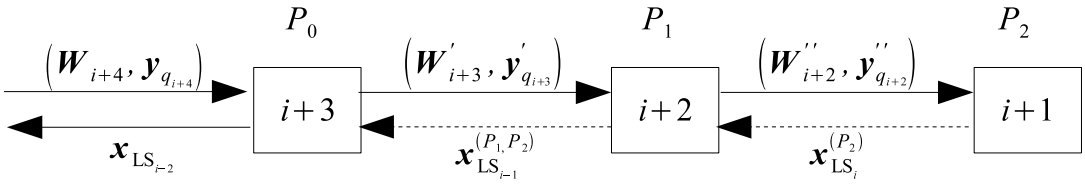

Figura 6.12: Desfase en el índice de la iteración entre el cálculo en la línea segmentada y la obtención de la solución

- Dado el grado de concurrencia que se desprende de las dependencias, para todas las iteraciones, asignaremos el mismo segmento/conjunto de particiones a un procesador: el procesador P_j, $0 \leq j \leq p-1$, poseerá siempre las submatrices A_j, $C_{j,j+1}, \ldots, C_{j,t-1}$, c_j y x_j.

- Las submatrices B_j, $j = 0, \ldots, t-1$ todavía no nulas, junto con el vector b, irán de procesador en procesador, conforme vaya avanzando la ejecución de los segmentos de la iteración.

6.2.2. Detalles de implementación

Para que un procesador pueda calcular su parte asignada de la solución RLS requiere de las calculadas por los procesadores de índice superior. Por ello, el procesador P_0 será el que posea la solución total y final de cualquier iteración i-ésima. Esta recolección de partes de la solución puede hacerse también de manera segmentada para que el algoritmo no pierda el carácter segmentado. Aparece, entonces, un pequeño inconveniente en el momento de calcular las solución RLS de la iteración i-ésima, ya que ésta debe ser realizada en orden inverso, es decir, las primeras componentes que se pueden calcular de la solución son las que ocupan las últimas posiciones (sustitución regresiva). Cuando P_{p-1} calcula dichas componentes, las envía a P_{p-2} y así sucesivamente. Cuando esto ocurre, el procesador P_{p-2} está haciendo cálculos en una iteración superior, por lo que deberá haber guardado datos de la iteración anterior y así sucesivamente. Esta situación se muestra en la figura 6.12.

Este problema de desfase puede solucionarse guardando la información necesaria en una cola circular con una estructura de datos acorde con la información que necesita ser

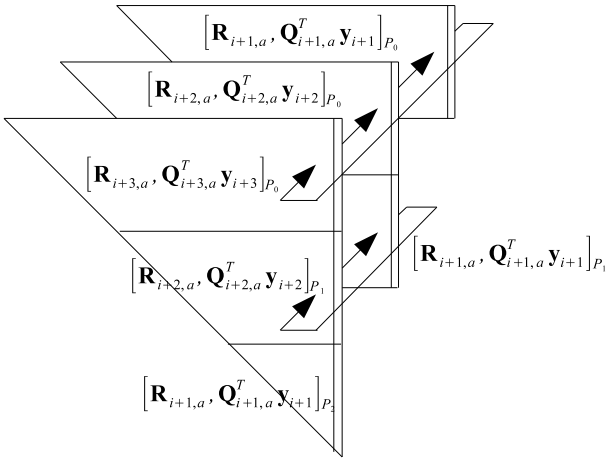

Figura 6.13: Cola circular en el algoritmo QRU-RLS

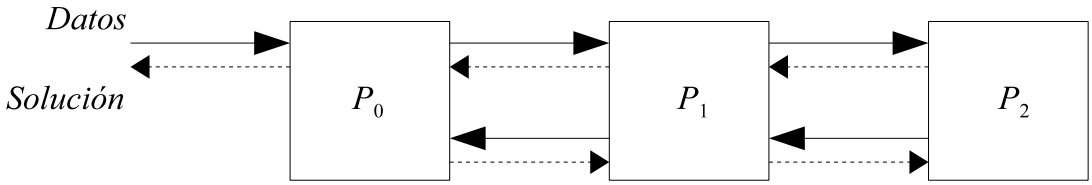

Figura 6.14: Línea segementada bidireccional para los dos flujos de información.

almacenada, tal y como muestra la figura 6.13

De manera intrínseca, queriendo respetar el carácter segmentado del algoritmo, la línea segmentada necesariamente es bidireccional: en un sentido se transmite la información necesaria para ir actualizando la submatriz $(\mathbf{R}_{i,a}, \mathbf{Q}^T_{i,a}\mathbf{y}_i)$, a $(\mathbf{R}_{i+1,a}, \mathbf{Q}^T_{i+1,a}\mathbf{y}_{i+1})$ y en sentido opuesto las componentes de $\mathbf{x}_{\mathrm{LS}_i}$ que van calculando los procesadores. Para evitar confusión en la notación, diremos que esta situación se corresponde realmente con una línea segmentada *unidireccional doble*. Aunque posteriormente se detallará en el apartado de equilibrado de carga, podemos plantear cada línea unidireccional en ambos sentidos, dando lugar a una línea segmentada *bidireccional doble*, la cual se muestra en la figura 6.14, donde con líneas continuas se muestra la transferencia de información para actualizar $(\mathbf{R}_{i,a}, \mathbf{Q}^T_{i,a}\mathbf{y}_i)$ a $(\mathbf{R}_{i+1,a}, \mathbf{Q}^T_{i+1,a}\mathbf{y}_{i+1})$ y con discontinuas la de la solución RLS. Todo esto implica, de alguna forma, que un nodo intermedio en la línea segmentada debe simultanear ocho flujos de información.

La figura 6.15 pretende ilustrar gráficamente el comportamiento del algoritmo en el

instante de tiempo en el que se está generando la solución $\mathbf{x}_{\mathrm{LS}}$ para el instante i-ésimo

6.2.3. Costes de los segmentos

Por claridad en las expresiones, supondremos que $m_i + q_i \geq n$. El coste asociado al segmento j-ésimo es el siguiente:

- Debe anular cada una de las columnas $c = 1, \ldots n_j$ de B_j, es decir, las columnas $c_{0_j}, \ldots, c_{0_j} + n_j - 1$ de $\mathbf{W}_i$ (véase la figura 6.7). Las transformaciones hay que aplicarlas a un total de $(n - c_{0_j} - c + 3)$ columnas. Luego reescribiendo (6.4):

$$\begin{aligned}
& \sum_{c=1}^{n_j} \left\{ [w_{CH}(q) + w_{CG}] + [w_{AH}(q) + w_{AG}] \, (n - c_{0_j} - c + 3) \right\} \\
= \; & [w_{CH}(q) + w_{CG}] \, n_j + [w_{AH}(q) + w_{AG}] \, (n - c_{0_j} + 3) n_j \\
- \; & [w_{AH}(q) + w_{AG}] \sum_{c=1}^{n_j} c \\
= \; & [w_{CH}(q) + w_{CG}] \, n_j + [w_{AH}(q) + w_{AG}] \left(n - c_{0_j} + \frac{5}{2} - \frac{1}{2} n_j \right) n_j
\end{aligned}$$

- Cuando un procesador reciba las componentes de la solución $\mathbf{x}_{\mathrm{LS}_i}$ de los procesadores de índice superior, podrá calcular las correspondientes asociadas a su rango de filas, siguiendo un procedimiento de sustitución regresiva por bloques. Si denotamos por $\mathbf{x}_{\mathrm{LS}_i}^{(P_j)}$ las componentes de $\mathbf{x}_{\mathrm{LS}_i}$ asociadas al procesador P_j ubicadas en la particion x_j, entonces la solución parcial que éste puede calcular puede describirse como:

$$\mathbf{x}_{\mathrm{LS}_i}^{(P_j)} = A_j^{-1} \left(c_j - \sum_{k=p-1}^{j-1} C_{jk} \mathbf{x}_{\mathrm{LS}_i}^{(P_k)} \right)$$

donde A_j, c_j y C_{jk} son las particiones asignadas al procesador P_j y $\mathbf{x}_{\mathrm{LS}_i}^{(P_k)}$ es la particion x_k de x asignada al procesador P_k, con $k > j$. El coste asociado al sumatorio

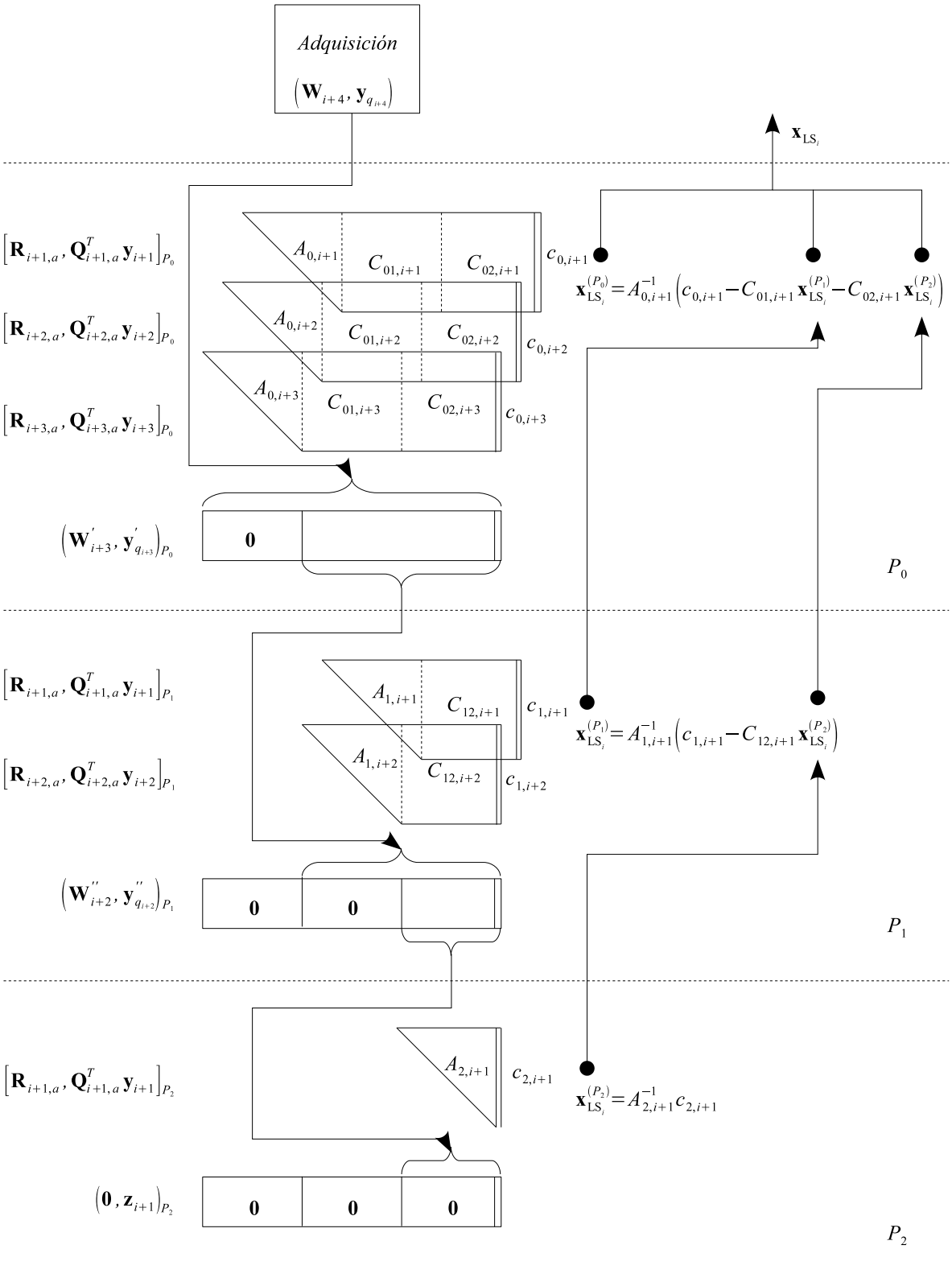

Figura 6.15: Estado del algoritmo al entregar $\mathbf{x}_{\mathrm{LS}_i}$.

de la expresión podemos calcularlo como el de una multiplicación matriz-vector de dimensiones $n_j \times (n - c_{0_j} - n_j + 1)$, cuyo coste podemos expresarlo como $n_j(2n - 2c_{0_j} - 2n_j + 1)$ flops; deberemos añadir n_j flops de la resta por lo que el coste parcial será:

$$\left[2n - 2c_{0_j} - 2n_j + 2\right] n_j \quad \text{flops}$$

Por último habrá que añadir el cálculo de la multiplicación por A_j^{-1}, que puede concebirse como el cálculo de la solución a un sistema triangular superior, en vez de tener que calcular la inversa, con un coste de n_j^2 flops, [87], por lo que el coste total será

$$\left[2n - 2c_{0_j} + 2 - n_j\right] n_j \quad \text{flops}$$

Por lo tanto el coste aritmético paralelo de una iteración en el procesador P_j, obviando las situaciones de llenado o vaciado de la línea segmentada, y suponiendo que ya se puede calcular la solución RLS, es:

$$\begin{aligned} W_{P_j,i}(z) &= \left[w_{CH}(q) + w_{CG}\right] n_j + \left[w_{AH}(q) + w_{AG}\right] \left(n - c_{0_j} + \frac{5}{2} - \frac{1}{2} n_j\right) n_j \\ &\quad + \left[2n - 2c_{0_j} + 2 - n_j\right] n_j \quad \text{flops} \end{aligned}$$

Podemos comprobar que

$$W_{\text{sec},i}(z) = \sum_{j=0}^{t-1} W_{P_j,i}(z)$$

Y por tanto que

$$W_{\text{sec}}(z) = \sum_{i=0}^{m/q-1} \sum_{j=0}^{t-1} W_{P_j,i}(z)$$

por lo que la sobrecarga aritmética por paralelización podemos considerarla idealmente nula.

6.2.4. Pseudocódigo

El **Algoritmo** 6 muestra el pseudocódigo para una línea segmentada unidireccional doble. En la descripción c_{0_j} denota el índice de la primera fila de $(\mathbf{R}_{i,a}, \mathbf{Q}_{i,a}^T \mathbf{y}_i)$ asignada el procesador P_j, y n_j la cantidad total de filas. Respecto a la gestión de la cola circular, el operador $|\cdot|$ pretende denotar la operación módulo tamaño de la cola circular.

6.2.5. Equilibrado de la carga

Línea segmentada unidireccional doble

Para conseguir un correcto equilibrado, debemos reescribir (2.8) a nivel de iteración, por lo que

$$\begin{aligned} [w_{CH}(q) + w_{CG}]\, n_j + [w_{AH}(q) + w_{AG}] \left(n - c_{0_j} + \frac{5}{2} - \frac{1}{2} n_j \right) n_j \quad &+ \\ \left[2n - 2c_{0_j} + 2 - n_j \right] n_j \quad &= \\ \alpha_j \left\{ [w_{CH}(q) + w_{CG}]\, n + [w_{AH}(q) + w_{AG}] \left(\frac{1}{2} n + \frac{3}{2} \right) n + n^2 \right\} & \end{aligned}$$

Podemos comenzar con n_0, con $c_{0_0} = 1$, resolviendo una ecuación de segundo grado. Para el resto de valores n_j, simplemente debemos tener en cuenta que

$$c_{0_j} = 1 + \sum_{k=0}^{j-1} n_k = n + 1 - \sum_{k=j}^{t-1} n_k$$

donde t es el número de particiones que en este caso coincidirá con el de procesadores.

Podemos emplear los valores de la carga asociada a los cálculos y aplicaciones de las transformaciones de Householder y rotaciones de Givens mostrados en la subsección 6.1.1.

Algoritmo 6 RLS paralelo: actualización de factorización QR

Entrada: $\mathbf{y} = (\mathbf{y}_0^*, \mathbf{y}_1^*, \ldots)^*$, $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$, en P_0
Salida: $\mathbf{x}_{\mathrm{LS},r}$ cada iteración r-ésima, en P_0

1: **para todo** P_j : en P_j **hacer**
2: $(\mathbf{R}_{i,a}, \mathbf{Q}_{i,a}^T \mathbf{y}_i)_{P_j} \leftarrow \mathbf{0}$
3: $r \leftarrow 0$
4: **para** $i = 0, \ldots$ **hacer**
5: **si** $P_j = P_0$ **entonces**
6: $(\mathbf{W}_i, \mathbf{y}_{q_i})_{P_0} \Leftarrow (\mathbf{W}_i, \mathbf{y}_{q_i})_{\text{adquisición}}$
7: **sino**
8: $(\mathbf{W}_i, \mathbf{y}_{q_i})_{P_j} \Leftarrow (\mathbf{W}'_i, \mathbf{y}'_{q_i})_{P_{j-1}}$
9: **fin si**
10: **para** $k = 1 : \min\{(i+1)q - c_{0_j} + 1, n_j\}$ **hacer**
11: Anular columna $k + c_{0_j} - 1$ de $\mathbf{W}_i$

$$\left(\begin{bmatrix} \mathbf{R}_{i,a} & \mathbf{Q}_{i,a}^T \mathbf{y}_i \\ \mathbf{W}_i & \mathbf{y}_{q_i} \end{bmatrix}_{P_j} \right) \leftarrow \mathbf{\Theta}_{k,P_j} \left(\begin{bmatrix} \mathbf{R}_{i,a} & \mathbf{Q}_{i,a}^T \mathbf{y}_i \\ \mathbf{W}_i & \mathbf{y}_{q_i} \end{bmatrix}_{P_j} \right)$$

12: **fin para**
13:

$$\left(\begin{bmatrix} \mathbf{R}_{i+1,a} & \mathbf{Q}_{i+1,a}^T \mathbf{y}_{i+1} \\ \mathbf{W}'_i & \mathbf{y}'_{q_i} \end{bmatrix}_{P_j} \right) \leftarrow \left(\begin{bmatrix} \mathbf{R}_{i,a} & \mathbf{Q}_{i,a}^T \mathbf{y}_i \\ \mathbf{W}_i & \mathbf{y}_{q_i} \end{bmatrix}_{P_j} \right)$$

14: **si** $P_j \neq P_{p-1}$ **entonces**
15: $(\mathbf{W}'_i, \mathbf{y}'_{q_i})_{P_j} \Rightarrow (\mathbf{W}_i, \mathbf{y}_{q_i})_{P_{j+1}}$
16: Guardar $[\mathbf{R}_{i+1,a}, \mathbf{Q}_{i+1,a}^T \mathbf{y}_{i+1}]_{P_j}$ —particiones A_j, $C_{j,j+1}, \ldots C_{j,p-1}$ y c_j— en posición $|i|$ de la cola circular
17: **fin si**
18: **si** ($\mathbf{x}_{\mathrm{LS}_r}^{(P_{j+1} \cdots P_{p-1})}$ recibido de P_{j+1}) **entonces**
19: Extraer particiones A_j, $C_{j,j+1}, \ldots C_{j,p-1}$ y c_j de posición r de la cola circular
20: $\mathbf{x}_{\mathrm{LS}_r}^{(P_j)} = A_j^{-1} \left(c_j - \sum_{k=p-1}^{j-1} C_{jk} \mathbf{x}_{\mathrm{LS}_r}^{(P_k)} \right)$
21: **si** $P_j \neq P_0$ **entonces**
22: Enviar $\mathbf{x}_{\mathrm{LS}_r}^{(P_j \cdots P_{p-1})}$ a P_{j-1}
23: **sino**
24: $\mathbf{x}_{\mathrm{LS}_r} = \mathbf{x}_{\mathrm{LS}_r}^{(P_0 \cdots P_{p-1})}$
25: **fin si**
26: $r \leftarrow |r+1|$
27: **fin si**
28: **fin para**
29: **fin para**

Podemos prever cierto desequilibrio residual debido a que nos hemos basado en un modelo teórico difícil de hacer coincidir exactamente con los resultados experimentales. Adicionalmente, el equilibrado ha sido tenido en cuenta para cuando hay más carga, es decir, para cuando ya se puede calcular la solución RLS. Si n tiene un valor cercano a m, sólo en las últimas iteraciones se empezará a calcular dicha solución, por lo que el sistema estaría desequilibrado durante una cantidad importante de iteraciones.

Línea segmentada bidireccional doble

En la figura 6.14 se mostraba el esquema que podemos emplear para asignar la carga de manera especular. Podemos comprobar que podemos aplicar este esquema de equilibrado ya que la carga de cualquier partición cumple la función (4.16).

$$\begin{aligned}
[w_{CH}(q)+w_{CG}]\,n_j+[w_{AH}(q)+w_{AG}]\left(n-c_{0_j}+\frac{5}{2}-\frac{1}{2}n_j\right)n_j \quad &+\\
\left[2n-2c_{0_j}+2-n_j\right]n_j \quad &=\\
\Big[(w_{AH}(q)+w_{AG}+2)n-(w_{AH}(q)+w_{AG}+2)c_{0_j}+&\\
\frac{5}{2}(w_{AH}(q)+w_{AG})+1+(w_{CH}(q)+w_{CG})\Big]\,n_j&\\
-\frac{1}{2}\left[w_{AH}(q)+w_{AG}+2\right]n_j^2&
\end{aligned}$$

siendo

- $\beta=\rho=-\left[w_{AH}(q)+w_{AG}+2\right]$
- $f(q,n)=(w_{AH}(q)+w_{AG}+2)n+\frac{5}{2}(w_{AH}(q)+w_{AG})+1+(w_{CH}(q)+w_{CG})$
- $c_{0_j}+c_{0_{j'}}=n+2-n_j$.

6.2.6. Análisis de las comunicaciones

Nos basaremos en el algoritmo basado en la línea segmentada unidireccional doble, obviando los intervalos de tiempo de llenado o vaciado de las líneas. Existen dos flujos de información de sentido opuesto: la información que un procesador P_j debe enviar a P_{j+1} es la parte de la matriz $\mathbf{W}_i$ todavía no anulada, luego será un total de $q(n - c_{0_j} - n_j + 1)$ elementos (la cantidad de tiempo correspondiente se denotará por $T_{C,P_j,i,\rightarrow}(z)$); y la información que deben enviar P_j a P_{j-1} que son las componentes de $\mathbf{x}_{\mathrm{LS}}$ ya calculadas por él y por los procesadores de índice superior, luego será un total de $(n-c_{0_j}+1)$ componentes (la cantidad de tiempo correspondiente será denotada por $T_{C,P_j,i,\leftarrow}(z)$).

Con un modelo afín de comunicaciones punto a punto, los tiempos empleados en estas comunicaciones, en cada iteración, son:

$$\begin{aligned} T_{C,P_j,i,\rightarrow}(z) &= \beta + \tau q(n - c_{0_j} - n_j + 1) \\ T_{C,P_j,i,\leftarrow}(z) &= \beta + \tau(n - c_{0_j} + 1) \end{aligned}$$

El tiempo destinado a transmitir la parte no nula de la matriz $\mathbf{W}_i$ será evidentemente superior al de las componentes de $\mathbf{x}_{\mathrm{LS}}$.

Si todas las comunicaciones pueden realizarse de manera simultánea en el tiempo, el tiempo de comunicación será el máximo de todos ellos. Dicho máximo se da entre el procesador P_0 y P_1. P_0 debe transmitir $q(n - n_0)$ elementos a P_1, mientras que éste debe transmitir $n - n_0$ elementos a P_0, luego este máximo será $q(n - n_0) < qn$.

$$\begin{aligned} T_C^{(A)}(z,p) &= \frac{m}{q} \max_{\forall 0 \leq j \leq p-2} \{T_{C,P_j,i,\rightarrow}(z)\} \\ &< \beta \frac{m}{q} + \tau nm \\ &= \Theta(mn) \end{aligned} \tag{6.8}$$

Si por el contrario, las comunicaciones deben llevarse a cabo en serie:

$$T_C^{(B)}(z,p) = \sum_{i=0}^{m/q-1} \left\{ T_{C,P_0,i,\rightarrow}(z) + T_{C,P_{p-1},i,\leftarrow}(z) + \sum_{j=1}^{p-2} \left[T_{C,P_j,i,\leftarrow}(z) + T_{C,P_j,i,\rightarrow}(z) \right] \right\}$$

donde

$$T_{C,P_0,i,\rightarrow}(z) + T_{C,P_{p-1},i,\leftarrow}(z) = 2\beta + \tau(q+1)(n-n_0) < 2\beta + \tau(q+1)n$$

y

$$\begin{aligned} T_{C,P_j,i,\leftarrow}(z) + T_{C,P_j,i,\rightarrow}(z) &= \beta + \tau q(n - c_{0_j} - n_j + 1) + \beta + \tau(n - c_{0_j} + 1) \\ &= 2\beta + \tau(q+1)(n - c_{0_j} + 1) - \tau q n_j \end{aligned}$$

además, ya que $n_j \leq n - c_{0_j} < n - c_{0_j} + 1$, entonces

$$\begin{aligned} T_{C,P_j,i,\leftarrow}(z) + T_{C,P_j,i,\rightarrow}(z) &< 2\beta + \tau q(n - c_{0_j} + 1) \\ &< 2\beta + \tau q n \end{aligned}$$

Por lo que

$$\begin{aligned} T_C^{(B)}(z,p) &< \frac{m}{q}\left[2\beta + \tau(q+1)n + (p-2)\left(2\beta + \tau q n\right)\right] \\ &< \frac{m}{q}(p-1)(2\beta + \tau q n) \qquad (6.9) \\ &= \Theta\left(pmn\right) \end{aligned}$$

Las comunicaciones para el caso bidireccional doble prácticamente se duplican, pero conservan el orden de magnitud.

6.2.7. Escalabilidad

Si el sistema está perfectamente equilibrado, el tiempo de sobrecarga del algoritmo paralelo es exclusivamente debido a las comunicaciones, por lo que para realizar el estudio de escalabilidad por isoeficiencia debemos hacer que el tiempo secuencial (6.3) ó (6.5), ($\Theta(mn^2)$), sea del mismo orden de magnitud que el tiempo total de sobrecarga, $pT_C^{(A)}(z) = \Theta(pmn)$ o $pT_C^{(B)}(z) = \Theta(p^2mn)$, según el modelo de comunicación a aplicar.

Podemos observar que para el modelo de comunicación (A) la escalabilidad por isoeficiencia implica que n debe crecer como $\Theta(p)$ y para el modelo (B), como $\Theta(p^2)$.

6.2.8. Resultados experimentales

Las pruebas realizadas, cuyos resultados se muestran a continuación, han sido llevadas a cabo en la máquina ccNUMA *Aldebarán*. La solución RLS sólo se calcula cuando puede darse la solución única, es decir cuando $m_i + q_i \geq n$.

Tiempos paralelos

Las figuras 6.16 y 6.17 muestran los tiempos de ejecución comparados entre la implementación en la línea segmentada unidireccional (LSU) y bidireccional (LSB). Los resultados no son tan significativos para valores de n cercanos a m ya que en tal caso el proceso aritmético total sólo se realiza en la última o últimas iteraciones.

Podemos observar, a excepción de cuando $q = 1$, que el tiempo de ejecución para la línea segmentada unidireccional doble es similiar al de la bidireccional. Tenemos dos conceptos que intervienen en el comportamiento de las versiones: el desequilibrado y el tiempo de copia en la cola circular.

Para dos procesadores, la diferencia tan importante de tiempos para $q = 1$ se debe a

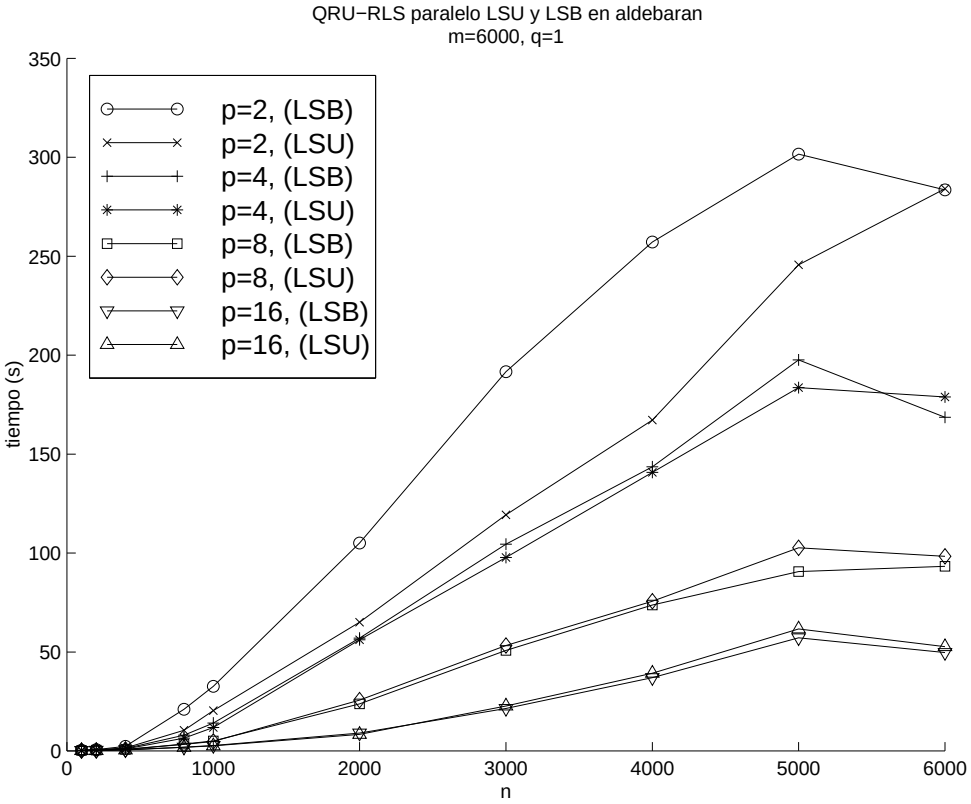

Figura 6.16: Tiempos de ejecución paralelos del algoritmo QRU-RLS para la implementación en la línea segmentada unidireccional doble y bidireccional doble para $q = 1$.

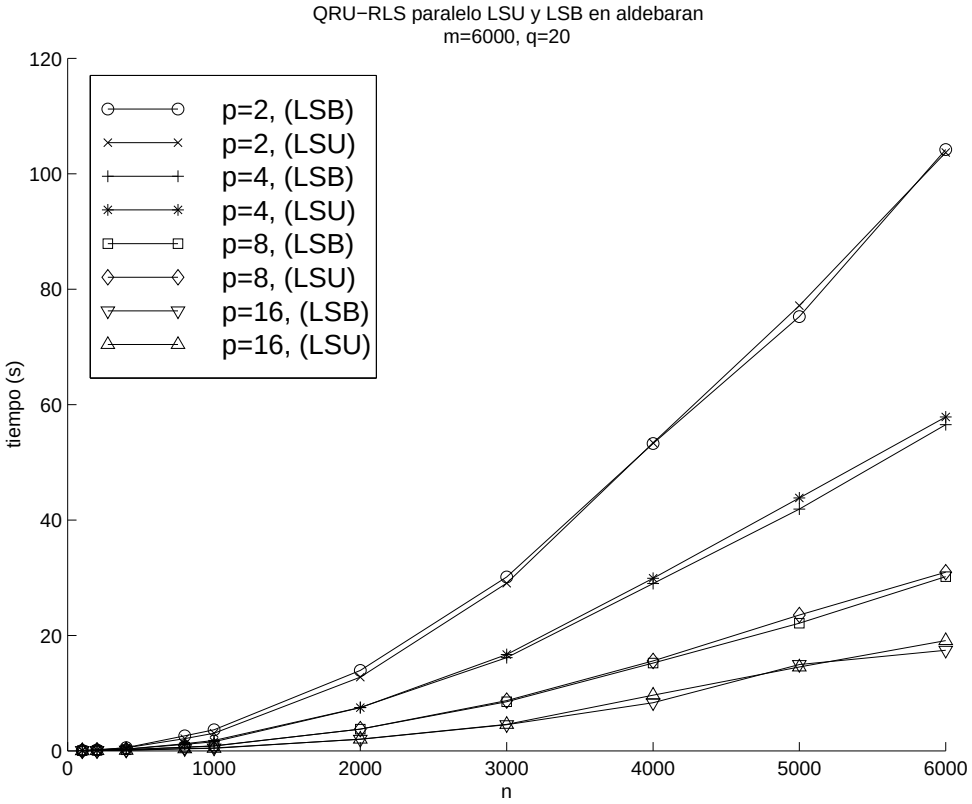

Figura 6.17: Tiempos de ejecución paralelos del algoritmo QRU-RLS para la implementación en la línea segmentada unidireccional doble y bidireccional doble para $q = 20$.

que en el caso unidireccional, sólo el primer procesador realiza copias; en el caso bidireccional los dos procesadores realizan copias (de las tres primeras particiones) lo cual influye notablemente en el tiempo de ejecución. Para un mayor número de procesadores, todos, salvo el último en el caso undireccional, realizan copias, homogeneizándose más la sobrecarga por copia para el caso unidireccional y el bidireccional. La cantidad de iteraciones en el algoritmo es m/q, por lo que para $q = 1$, este proceso se acumula m veces; para valores de q superiores la cantidad de iteraciones es inversamente proporcional. Además, con un valor de q tan pequeño aparece un desequilibrio temporal, debido a que la matriz $\mathbf{R}_{i,a}$ tardará más iteraciones en llenarse, por lo que se tardarán más iteraciones en conseguir el equilibrio previsto.

Tiempo de copia en cola circular

El tiempo de copia es más significativo de lo que podría aparentar en primera instancia. Las figura 6.18 muestra el tiempo empleado en guardar la información para cuatro procesadores en la línea segmentada bidireccional (el descenso del tiempo de copia conforme crece n es debido a que la copia se realiza cuando vaya a ser posible calcular la actualización de la solución RLS, y eso sólo ocurre cuando $(i + 1)q \geq n$; si n es muy grande, sólo ocurrirá en las últimas iteraciones).

Este tiempo va siendo inferior conforme aumenta el número de procesadores, ya que la cantidad de filas asignada a cada uno de ellos va disminuyendo y por lo tanto va siendo menor la cantidad de información a guardar.

Equilibrado de carga

El equilibrado de carga para la línea segmentada unidireccional doble ha resultado ser el previsto, es decir, con cierto desequilibrio debido a la imprecisión del modelo. Esto podemos observarlo en la figura 6.19. Podemos apreciar cómo el último procesador es el que menos tiempo emplea debido a que no realiza copias en la cola circular.

Para el caso bidireccional, también podemos observar un desequilibrio notable para

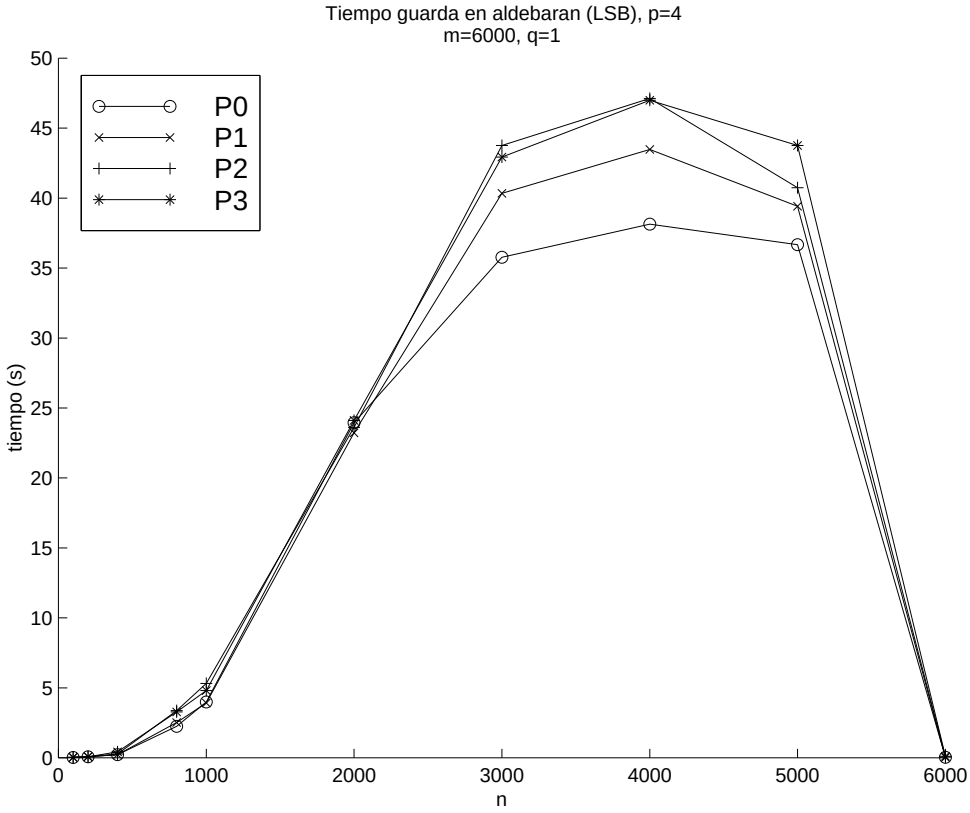

Figura 6.18: Tiempos de copia en la cola circular para la línea segmentada bidireccional.

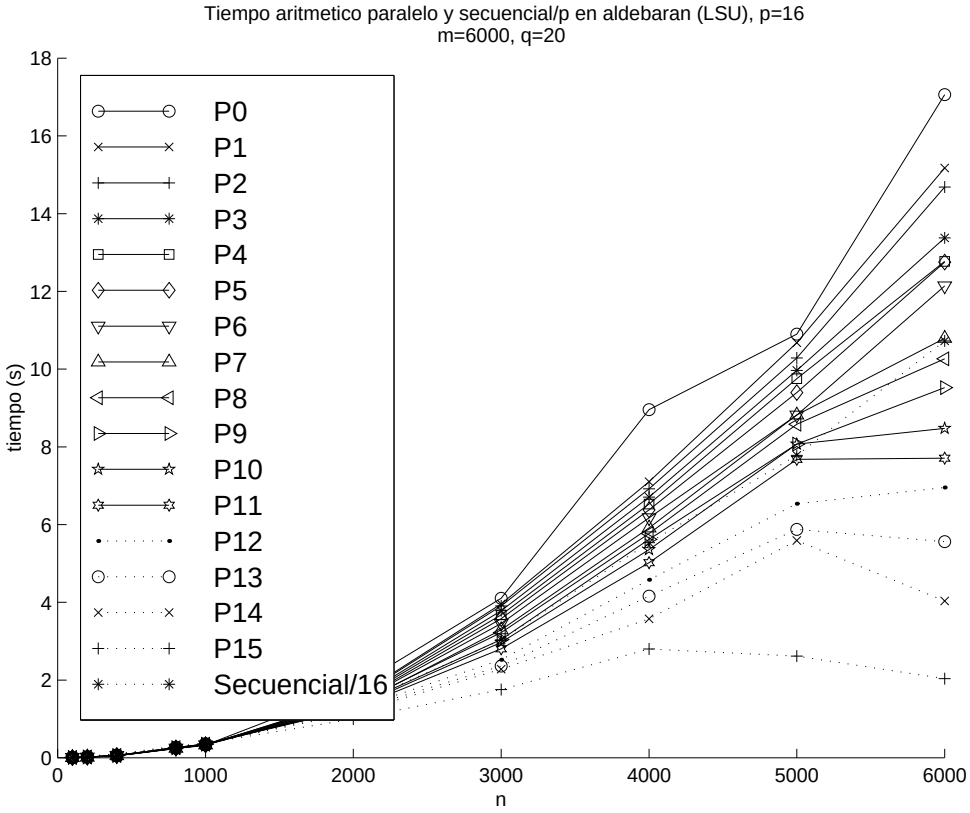

Figura 6.19: Desequilibrado en la línea segmentada unidireccional.

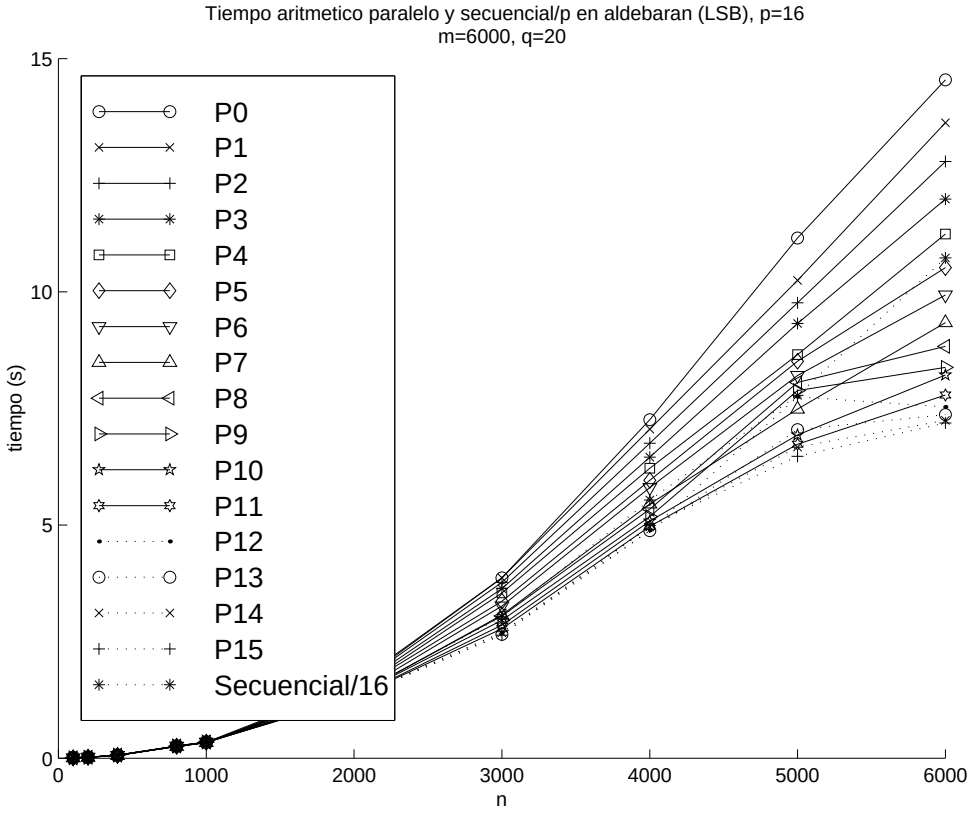

Figura 6.20: Desequilibrado en la línea segmentada bidireccional.

valores de n altos, lo que podemos observar en la figura 6.20.

El origen del desequilibrado es distinto para el caso unidireccional y el bidireccional: el unidireccional estará desequilibrado debido a que el modelo de tiempo de ejecución muy probablemente no será exacto; el bidireccional estará más desequilibrado para valores mayores de n debido a que habrá un retardo importante entre el trabajo de cierta partición y su especular, asignadas ambas al mismo procesador, ya que hasta que no se llena la porción de la matriz $\mathbf{R}_{a,i}$ de cierta partición, no hay trabajo aritmético.

Tiempo de sobrecarga

Hemos estimado el tiempo de sobrecarga restando del tiempo de ejecución paralelo el tiempo ejecución secuencial dividido por el número de procesadores. En las figuras 6.21 y 6.22 se observan los resultados, donde se aprecia que para valores de q pequeños, el tiempo de sobrecarga se debe principalmente al tiempo de copia dada la forma que tienen

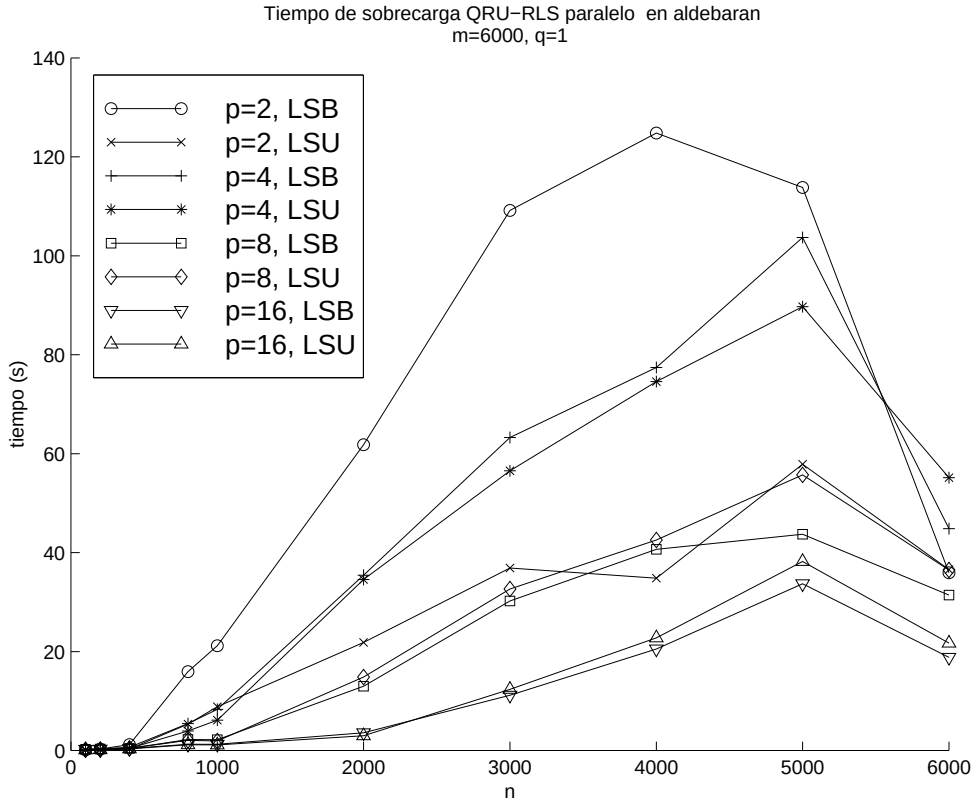

Figura 6.21: Tiempo de sobrecarga para $q = 1$

las curvas. Para valores de q grandes, el tiempo de sobrecarga tiene un aspecto cuadrático, luego interviene de una manera más importante el desequilibrio de la carga. El tiempo de comunicación aparece inmerso en estos tiempos debido a su menor peso.

Eficiencia

Hemos observado que la eficiencia aumenta en términos generales conforme aumenta el parámetro q. Independientemente del valor de q, la cantidad de información a guardar depende de n y de n_j; con valores de q bajos, la cantidad de iteraciones m/q es alta, luego habrá que realizar muchas veces la guarda o copia, mientras que con valores de q altos, dichas veces será proporcionalmente menor, lo cual influye notablemente en la eficiencia, tal y como se puede observar en la eficiencia de las figuras 6.23 y 6.24.

Se observa cómo la eficiencia sube conforme aumenta el valor del parámetro q debido al problema del tiempo de copia. Este problema no es tan importante cuando, siendo $q = 1$,

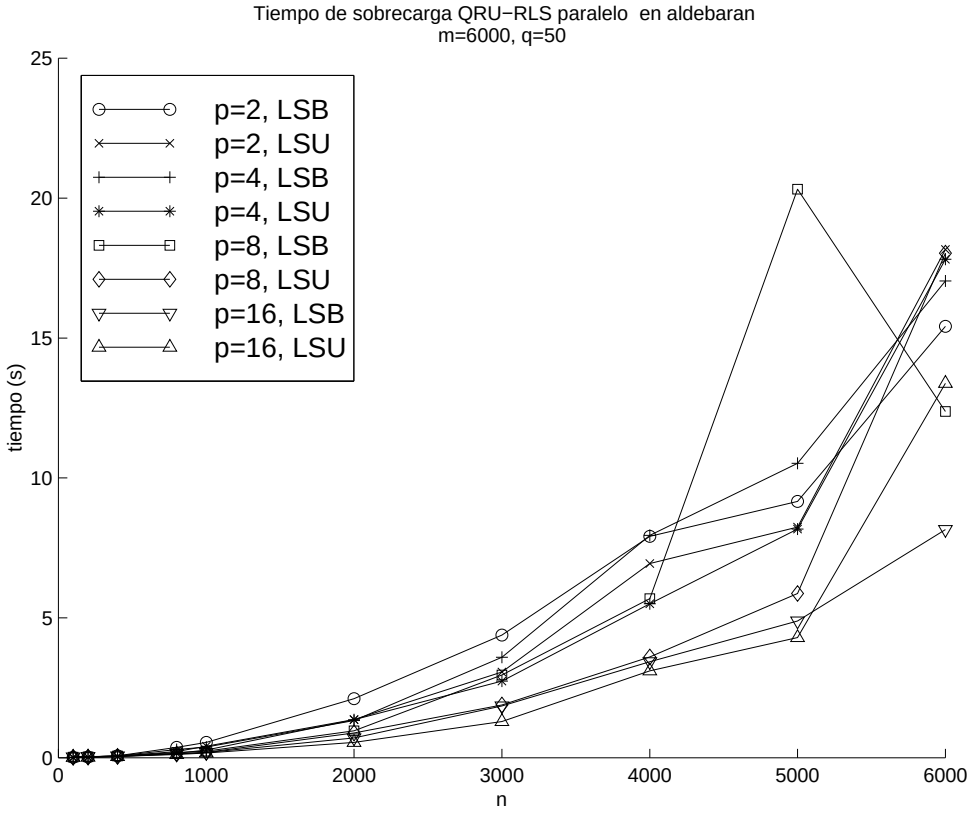

Figura 6.22: Tiempo de sobrecarga para $q = 50$

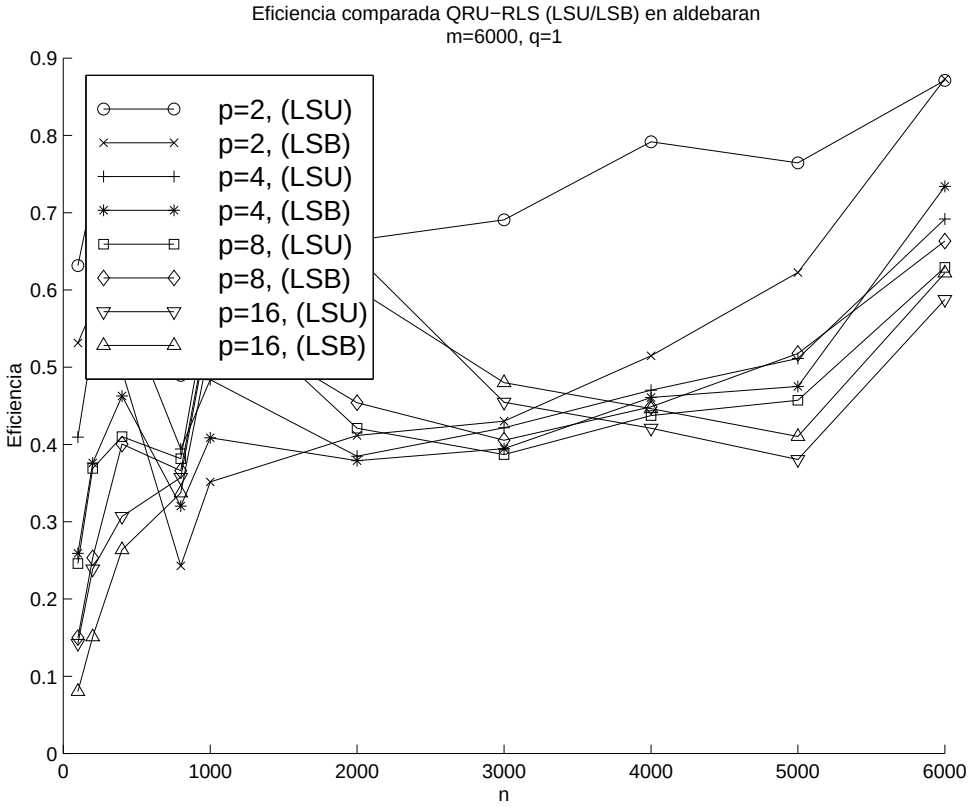

Figura 6.23: Eficiencia comparada para $q = 1$.

se trabaja con sólo dos procesadores en la línea unidireccional por los motivos expuestos anteriormente.

6.3 Conclusiones y posibles mejoras

El algoritmo paralelo obtiene un buen rendimiento para valores de q altos y para valores de n relativamente pequeños comparados con m, por los motivos expuestos anteriormente. Un aspecto que sería deseable mejorar en el algoritmo paralelo es el tamaño relativamente elevado que debe tener la estructura de la cola circular, tanto por el requerimiento de almacenamiento, como por el tiempo necesario para realizar la copia, lo cual incide claramente en el rendimiento del algoritmo. Habría que reconcebir el algoritmo para que no hiciera falta guardar toda esta información para actualizar la solución RLS, siempre y cuando esta reconstrucción no implicara un coste superior al de la copia. Una posibilidad es emplear la formulación vista en la subsección que relaciona este algoritmo con el fil-

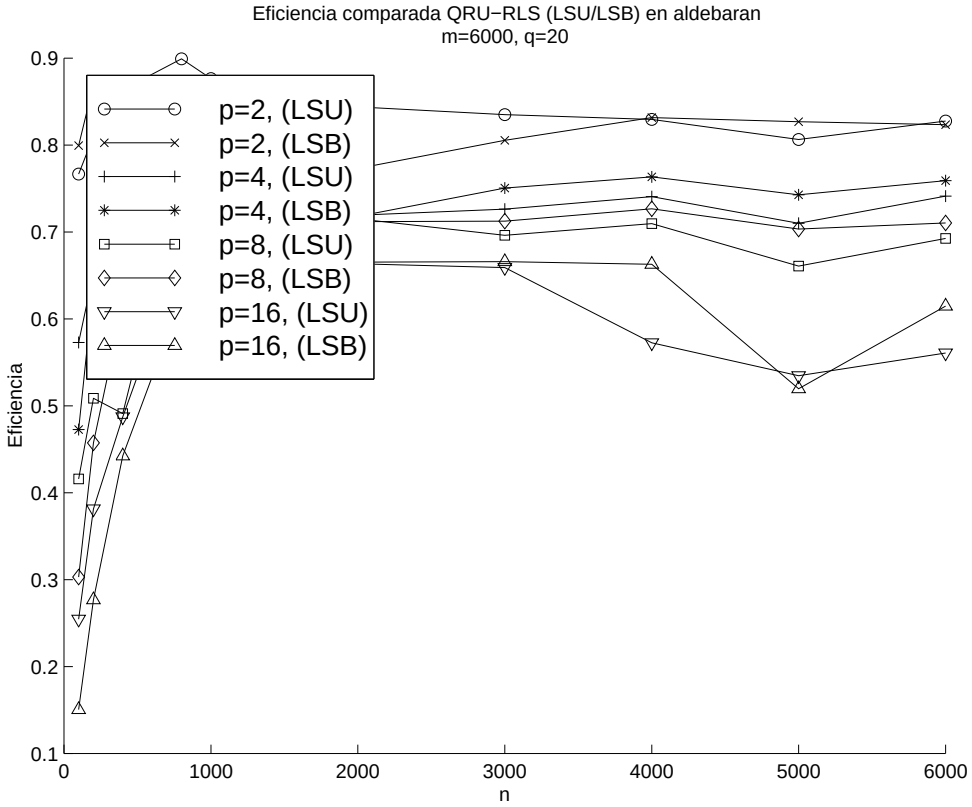

Figura 6.24: Eficiencia comparada para $q = 20$.

tro de información, pero nos encontramos con una nueva matriz a actualizar $\mathbf{R}_{i,a}^{-1}$, lo que requerirá tiempo aritmético, contrarrestando el objetivo de disminuir el tiempo de copia. Otra posible mejora es tener en cuenta el tiempo de copia en las ecuaciones de equilibrado.

Las submatrides triangulares no han sido empaquetadas, por lo que puede resultar de interés el observar el comportamiento temporal del algoritmo, cuando las submatrices son guardadas de manera empaquetada.

Los flujos de información no han empleado las versiones síncronas de los envíos inmediatos, por lo que su transcurso es libre a lo largo de las líneas segmentadas, lo cual es otra fuente potencial de desequilibrio; una sincronización explícita puede ayudar a un equilibrado mejor, pero quizá a expensas de un mayor tiempo de comunicación. Puede ser interesante el observar cuál es el nuevo comportamiento del algoritmo bajo estas circunstancias. Al igual que con el algoritmo SRKF, puede ser interesante la implementación basada en *threads* para la atención a los múltiples flujos de comunicación, intentando au-

mentar la equidad en la atención a los mismo, disminuir retardos, etc., en una versión MPI *thread safe*.

Algunos paquetes de álgebra lineal numérica en paralelo como `ScaLAPCK` [100] pueden realizar la actualización QR en paralelo, utilizando rutinas como `PDGEQRF` junto con `PDORMQR`. Especial interés están teniendo las versiones *fuera de núcleo* (*out of core* — OOC—) como en [101] y [102] en las que las dimensiones de la matriz problema son extremadamente grandes como para que quepan en memoria principal. Nos planteamos como ampliación del estudio realizado, el realizar la comparación con las herramientas suministradas por `ScaLAPACK`, así como la revisión del algoritmo para la versión OOC.

Por último, un equilibrado adaptativo resulta interesante allá donde la carga en los distintos procesadores pueda ser previsiblemente variable con el tiempo.

Al igual que ocurría con los algoritmos descritos en los capítulos anteriores, resulta de interés el conocer el valor óptimo del parámetro q que hace mínimos los tiempos de ejecución.

7

Comparación de los métodos RLS implementados

En este capítulo realizaremos una comparación de las prestaciones de los tres algoritmos secuenciales y sus versiones paralelas expuestos en los capítulos anteriores.

7.1 Esquemas algorítmicos

Los tres métodos estudiados: SRKF-RLS, SRIF-RLS y QRU-RLS poseen muchos denominadores en común: hay que hacer ceros en cierta submatriz de la matriz problema a partir de transformaciones ortogonales (unitarias). En la figura 7.1 se observan las submatrices a anular de color gris.

La actualización de la solución es común para los dos primeros, y además de bajo coste:

$$\begin{aligned} \hat{\mathbf{x}}_{i+1} &= \lambda^{-1/2}\hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i}\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{y}_i - \mathbf{H}_i\hat{\mathbf{x}}_i\right) \\ \mathbf{x}_{\mathrm{LS}_i} &= \left(\lambda^{1/2}\right)^{i+1}\hat{\mathbf{x}}_{i+1} \end{aligned}$$

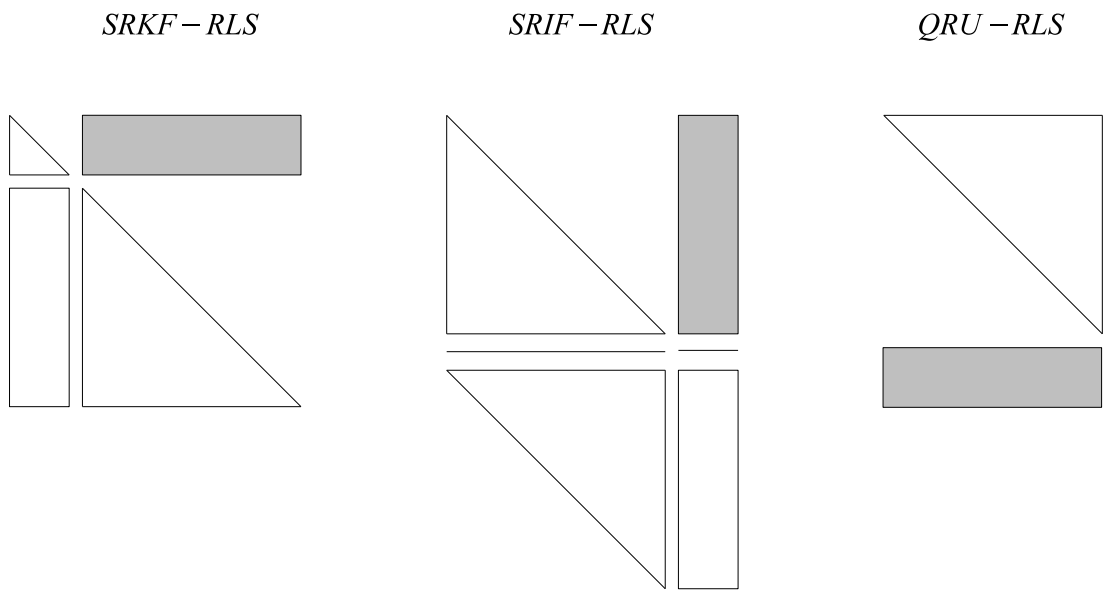

Figura 7.1: Comparación entre los tres métodos RLS.

mientras que en el método QRU-RLS es:

$$\mathbf{x}_{\mathrm{LS}_{i+1}} = \mathbf{R}_{i+1,a}^{-1} \left(\mathbf{Q}_{i+1,a}^{T} \mathbf{y}_{i+1} \right)$$

con un coste superior.

La carga de trabajo en cada versión es:

$$\begin{aligned} W_{\text{sec,SRKF-RLS}}(z) &\approx \left(1 + \frac{1}{2} w_{AG}\right) n^2 m \\ W_{\text{sec,SRIF-RLS}}(z) &\approx \left(w_{AH}(q) + w_{AG}\right) n^2 \frac{m}{q} \\ W_{\text{sec,QRU-RLS}}(z) &\approx \frac{1}{2q} \left(w_{AH}(q) + w_{AG} + \frac{1}{q}\right) n^2 m \end{aligned}$$

y suponiendo que $w_{AH}(q) = 4q$ y $w_{AG} = 6$, [87]:

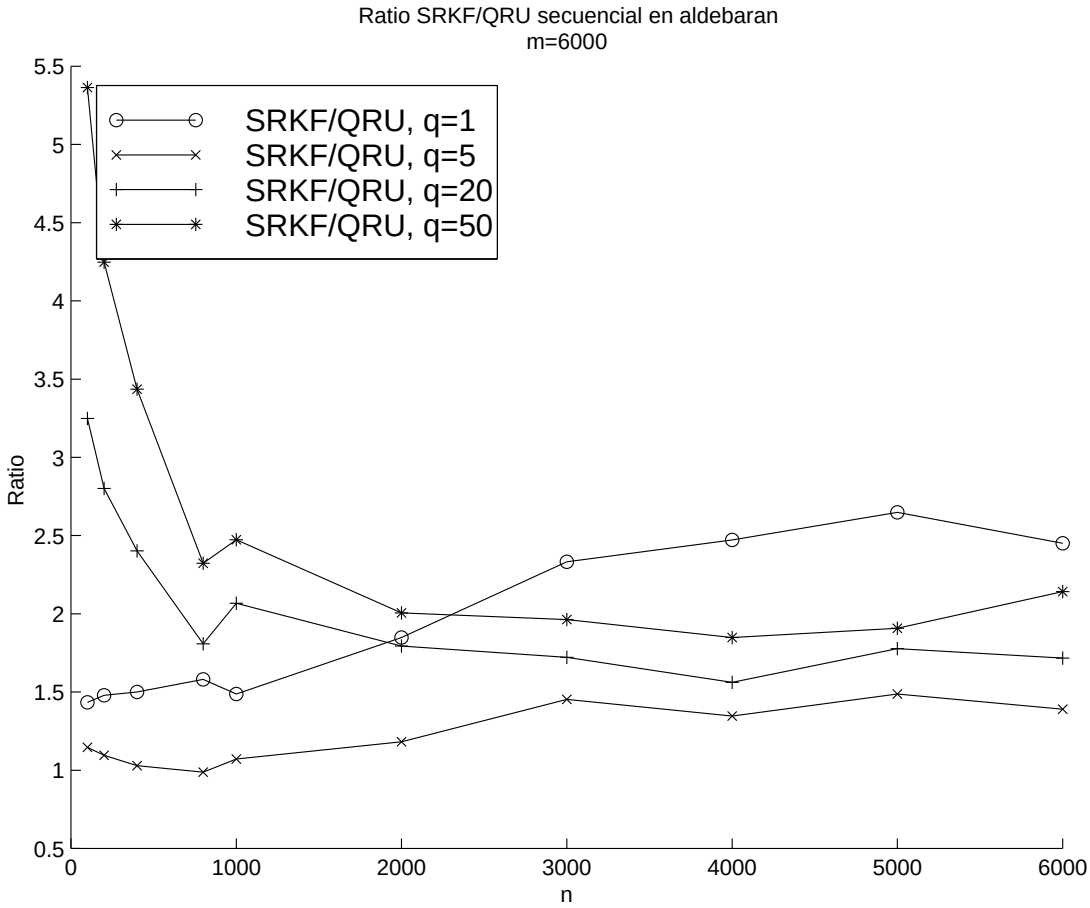

Figura 7.2: Ratios de tiempos de ejecución secuencial del método SRKF y QRU.

$$W_{\text{sec,SRKF-RLS}}(z) \approx 4n^2m \tag{7.1}$$

$$W_{\text{sec,SRIF-RLS}}(z) \approx \left(4 + \frac{6}{q}\right) n^2m \tag{7.2}$$

$$W_{\text{sec,QRU-RLS}}(z) \approx \left(2 + \frac{4}{q}\right) n^2m \tag{7.3}$$

De las expresiones teóricas anteriores en términos generales, se deduce que el método más lento es el SRIF-RLS y los más rápidos, el SRKF-RLS y el QRU-RLS, lo cual se ha verificado en la práctica. Los ratios de los tiempos de ejecución secuencial han resultado los que se muestran en las figuras 7.2, 7.3 y 7.4.

Podemos observar que, a excepción de $q = 1$, asintóticamente las curvas tienden hacia el

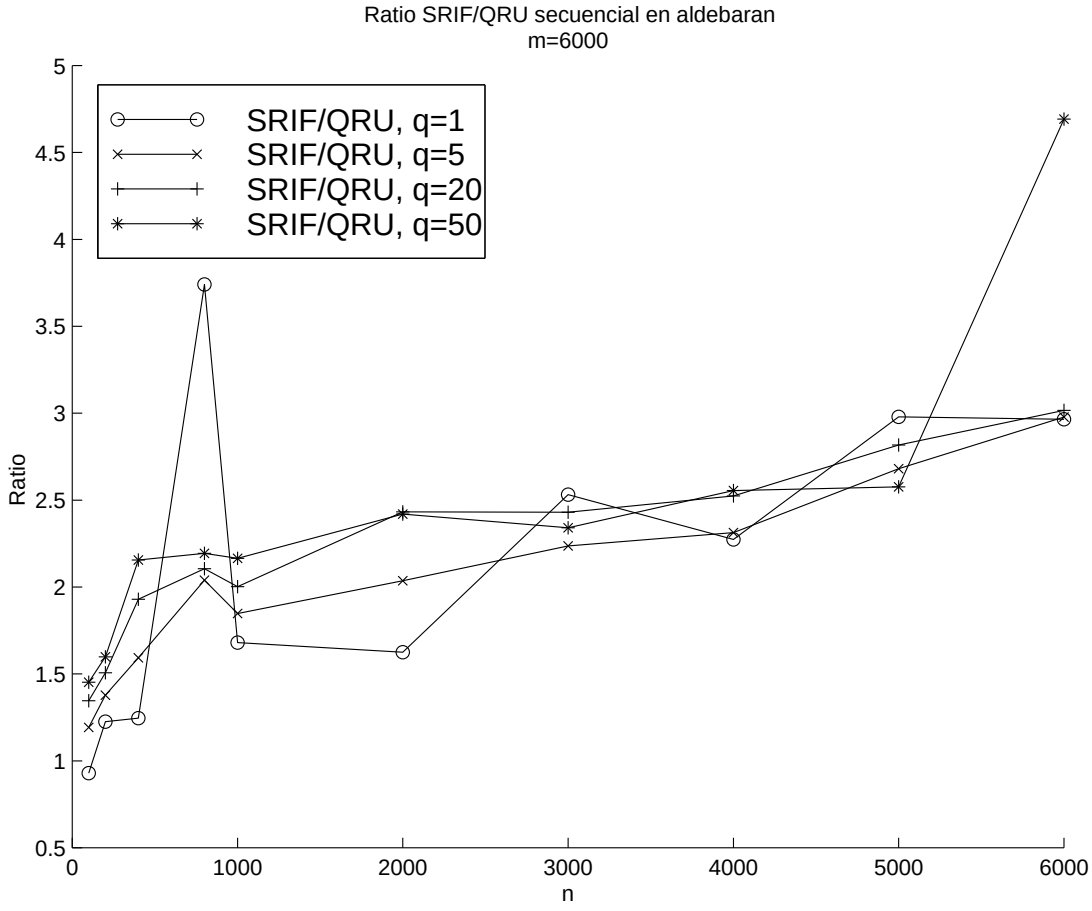

Figura 7.3: Ratios de tiempos de ejecución secuencial del método SRIF y QRU.

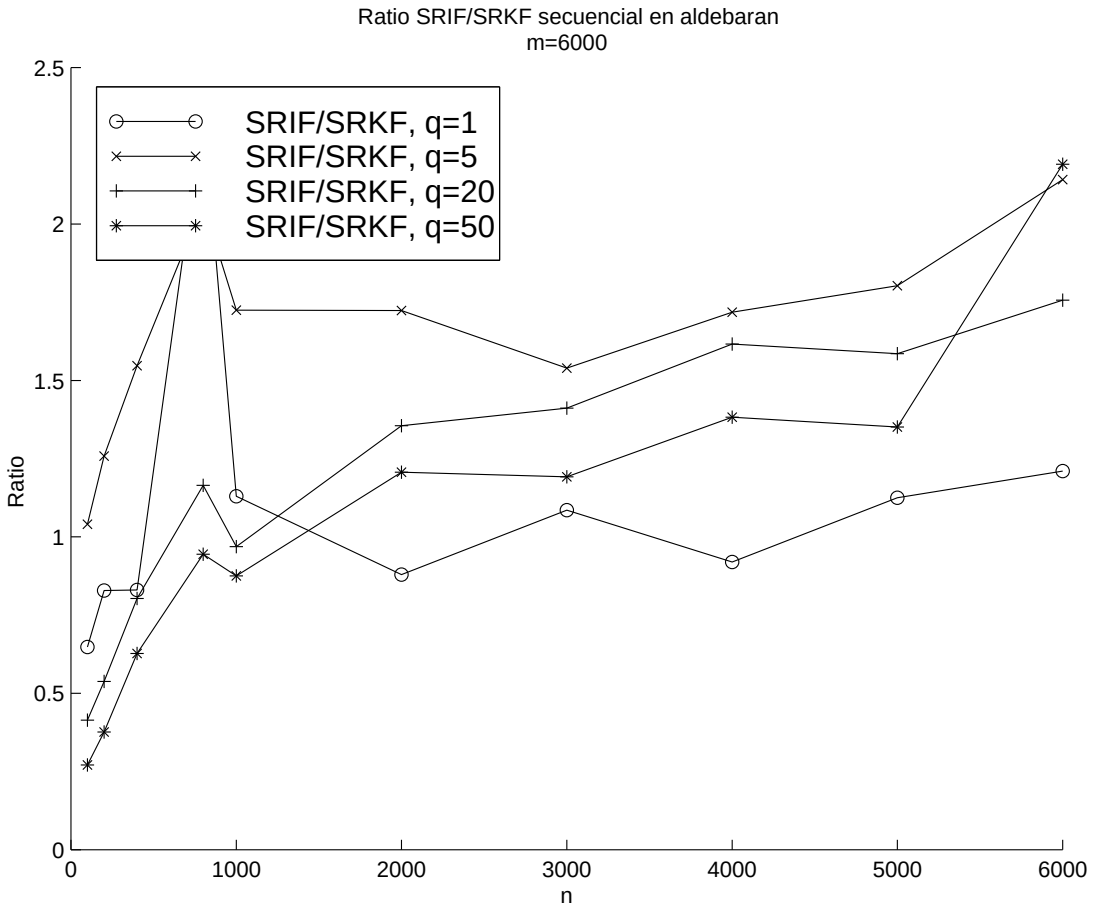

Figura 7.4: Ratios de tiempos de ejecución secuencial del método SRIF y SRKF.

ratio 2.0 en el ratio SRKF/QRU (figura 7.2) conforme va creciendo el valor del parámetro q. En $q = 1$, la situación es excepcional, ya que en el método QRU-RLS no se aplica realmente ninguna transformación de Householder, por lo que el modelo de tiempo de ejecución no se ajusta a la realidad.

Para el ratio SRIF/QRU (figura 7.3) observamos en términos generales un ratio algo superior a 2.0 y con tendencia creciente con n con cierta independencia del valor del parámetro q. Por último, el ratio SRIF/SRKF (figura 7.4) muestra cierta tendencia a la unidad conforme va creciendo el valor de q. Estos comportamiento encajan con las deducciones que se pueden obtener de las expresiones teóricas (7.1), (7.2) y (7.3).

7.2 Eficiencias

El algoritmo secuencial más rápido de todos los implementados es el basado en la factorización QR. Si lo tomamos como algoritmo secuencial de referencia, podemos comparar la eficiencia de la paralelización del propio algoritmo QRU-RLS (versión unidireccional) con el SRKF-RLS (versión unidireccional), lo cual se muestra en la figura 7.5 para 8 procesadores, donde podemos apreciar en términos en términos generales que el algoritmo QRU-RLS posee una mejor eficiencia. Las mismas conclusiones se deducen para el algoritmo SRIF-RLS (figura 7.6).

7.3 Líneas segmentadas unidireccionales y bidireccionales

La intención del rediseño del algoritmo paralelo con las líneas segmentadas bidireccionales tanto para el algoritmo SRKF-RLS como el QRU-RLS era el tener un equilibrado de carga más efectivo e independiente del modelo de tiempo de ejecución que se hubiera calculado. El precio a pagar era un incremento en el tiempo de comunicación y cierto desajuste en la atención a los múltiples flujos de información que aparecen. Las ventajas previstas han sido en ocasiones superadas por los inconvenientes que aparecen, por lo que sólo para algunas combinaciones particulares de parámetros se ha obtenido una clara

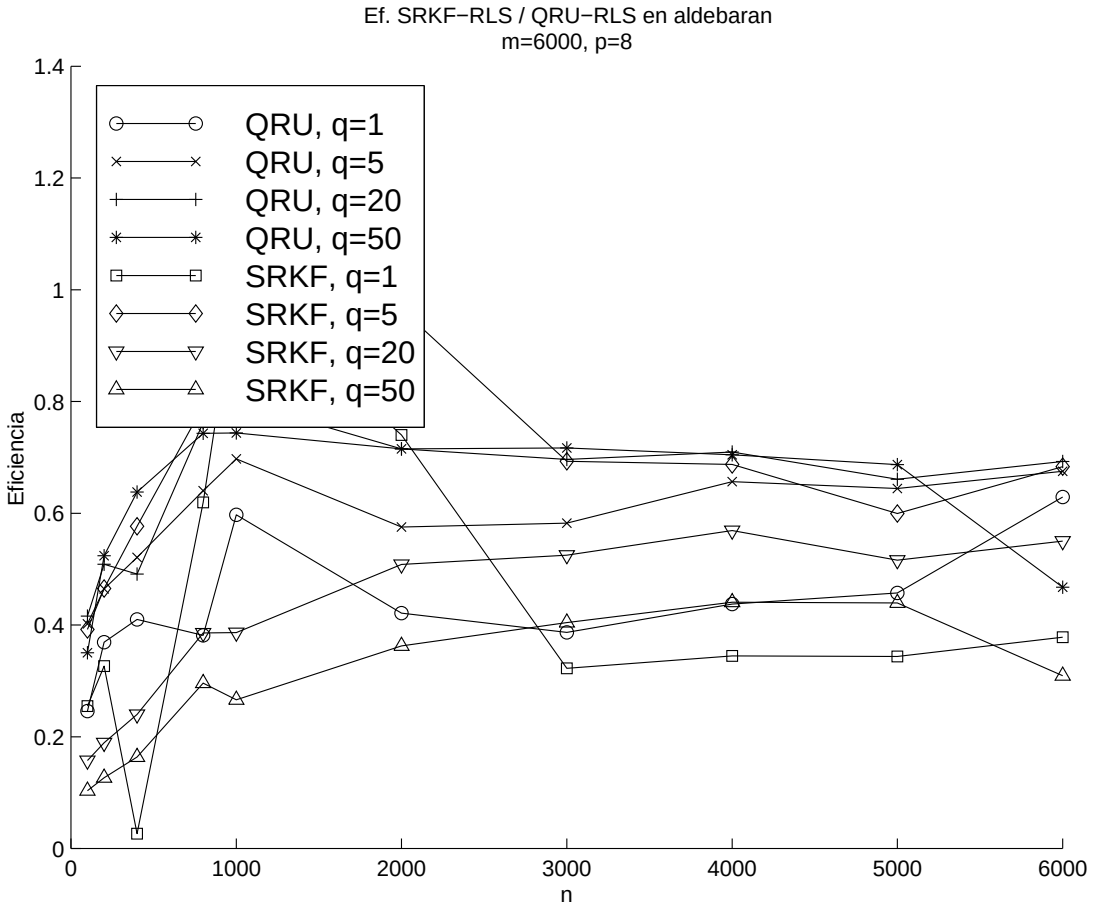

Figura 7.5: Eficiencia de las paralelizaciones QRU-RLS y SRKF-RLS tomando como algoritmo secuencial de referencia el QRU-RLS, con $p = 8$

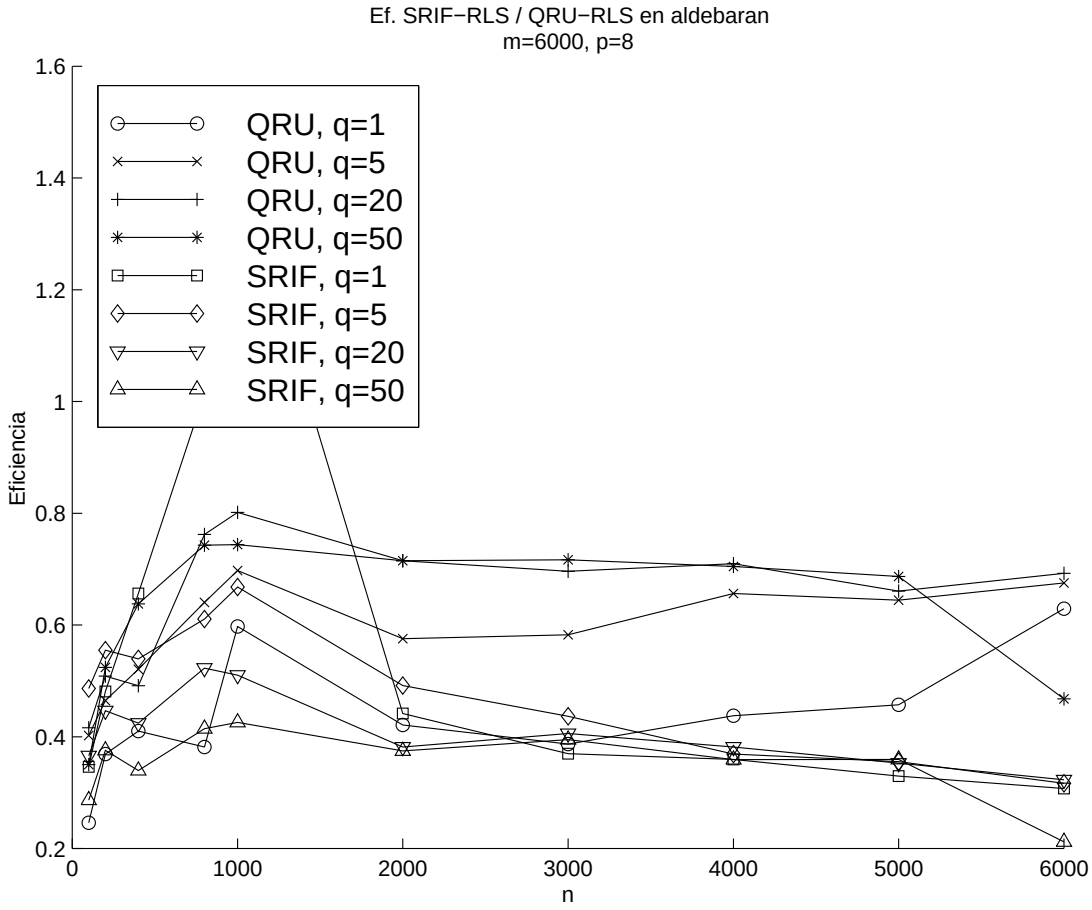

Figura 7.6: Eficiencia de las paralelizaciones QRU-RLS y SRIF-RLS tomando como algoritmo secuencial de referencia el QRU-RLS con $p = 8$

mejora, concretamente para $q = 1$. El equilibrado propuesto para el caso unidireccional, aunque lejos de ser perfecto, ha dado buenos resultados, por lo que los modelos teóricos de tiempos de ejecución se han ajustado de manera relativa en cierta medida a los reales.

7.4 Tiempos de comunicación

Los tres algoritmos paralelos tienen tiempos de comunicación del mismo orden de magnitud. En términos generales, los métodos QRU-RLS y SRIF-RLS tienen tiempos de comunicación similares (compárese (5.11) y (5.13) con (6.8) y (6.9)), aunque el primero establece el doble de transferencias. Los tiempos de comunicación para la versión SRKF-RLS son sutilmente superiores, según se desprende de la comparación con (4.19) y (4.20)

7.5 Escalabilidad

En los tres casos el estudio sobre la escalabilidad arroja las mismas conclusiones, es decir, suponiendo el caso más desfavorable en el que no pueda darse el solapamiento entre el cálculo y las comunicaciones, el tamaño del problema, adoptado como n debe crecer como $\Theta(p)$ u $\Theta(p^2)$ en función de si pueden simultanearse todas las transferencias o deben serializarse respectivamente.

7.6 Conclusiones

Se han implementado un total de cinco versiones secuenciales y otras tantas paralelas. Para el método SRKF, tenemos una única versión secuencial y dos versiones paralelas (LSU y LSB); para el método SRIF se han implementado dos versiones secuenciales (Householder+Givens y Givens), y una única versión paralela (LSU); por último, para el método QRU, se han implementado numerosas versiones secuenciales, de las que han destacado de nuevo dos (Householder+Givens y Givens) y para la primera de ellas se han implementado dos versiones paralelas (LSU y LSB).

De los tres algoritmos secuenciales estudiados, el más rápido es el basado en la actualización de la factorización QR. De las paralelizaciones llevadas a cabo, de nuevo, la basada en la actualización de la factorización QR sigue siendo la más rápida, a pesar de la sobrecarga por copia en la cola circular. Esta es la única desventaja observada, ya que en una implementación VLSI la cantidad de memoria necesaria puede hacer que no sea tan competitivo. Cada variante plantea numerosas posibilidades de mejora, aunque el denominador común radica en el replanteamiento de esquemas de equilibrado tanto estáticos que implican la obtención de modelos de tiempos de ejecución precisos, como adaptativos que permitan redistribuir el trabajo en el tiempo de manera equitativa, si la carga de algún o algunos procesadores varía. Adicionalmente, el estudio de transferencias síncronas entre los procesadores contiguos de la línea segmentada puede dar lugar a resultados que puedan ser de interés. Para el caso de las líneas segmentadas bidireccionales es previsible una mejora en el comportamiento si se emplean versiones *thread safe* de las bibliotecas de comunicaciones.

Allá donde los métodos basados en las rotaciones de Givens hayan sido ventajosos, pueden llegar a serlo aún más aplicando transformaciones de Householder para cierto rango de valores de q. Puede resultar de interés el conocer teórica o heurísticamente el valor óptimo de q que minimiza los tiempos de ejecución. Para algunas versiones puede resultar provechoso el empleo de versiones empaquetadas de los datos.

El interés en estudiar estos tres algoritmos no sólo se circunscribe en abordar el problema RLS secuencial o paralelamente de la mejor forma posible, atendiendo a las condiciones de contorno. Para nuestro trabajo, estos algoritmos nos resultan de interés por formar la base de una serie de algoritmos empleados en ciertas aplicaciones del campo de las Comunicaciones. Así como el método QRU-RLS ha sido el claro ganador, en términos generales, de los algoritmos implementados en este trabajo hasta ahora para el problema RLS, no lo será para este problema de Comunicaciones debido a que su concepción no encaja con las condiciones del nuevo problema a resolver.

8 Aplicaciones

En este capítulo describiremos una aplicación perteneciente al campo de Teoría de la Señal y Comunicaciones. Para su solución, emplearemos dos algoritmos paralelos descritos en capítulos anteriores. Sobre uno de ellos propondremos una modificación que supone un ahorro sustancial del coste computacional.

8.1 Sistemas MIMO

Un sistema MIMO se caracteriza por tener múltiples entradas y múltiples salidas (*Multiple Input Multiple Output*). En la actualidad este tipo de sistemas se usa en el área de Comunicaciones para incrementar el rendimiento del sistema y la inmunidad de la señal frente a interferencias, ruido y desvanecimiento.

En este contexto de Comunicaciones y más concretamente, en comunicaciones inalámbricas, el sistema MIMO se suele establecer teniendo múltiples antenas transmisoras y múltiples antenas receptoras. Estas configuraciones pueden aportar dos ventajas: multiplexación (transmisión de diferentes señales a través de diferentes canales paralelos entre el transmisor y el receptor) y diversidad (transmisión de la misma señal pero por diferentes canales paralelos), aumentando la inmunidad de dicha señal frente a los problemas comentados

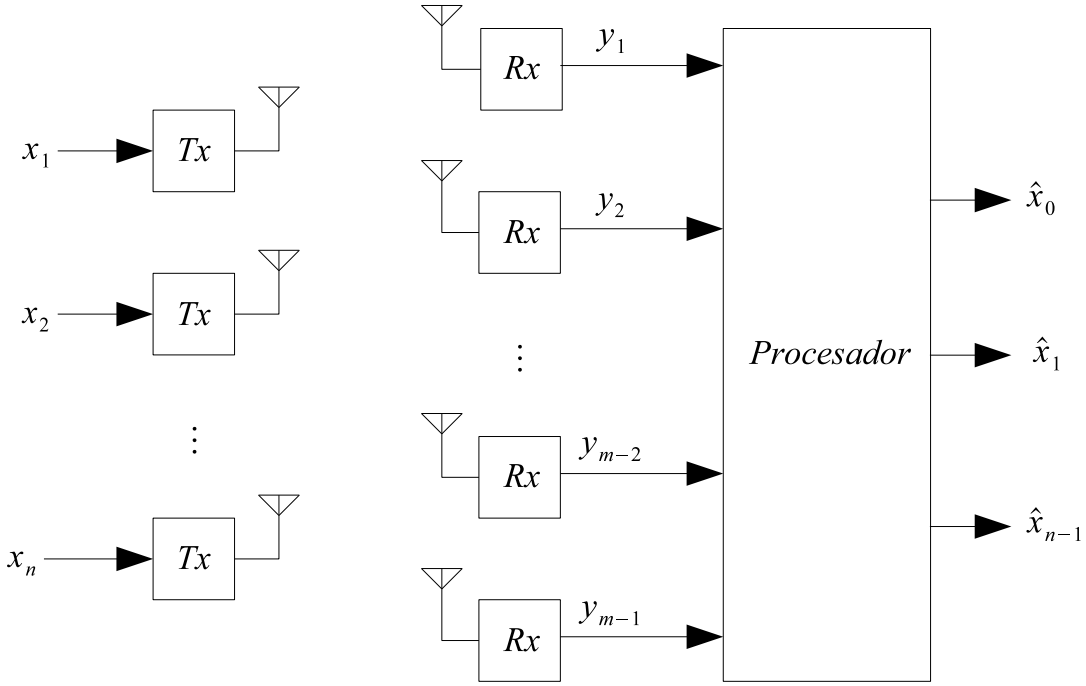

Figura 8.1: Sistema MIMO

anteriormente. Suele existir un compromiso entre ambas ventajas. Los sistemas MIMO pueden establecerse punto a punto, pero también punto-multipunto o multipunto-punto.

V-BLAST (Vertical Bell Laboratories Layered Space-Time Architecture) es una técnica de multiplexación espacial que emplea múltiples antenas transmisoras y múltiples antenas receptoras; cada antena transmisora transmite una porción de la información total, llegando a todas las antenas receptoras por múltiples caminos (figura 8.1).

Con el procesado oportuno se recupera la información original, que algebraicamente podemos describir como

$$\begin{aligned}
\mathbf{y}(t) &= \mathbf{H}\mathbf{x}(t) + \mathbf{v}(t) \\
\mathbf{x}(t) &= (x_1(t), x_2(t), \ldots, x_n(t))^T \\
\mathbf{y}(t) &= (y_1(t), y_2(t), \ldots, y_m(t))^T \\
\mathbf{v}(t) &= (v_1(t), v_2(t), \ldots, v_m(t))^T
\end{aligned}$$

$$\mathbf{H} = \begin{pmatrix} h_{11} & h_{12} & \cdots & h_{1n} \\ h_{21} & h_{22} & \cdots & h_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ h_{m1} & h_{m2} & \cdots & h_{mn} \end{pmatrix}$$

donde $\mathbf{x}(t)$ es un vector que cuyas componentes representan la señal que se transmite por cada antena, $\mathbf{y}(t)$, otro vector cuyas componetes representan la señal que se recibe por cada antena, $\mathbf{v}(t)$ es el ruido en la transmisión, y $\mathbf{H}$ es la matriz del canal que representa la respuesta del mismo a los diferentes canales que pueden establecerse entre cualquier par de antenas (véase la figura 1.1).

El procesado de la señal recibida puede llevarse a cabo de múltiples formas: de manera óptima, por máxima verosimilitud (buscando la secuencia de información que más se parezca a la secuencia recibida), dando lugar a métodos como la *Decodificación Esférica* o *Sphere Decoding*, o subóptima, como por ejemplo empleando detección lineal. Evidentemente, los mejores resultados se obtiene de la primera forma, pero su coste computacional suele ser más elevado que de la segunda. Nosotros abordaremos uno de los métodos que mejores prestaciones proporcionan para la detección lineal.

Dentro de los métodos de detección lineal, podemos encontrar el más sencillo que consiste en estimar $\mathbf{x}$, como $\hat{\mathbf{x}}$ a partir de la pseudoinversa de la matriz del canal, y a continuación cuantificar los resultados: $\hat{\mathbf{x}} = \lfloor \mathbf{H}^{\dagger}\mathbf{y} \rceil$, donde $\lfloor \cdot \rceil$ denota la traducción, *mapeo* o cuantificación de la solución algebraica al conjunto de símbolos. Esta estimación es la denominada de Babai, o detección por forzador de zeros (*zero forcing —ZF—*). El principal problema de esta modalidad radica en el realce del ruido en canales con ciertas características (gran dispersión de autovalores). El método alternativo es realizar una estimación por mínimos cuadradados medios o MMSE (*Minimum Mean Square Error Estimation)*: $\hat{\mathbf{x}} = \lfloor \left(\mathbf{H}^{*}\mathbf{H} + \alpha^{2}\mathbf{I}\right)^{-1}\mathbf{H}^{*}\mathbf{y} \rceil$, donde α^{-1} se interpreta como cierta relación señal-ruido, proporcionando mejores resultados.

Existe una variedad para los métodos de detección lineal denominada SIC (*Succesive Interference Cancellation*) basada en realizar la decodificación símbolo a símbolo, o componente a componente, eliminando su supuesta influencia en la señal recibida, de manera sucesiva. La decodificación de cada componente puede hacerse bien empleando la ténica ZF, bien MMSE.

Una última alternativa, basada en la anterior, denominada OSIC (*Ordered SIC*) consiste en ir decodificando primeramente los símbolos o componentes con mejor relación señal-ruido o error absoluto y después los de menor, siendo ésta la opción que mejor resultados proporciona. Esta es la técnica en la que está basado V-BLAST.

Existe una gran cantidad de algoritmos adicionales también subóptimos como *Lattice Reduction*, [103], basados en programación semidefinida, [30], *Second Order Cone Programming*, [104], variantes de V-BLAST con compensación de error, [105], *Markov Chain Montecarlo*, [106], etc.

8.2 OSIC

En [22] podemos encontrar un método para la *descodificación por cancelación de interferencia sucesiva y ordenada* (Ordering and Succesive Interference Cancelling —OSIC—) empleado en V-BLAST y una aplicación para la detección de señales multiportadora en [107], donde la dimensión del problema puede alcanzar varios miles. En este sistema MIMO, en primera instancia estamos interesados en resolver el típico sistema perturbado $\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v}$, donde la matriz $\mathbf{H} \in \mathbb{C}^{m \times n}$, supuestamente de rango completo, representa la *matriz de canal* y cualquier manipulación de los símbolos previamente a su transmisión, $\mathbf{x}$ es un vector cuyas componentes pertenecen a un conjunto de símbolos discreto, y $\mathbf{v}$ es el ruido del proceso La estimación de MMSE (Minimum Mean Square Error estimation) proporciona como solución:

$$\hat{\mathbf{x}} = \left\lfloor \begin{pmatrix} \mathbf{H} \\ \sqrt{\alpha}\mathbf{I}_n \end{pmatrix}^{\dagger} \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix} \right\rfloor = \left\lfloor \mathbf{H}_{\alpha}^{\dagger}\mathbf{y} \right\rceil \tag{8.1}$$

donde α^{-1} puede ser interpretada como una relación señal-ruido y $\mathbf{H}_{\alpha}^{\dagger} \in \mathbb{C}^{n\times m}$ denota las primeras m columnas de la pseudoinversa de $(\mathbf{H}^*, \sqrt{\alpha}\mathbf{I}_n)^*$. En OSIC, las componentes $x_i, i = 1, \ldots n$, son decodificadas en cierto orden: primeramente la más fuerte (con mayor relación señal-ruido) finalizando por la más débil, cancelando la contribución de la componente decodificada en la señal recibida; con la decodificación de cada componente se debe repetir el proceso. La estimación de cada componente requiere de un *mapeo* o traducción del resultado algebraico real o complejo al conjunto discreto de símbolos. Sea $\mathbf{P}$ la matriz de covarianza del error en la estimación de la solución (matriz simétrica y definida positiva)

$$\mathbf{P} = \mathrm{E}\{(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^*\} = (\alpha\mathbf{I}_n + \mathbf{H}^*\mathbf{H})^{-1}$$

Si $\mathbf{P}_{jj} = \mathrm{mín}\{\mathrm{diag}(\mathbf{P})\}$, entonces x_j es la componente de $\hat{\mathbf{x}}$ con mayor relación señal-ruido que puede ser decodificada como

$$\hat{x}_j = \left\lfloor \mathbf{H}_{\alpha,j}^{\dagger}\mathbf{y} \right\rceil$$

donde $\mathbf{H}_{\alpha,j}^{\dagger}$ es la j-ésima fila de $\mathbf{H}_{\alpha}^{\dagger}$ (el denominado j-ésimo *vector anulador*). Si $\mathbf{H} = (\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h_n})$ entonces podemos cancelar la contribución de la estimación de x_j en la señal recibida como

$$\mathbf{y}' = \mathbf{y} - \mathbf{h}_j\hat{x}_j \tag{8.2}$$

El procedimiento debe ser repetido para la matriz de canal *reducida* $\mathbf{H}'$, obtenida eliminando la j-ésima columna de $\mathbf{H}$. Ello implica que debemos recalcular la pseudoinversa de la matriz de canal reducida y seleccionar la nueva componente más fuerte, aplicando los mismos cálculos.

El cálculo de las pseudoinversa de las matrices de canal reducidas iterativamente, hasta que todas las componentes de la solución han sido decodificadas representa una alto coste computacional. Este cálculo iterativo de las pseudoinversas es evitado en [22].

Si las estimaciones $\hat{x}_j$ pertenecieran al cuerpo de los complejos (o reales), no existiría diferencia entre aplicar un método ordenado en la decodificación o no. Al tener que *mapear* o traducir las soluciones algebraicas al subconjunto de los símbolos, no podemos plantear un esquema LS o RLS convencional a (8.1), sin tener que resolver iterativamente las pseudoinversas de las matrices de canal reducidas.

Sea

$$\mathbf{P}^{-1} = \mathbf{P}^{-*/2}\mathbf{P}^{-1/2} \tag{8.3}$$

la factorización de Cholesky de $\mathbf{P}^{-1} = (\alpha\mathbf{I}_n + \mathbf{H}^*\mathbf{H})$, entonces el factor raíz cuadrada $\mathbf{P}^{-1/2}$ y su inversa $\mathbf{P}^{1/2}$ son matrices triangulares superiores.

Si calculamos la siguiente factorización QR:

$$\begin{pmatrix} \mathbf{H} \\ \sqrt{\alpha}\mathbf{I}_n \end{pmatrix} = \mathbf{QR} \tag{8.4}$$

donde $\mathbf{R} \in \mathbb{C}^{n\times n}$, las columnas de $\mathbf{Q} \in \mathbb{C}^{(m+n)\times n}$ son ortogonales, y definimos

$$\mathbf{Q}_\alpha = (\mathbf{q}_{\alpha,1}, \mathbf{q}_{\alpha,2}, \ldots \mathbf{q}_{\alpha,n})$$

como las primeras m filas y n columnas de $\mathbf{Q}$, entonces

$$\begin{aligned}
\mathbf{P}^{-1} &= \mathbf{P}^{-*/2}\mathbf{P}^{-1/2} \\
&= (\alpha\mathbf{I}_n + \mathbf{H}^*\mathbf{H}) \\
&= \begin{pmatrix} \mathbf{H}^* & \sqrt{\alpha}\mathbf{I}_n \end{pmatrix} \begin{pmatrix} \mathbf{H} \\ \sqrt{\alpha}\mathbf{I}_n \end{pmatrix} \\
&= \mathbf{R}^*\mathbf{Q}^*\mathbf{Q}\mathbf{R} \\
&= \mathbf{R}^*\mathbf{R}
\end{aligned}$$

pudiendo identificar $\mathbf{R} = \mathbf{P}^{-1/2}$, verificando que

$$\mathbf{H}_\alpha^\dagger = \mathbf{P}^{1/2}\mathbf{Q}_\alpha^* \tag{8.5}$$

Supongamos que las filas de la matriz triangular superior $\mathbf{P}^{1/2}$ están ordenadas por norma euclídea decreciente (en caso contrario, podemos conseguirlo utilizando una matriz de permutación $\mathbf{\Pi}$ y un transformación unitaria $\mathbf{\Sigma}$ en $\mathbf{\Pi}\mathbf{P}^{1/2}\mathbf{\Sigma}\mathbf{\Sigma}^*\mathbf{Q}_\alpha^*$ manteniendo la estructura de matriz triangular superior en $\mathbf{\Pi}\mathbf{P}^{1/2}\mathbf{\Sigma}$; la permutación también debe ser aplicada a la solución; el coste de estas operaciones puede comprobarse que es $\Theta(n^2)+\Theta(mn)$); entonces, las entradas diagonales de $\mathbf{P}$ estarán ordenadas por valor decreciente y consecuentemente los símbolos en el vector $\hat{\mathbf{x}}$ estarán ordenados por relación señal-ruido creciente. La primera componente a decodificar será entonces $\hat{x}_n$:

$$\hat{x}_n = \lfloor \mathbf{p}_n^{1/2}\mathbf{Q}_\alpha^*\mathbf{y} \rceil$$

donde $\mathbf{p}_n^{1/2} = (\mathbf{0}_{n-1}, p_n^{1/2})$ es la última fila de $\mathbf{P}^{1/2}$, de modo que

$$\hat{x}_n = \lfloor p_n^{1/2}\mathbf{q}_{\alpha,n}^*\mathbf{y} \rceil$$

por lo que el vector anulador para la n-ésima componente resulta ser

$$\mathbf{H}_{\alpha,n}^{\dagger} = p_n^{1/2}\mathbf{q}_{\alpha,n}^{*}$$

Ahora debemos cancelar la influencia de $\hat{x}_n$ en la señal recibida utilizando (8.2) para $j = n$, y repetir el proceso utilizando ahora la matriz de canal reducida $\mathbf{H}' = (\mathbf{h_1}, \mathbf{h_2}, \ldots, \mathbf{h}_{n-1})$. La nueva factorización QR es:

$$\begin{pmatrix} \mathbf{H}' \\ \sqrt{\alpha}\mathbf{I}_{n-1} \end{pmatrix} = \mathbf{Q}'\mathbf{R}'$$

Entonces, si definimos $\mathbf{Q}'_{\alpha}$ como las primeras m filas y $n-1$ columnas de $\mathbf{Q}'$, podemos verificar que

$$\mathbf{Q}_{\alpha} = \begin{pmatrix} \mathbf{Q}'_{\alpha}, & \mathbf{q}_{\alpha,n} \end{pmatrix}, \quad \text{y} \quad \mathbf{R} = \begin{pmatrix} \mathbf{R}' & \times \\ \mathbf{0} & p_n^{1/2} \end{pmatrix}$$

donde no estaremos interesados en las entradas denotadas con $\times$. Así pues, no es necesario recalcular de nuevo estas matrices, y en general:

$$\mathbf{H}_{\alpha,j}^{\dagger} = p_j^{1/2}\mathbf{q}_{\alpha,j}^{*} \tag{8.6}$$

donde $p_j^{1/2}$ es la j-ésima entrada diagonal de $\mathbf{P}^{1/2} = \mathbf{R}^{-1}$, para $j = 1, \ldots, n$ no siendo necesario obtener la pseudoinversa de una matriz de canal reducida sucesiva o iterativamente. Por lo que todos los datos necesarios para obtener todos los vectores anuladores son $\mathbf{P}^{1/2}$ y $\mathbf{Q}_{\alpha}$.

A continuación se muestra una alternativa para el cálculo de $\mathbf{H}_{\alpha}^{\dagger}$. Supongamos de momento que estamos interesados en resolver (8.1) como un problema MMSE, pero sin la metodología OSIC. $\hat{\mathbf{x}}$ podría haber sido obtenido iterativamente utilizando un filtro de Kalman, [22], como:

$$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i + \overline{\mathbf{K}}_{p,i} r_{e,i}^{1/2} \left(\mathbf{y}_i - \mathbf{H}_i \hat{\mathbf{x}}_i \right), \ i = 0, \ldots m-1 \tag{8.7}$$

donde $\mathbf{H}_i \in \mathbb{C}^{1\times n}$ es la i-ésima fila de $\mathbf{H}$ e $\mathbf{y}_i \in \mathbb{C}$ es la i-ésima componente de $\mathbf{y}$, para $i = 0, \ldots, m-1$, y

$$\begin{array}{lll} \mathbf{P}_{(0)} = \dfrac{1}{\alpha}\mathbf{I}_n & r_{e,i} = 1 + \mathbf{H}_i \mathbf{P}_{(i)} \mathbf{H}_i^* & \overline{\mathbf{K}}_{p,i} = \mathbf{P}_{(i)} \mathbf{H}_i^* r_{e,i}^{-*/2} \\ \mathbf{P}_{(i+1)} = (\mathbf{P}_{(i)}^{-1} + \mathbf{H}_i^* \mathbf{H}_i)^{-1} & \hat{\mathbf{x}}_0 = \mathbf{0} & \hat{\mathbf{x}} = \hat{\mathbf{x}}_m \end{array}$$

La condición inicial de que $\mathbf{P}_0 = 1/\alpha \mathbf{I}_n$ es debido a que

$$\mathbf{P} = (\alpha \mathbf{I}_n + \mathbf{H}^* \mathbf{H})^{-1} = \left(\alpha \mathbf{I}_n + \sum_{j=0}^{m-1} \mathbf{H}_j^* \mathbf{H}_j \right)^{-1}$$

donde $\mathbf{H}_j$ es la j-ésima fila de $\mathbf{H}$, por lo que si definimos

$$\mathbf{P}_{(i)} = \left(\alpha \mathbf{I}_n + \sum_{j=0}^{i-1} \mathbf{H}_j^* \mathbf{H}_j \right)^{-1}$$

siendo por tanto $\mathbf{P} = \mathbf{P}_{(m)}$, o bien

$$\begin{aligned} \mathbf{P}_{(i)}^{-1} &= \left(\alpha \mathbf{I}_n + \sum_{j=0}^{i-1} \mathbf{H}_j^* \mathbf{H}_j \right) \\ &= \mathbf{P}_{(i-1)}^{-1} + \mathbf{H}_{i-1}^* \mathbf{H}_{i-1} \\ \mathbf{P}_{(i)} &= \left(\mathbf{P}_{(i-1)}^{-1} + \mathbf{H}_{i-1}^* \mathbf{H}_{i-1} \right)^{-1} \end{aligned}$$

y utilizamos el lema de inversión de matrices, entonces:

$$\mathbf{P}_{(i)} = \mathbf{P}_{(i-1)} - \frac{\mathbf{P}_{(i-1)} \mathbf{H}_i^* \mathbf{H}_i \mathbf{P}_{(i-1)}}{r_{e,i}}$$

con $r_{e,i} = 1 + \mathbf{H}_i \mathbf{P}_{(i-1)} \mathbf{H}_i^*$ y por tanto $\mathbf{P}_{(0)} = 1/\alpha \mathbf{I}_n$.

Supongamos a continuación una situación *artificial* en la que $\mathbf{y} = \mathbf{e}_j$, donde $\mathbf{e}_j$ es un

vector columna de m componentes nulo a excepción de la entrada j-ésima en la que hay un 1. Sea $\hat{\mathbf{x}}^{(\mathbf{e}_j)}$ la solución a (8.1) obtenida por medio de (8.7) para este valor particular de $\mathbf{y}$, así pues

$$\hat{\mathbf{x}}^{(\mathbf{e}_j)} = \mathbf{H}_\alpha^\dagger \mathbf{e}_j$$

Si organizamos los resultados en forma de vector fila para $j = 1, \ldots, m$, entonces

$$\left(\hat{\mathbf{x}}^{(\mathbf{e}_1)}, \hat{\mathbf{x}}^{(\mathbf{e}_2)}, \cdots, \hat{\mathbf{x}}^{(\mathbf{e}_m)}\right) = \mathbf{H}_\alpha^\dagger \left(\mathbf{e}_1, \mathbf{e}_1, \cdots, \mathbf{e}_m\right) = \mathbf{H}_\alpha^\dagger \mathbf{I}_m = \mathbf{H}_\alpha^\dagger$$

Cada $\hat{\mathbf{x}}^{(\mathbf{e}_j)}$ podría haber sido obtenido de forma iterativa, por lo que podríamos concebir $\mathbf{H}_\alpha^\dagger$ de forma iterativa también:

$$\left(\hat{\mathbf{x}}_{i+1}^{(\mathbf{e}_1)}, \hat{\mathbf{x}}_{i+1}^{(\mathbf{e}_2)}, \cdots, \hat{\mathbf{x}}_{i+1}^{(\mathbf{e}_m)}\right) = \mathbf{H}_{\alpha,(i+1)}^\dagger, \ i = 0, \ldots, m-1 \tag{8.8}$$

de modo que $\mathbf{H}_\alpha^\dagger = \mathbf{H}_{\alpha,(m)}^\dagger$. Para evitar confusiones en la notación, emplearemos el subíndice entre paréntesis (i) para denotar cada una de las instancias de $\mathbf{H}_{\alpha,(i)}^\dagger$ en las iteraciones del algoritmo. Reescribiendo (8.7) en forma de vector y para $\mathbf{H}_\alpha^\dagger$, entonces para $i = 0, \ldots m-1$ y $\mathbf{H}_{\alpha,(0)}^\dagger = \mathbf{0}$:

$$\mathbf{H}_{\alpha,(i+1)}^\dagger = \mathbf{H}_{\alpha,(i)}^\dagger + \overline{\mathbf{K}}_{p,i} r_{e,i}^{1/2} \left(\mathbf{e}_{i+1}^* - \mathbf{H}_i \mathbf{H}_{\alpha,(i)}^\dagger\right) \tag{8.9}$$

De modo que obtenemos una manera alternativa de obtener $\mathbf{H}_\alpha^\dagger$ utilizando un filtro de Kalman.

Podemos generalizar (8.9) procesando un bloque de q filas consecutivas de $\mathbf{H}$ (supondremos de nuevo que m/q es un entero), por lo que $\mathbf{H}_i \in \mathbb{C}^{q \times n}$. Entonces para $i = 0, \ldots m/q-1$, y $\mathbf{H}_{\alpha,(0)}^\dagger = \mathbf{0}$, de nuevo:

$$\mathbf{H}_{\alpha,(i+1)}^\dagger = \mathbf{H}_{\alpha,(i)}^\dagger + \overline{\mathbf{K}}_{p,i} \mathbf{R}_{e,i}^{1/2} \left(\mathbf{\Gamma}_{i+1}^* - \mathbf{H}_i \mathbf{H}_{\alpha,(i)}^\dagger\right) \tag{8.10}$$

donde $\mathbf{\Gamma}_{i+1} = \left(\mathbf{0}^T_{iq\times q}, \mathbf{I}_q, \mathbf{0}^T_{(m-q(i+1))\times q}\right)^T \in \mathbb{R}^{m\times q}$, y en la última iteración, $\mathbf{H}^\dagger_\alpha = \mathbf{H}^\dagger_{\alpha,(m/q)}$.

Se pueden evitar inversiones de matrices o formas cuadráticas utilizando algunas de las variantes algorítmicas del filtro de Kalman que aparecen en [13].

Cabe comentar que podemos replantear todo el esquema cambiando algunas referencias: en vez de calcular la factorización QR de (8.4), podemos calcular la factorización QL de $(\mathbf{H}^*, \sqrt{\alpha}\mathbf{I}_n)^* = \mathbf{QL}$ (con $\mathbf{L}$, matriz triangular inferior), y factorizar $\mathbf{P}$ en vez de $\mathbf{P}^{-1}$, en sus factores raíz cuadrada como los triángulos de Cholesky $\mathbf{P} = \mathbf{P}^{1/2}\mathbf{P}^{*/2}$, siendo ahora $\mathbf{P}^{1/2}$ ó $\mathbf{P}^{-1/2}$, triangular inferior. Entonces $\mathbf{P}^{-1} = \mathbf{P}^{-*/2}\mathbf{P}^{-1/2}$, pudiendo identificar ahora $\mathbf{L}$ con la nueva $\mathbf{P}^{-1/2}$, siendo el resto de cálculos y conclusiones idénticos.

La forma de obtener $\mathbf{H}^\dagger_{\alpha,(m/q)}$ en (8.8), (8.9) ó (8.10) sigue siendo costosa, ya que cada columna de $\mathbf{H}^\dagger_\alpha$ tiene un coste de $\Theta(mn^2)$, por lo que el coste de las m columnas sería del orden de $\Theta(m^2n^2)$. Podemos optimizar el coste de la siguiente forma: podemos calcular $\mathbf{Q}_\alpha = \mathbf{H}^{\dagger *}_\alpha \mathbf{P}^{-*/2}$ en vez de $\mathbf{H}^\dagger_\alpha$ y obtener el j-ésimo vector anulador como una escalado de la j-ésima columna de $\mathbf{Q}_\alpha$, (8.6). Centraremos nuestra atención en dos formas de obtener $\mathbf{Q}_\alpha$: la primera de ellas está basada en la versión raíz cuadrada del filtro de Kalman, [22, 108], y la segunda, nuestra aportación, está basada en el filtro de información, [109].

Al emplear sendos algoritmos para resolver el problema OSIC, la diferencia entre ellos estriba exclusivamente en la forma de obtener la matriz $\mathbf{Q}_\alpha$, siendo el resto de proceso idéntico. Este es el motivo por el que nuestra atención se centrará en la paralelización de los algoritmos que obtienen esta matriz $\mathbf{Q}_\alpha$.

8.3 Versión raíz cuadrada del filtro de Kalman para OSIC

En [22] se propone calcular $\mathbf{Q}_\alpha$ a partir de la versión raíz cuadrada del filtro de Kalman (con $\lambda = 1$), realizando las siguientes iteraciones:

Entrada: $\mathbf{H} = (\mathbf{H}^*_0, \mathbf{H}^*_1, \ldots)^*$, $\mathbf{P}^{1/2}_0 = \frac{1}{\sqrt{\alpha}}\mathbf{I}_n$ y $\mathbf{Q}_{\alpha,(0)} = \mathbf{0}_{m\times n}$

Salida: Actualización de $\mathbf{Q}_\alpha = \mathbf{Q}_{\alpha,(m/q)}$

para $i = 0, \ldots, m/q - 1$ **hacer**

Calcular y aplicar $\mathbf{\Theta}_i$ unitaria tal que:

$$\mathbf{E}_i\mathbf{\Theta}_i = \begin{pmatrix} \mathbf{I}_q & \mathbf{H}_i\mathbf{P}_i^{1/2} \\ \mathbf{0} & \mathbf{P}_i^{1/2} \\ -\mathbf{\Gamma}_{i+1} & \mathbf{Q}_{\alpha,(i)} \end{pmatrix} \mathbf{\Theta}_i \;=\; \begin{pmatrix} \mathbf{R}_{e,i}^{1/2} & \mathbf{0} \\ \overline{\mathbf{K}}_{p,i} & \mathbf{P}_{i+1}^{1/2} \\ \mathbf{Z}_i & \mathbf{Q}_{\alpha,(i+1)} \end{pmatrix} = \mathbf{F}_i$$

fin para

Para evitar confusiones en la notación, cada una de las instancias en cada iteración i-ésima de las submatrices del algoritmo se denotan con el subíndice i, salvo en la matriz $\mathbf{Q}_\alpha$, en el que se muestra entre paréntesis, (i). Esta notación tiene la excepción de $\mathbf{H}_i$ que denota la fila i-ésima de $\mathbf{H}$.

Por lo tanto, ya podemos obtener los vectores anuladores a partir de $\mathbf{Q}_\alpha$ y $\mathbf{P}^{1/2}$, según (8.6).

Existen dos diferencias respecto al algoritmo empleado para resolver el problema RLS visto en el capítulo 4:

- En este algoritmo no se calcula la solución LS $\mathbf{x}_{\text{LS}}$, sólo la matriz $\mathbf{Q}_\alpha$ que junto con $\mathbf{P}^{1/2}$ permiten obtener los vectores anuladores con los que hallar la solución $\hat{\mathbf{x}}$.

- La matriz problema tiene un bloque de filas adicional $(-\mathbf{\Gamma}_{i+1}, \quad \mathbf{Q}_{\alpha,(i)})$.

Podemos emplear exactamente el mismo algoritmo que en el capítulo 4, pero extendiendo la aplicación de las rotaciones de Givens a un mayor número de filas, abarcando hasta este bloque de filas añadido (concretamente hasta la parte no nula, ya que $\mathbf{\Gamma}_{i+1}$ sólo tiene sus primeras $(i+1)q$ filas no nulas).

De manera que deberemos modificar el coste del algoritmo secuencial añadiendo a (4.4) los flops asociados a las nuevas $(i+1)q$ filas, es decir $w_{AG}(i+1)q$ flops y obviar los de la

solución LS que no es calculada de esta forma. De manera que en el cómputo de los costes (4.7), que redenominaremos $W_{\text{SRKF-sec},i}(z)$, y (4.9), que redenominaremos $W_{\text{SRKF-sec}}(z)$, habrán que modificarlos como

$$\begin{aligned} W_{\text{sec},i}(z) &= W_{\text{SRKF-sec},i}(z) + \sum_{c=1}^{n}\sum_{r=1}^{q} w_{AG}(i+1)q \\ &= qn^2 + w_{AG}\left(\frac{1}{2}q + \frac{1}{2}n + 1\right)qn + w_{AG}(i+1)q^2 n \\ &= qn^2 + w_{AG}\left(\frac{1}{2}q + \frac{1}{2}n + 1 + (i+1)q\right)qn \quad \text{flops} \end{aligned}$$

y

$$\begin{aligned} W_{\text{sec}}(z) &= W_{\text{SRKF-sec}}(z) + \sum_{i=0}^{m/q-1}\sum_{c=1}^{n}\sum_{r=1}^{q} w_{AG}(i+1)q \\ &= \left(1 + \frac{1}{2}w_{AG}\right)n^2 m + w_{AG}\left(\frac{1}{2}q + 1\right)nm + \frac{1}{2}w_{AG}nm^2 + \frac{1}{2}w_{AG}qnm \\ &\approx \left(1 + \frac{1}{2}w_{AG}\right)n^2 m + \frac{1}{2}w_{AG}nm^2 \\ &= \Theta\left(n^2 m\right) + \Theta\left(nm^2\right) \end{aligned} \tag{8.11}$$

Para realizar una comparación más clara con el algoritmo para OSIC basado en el filtro de información, que será expuesto en la sección siguiente, nos resulta conveniente expresar el coste de manera alternativa, basándonos en el modelo de llamadas. De la misma forma que obramos para el algoritmo SRKF que resolvía el problema RLS, denotaremos por $\text{w}_{\text{TRMM}}(f,c)$ el coste de realizar la multiplicación matricial y por $\text{w}_{\text{ROT}}(f)$ el de la aplicación de las rotaciones de Givens (subseccion 4.1.2) , de modo que

$$
\begin{aligned}
W_{\text{sec}}(z) &= \sum_{i=0}^{m/q-1} \left\{ \text{w}_{\text{TRMM}}(q,n) + \sum_{c=1}^{n} \left[\sum_{r=1}^{q} \text{w}_{\text{ROT}}(r-q+1) \right.\right. \\
&\quad \left.\left. + \text{w}_{\text{ROT}}(n-c+1+(i+1)q) \right] \right\} \\
&= \sum_{i=0}^{m/q-1} \left\{ \text{w}_{\text{TRMM}}(q,n) + \sum_{c=1}^{n} \left[q\text{w}_{\text{ROT}}(n-c+1+(i+1)q) \right.\right. \\
&\quad \left.\left. + \sum_{r=1}^{q} \text{w}_{\text{ROT}}(r) \right] \right\} \\
&= \sum_{i=0}^{m/q-1} \left\{ \text{w}_{\text{TRMM}}(q,n) + n\sum_{r=1}^{q} \text{w}_{\text{ROT}}(r) + q\sum_{c=1}^{n} \text{w}_{\text{ROT}}(c+(i+1)q) \right\} \\
&= \frac{m}{q}\text{w}_{\text{TRMM}}(q,n) + \frac{m}{q}n\sum_{r=1}^{q} \text{w}_{\text{ROT}}(r) \\
&\quad + q\sum_{i=0}^{m/q-1}\sum_{c=1}^{n} \text{w}_{\text{ROT}}(c+(i+1)q) \qquad (8.12)
\end{aligned}
$$

Respecto al algoritmo paralelo, [108], además de las consecuentes diferencias por la ampliación del algoritmo secuencial, lo más significativo será que la cantidad de información a enviar de un procesador al siguiente dentro de la línea segmentada (unidireccional o bidireccional) será creciente con el número de iteración.

De la misma forma, deberemos modificar el coste del algoritmo paralelo. Concretamente, para la iteración i-ésima en el procesador P_j, la actualización de (4.12), que redenominaremos $W_{\text{SRKF-}P_j,i}(z)$, es:

$$
\begin{aligned}
W_{P_j,i}(z) &= W_{\text{SRKF-}P_j,i}(z) + \sum_{c=1}^{n_j}\sum_{r=1}^{q} w_{AG}(i+1)q \\
&= (2q+w_{AG}q)nn_j - \frac{1}{2}(2q+w_{AG}q)n_j^2 \\
&\quad - \left[(2q+w_{AG}q)c_{0_j} - (2q+w_{AG}q) - w_{AG}q - \frac{1}{2}w_{AG}q^2\right] n_j \\
&\quad + w_{AG}(i+1)q^2 n_j
\end{aligned}
$$

Consecuentemente, los valores numéricos que entran en juego en el equilibrado de una línea segmentada unidireccional hay que modificarlos oportunamente, es decir, la versión de la expresión (4.14) será:

$$\begin{aligned}(2q+w_{AG}q)nn_j-\frac{1}{2}\left(2q+w_{AG}q\right)n_j^2\\ -\left[(2q+w_{AG}q)c_{0_j}-(2q+w_{AG}q)-w_{AG}q-\frac{1}{2}w_{AG}q^2-w_{AG}(i+1)q^2\right]n_j\\ \alpha_j\left[qn^2+w_{AG}\left(\frac{1}{2}q+\frac{1}{2}n+1+(i+1)q\right)qn\right]\end{aligned}$$

que para $i=m/q-1$:

$$\begin{aligned}(2q+w_{AG}q)nn_j-\frac{1}{2}\left(2q+w_{AG}q\right)n_j^2\\ -\left[(2q+w_{AG}q)c_{0_j}-(2q+w_{AG}q)-w_{AG}q-\frac{1}{2}w_{AG}q^2-w_{AG}mq\right]n_j \quad =\\ \alpha_j\left[qn^2+w_{AG}\left(\frac{1}{2}q+\frac{1}{2}n+1+m\right)qn\right]\end{aligned}$$

Donde hemos asignado un valor de $w_{AG}=6$ flops, según la referencia [87], para poder resolver la secuencia de ecuaciones de segundo grado necesaria para obtener los distintos n_j.

8.3.1. Análisis de las comunicaciones

A las comunicaciones entre cualquier par de procesadores, hay que añadir las $(i+1)$ filas de q elementos de $\mathbf{\Gamma}_{i+1}$ que van siendo actualizadas, por lo que

$$T_{C,P_j,i}(z)=\beta+\tau\left[qn+\frac{1}{2}q(q+1)+q\sum_{k=0}^{j}n_k+\right]+\tau(i+1)q$$

Para el modelo de comunicación en el que las comunicaciones pueden ser simultáneas:

$$\begin{aligned} T_{C,i}(z,p)^{(A)} &= \max_{j=0,\ldots p-2}\{T_{C,P_j,i}(z)\} \\ &= T_{C,P_{p-2},i}(z) \\ &= \beta + \tau\left[qn + \frac{1}{2}q(q+1) + q(n-n_{p-1}) + (i+1)q\right] \end{aligned}$$

Por lo que, si obviamos el tiempo de llenado o vaciado de la línea segmentada:

$$\begin{aligned} T_C^{(A)}(z,p) &= \sum_{i=0}^{m/q-1} T_{C,i}^{(A)}(z,p) \\ &= \beta\frac{m}{q} + m\tau\left[n + \frac{1}{2}(q+1) + (n-n_{p-1}) + \frac{1}{2}\cdot\frac{m}{q} + \frac{1}{2}\right] \\ &= \Theta(mn) + \Theta\left(\frac{m^2}{q}\right) \end{aligned} \tag{8.13}$$

Para el modelo de comunicación en el que las comunicaciones deben serializarse:

$$\begin{aligned} T_{C,i}^{(B)}(z,p) &= \sum_{j=0}^{p-2} T_{C,P_j,i}(z) \\ &= \sum_{j=0}^{p-2}\left\{\beta + \tau\left[qn + \frac{1}{2}q(q+1) + q\sum_{k=0}^{j} n_k + (i+1)q\right]\right\} \\ &= (p-1)\left\{\beta + \tau\left[qn + \frac{1}{2}q(q+1) + (i+1)q\right]\right\} + \tau q\sum_{j=0}^{p-2}\sum_{k=0}^{j} n_k \\ &< (p-1)\left\{\beta + \tau\left[qn + \frac{1}{2}q(q+1) + (i+1)q\right]\right\} + \tau q(p-1)n \end{aligned}$$

Para todas las iteraciones, el resultado será:

$$
\begin{aligned}
T_C^{(B)}(z,p) &< \sum_{i=0}^{m/q-1} \left\{ (p-1) \left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) + (i+1)q \right] \right\} + \tau q(p-1)n \right\} \\
&< \frac{m}{q} \left\{ (p-1) \left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) + \frac{1}{2}m + \frac{1}{2}q \right] \right\} + \tau q(p-1)n \right\} \\
&< \frac{m}{q}(p-1)\beta + m(p-1)\tau \left[n + \frac{1}{2}(q+1) + \frac{1}{2q}m + \frac{1}{2} \right] + \tau m(p-1)n \\
&= \Theta(mpn) + \Theta\left(\frac{m^2 p}{q} \right) \qquad (8.14)
\end{aligned}
$$

Escalabilidad

Ya que la paralelización no ha implicado sobrecarga aritmética, debemos comparar el tiempo del algoritmo secuencial (8.11) con el tiempo total de sobrecarga de comunicación $pT_C(z,p)$ con alguno de los modelos extremos A o B, (8.13) o (8.14).

Para el modelo de comunicación (A), podemos comprobar que $n = \Theta(p)$, $m = \Theta(p)$ y $n^2/m = \Theta(p/q)$ (si $m = \Theta(n)$, entonces $n = \Theta(p/q)$), luego generalizando $n = \Theta(p)$ y $m = \Theta(p)$

Para el modelo de comunicación (B), podemos comprobar que tanto $n = \Theta(p^2)$ como $m = \Theta(p^2)$, y $n^2/m = \Theta(p^2/q)$ (por lo que si $m = \Theta(n)$, entonces $n = \Theta(p^2/q)$. Por lo que podemos generalizar el resultado de la escalabilidad como que tanto $n = \Theta(p^2)$ como $m = \Theta(p^2)$.

8.3.2. Resultados experimentales

Los resultados que a continuación se muestran han sido obtenidos de ejecuciones realizadas en la máquina ccNUMA *Aldebarán*.

Tiempo de ejecución secuencial

La figura 8.2 muestra los resultados de la ejecución secuencial de este algoritmo.

Encontramos un comportamiento esperado a excepción de la curva para el parámetro

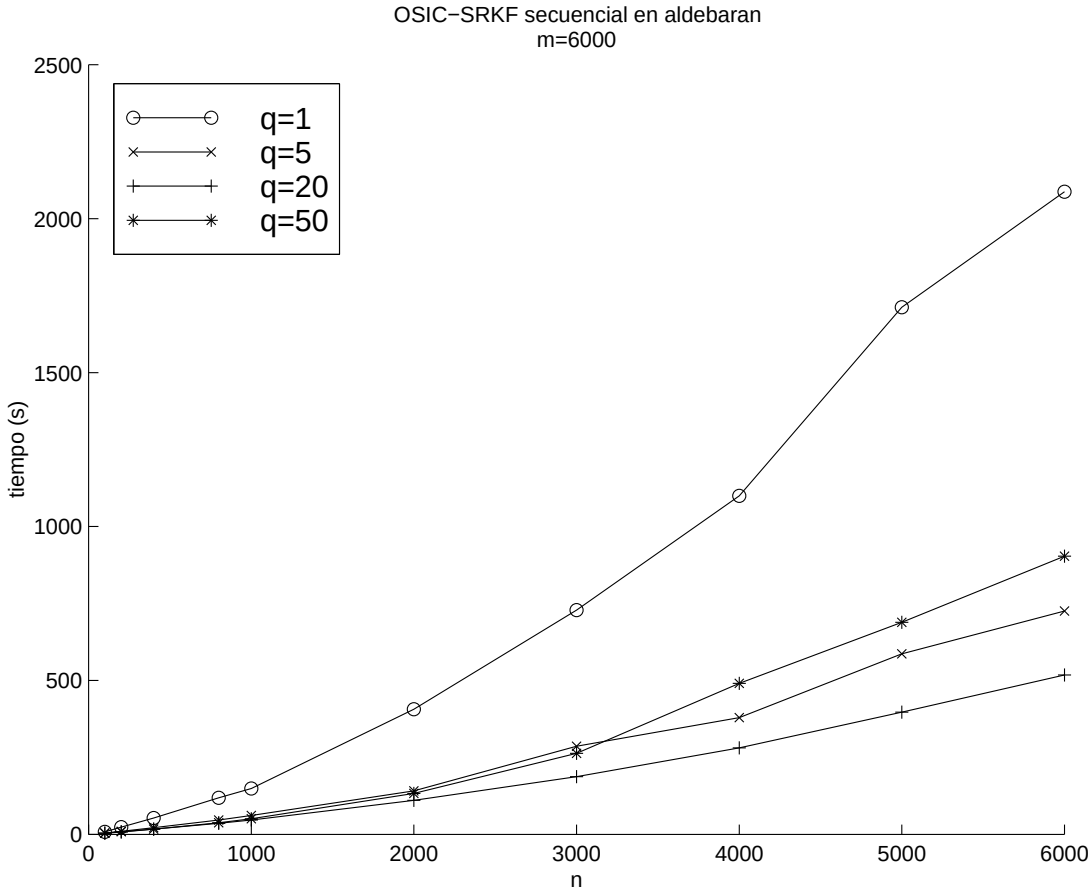

Figura 8.2: Tiempos de ejecución secuencial del algoritmo OSIC-SRKF

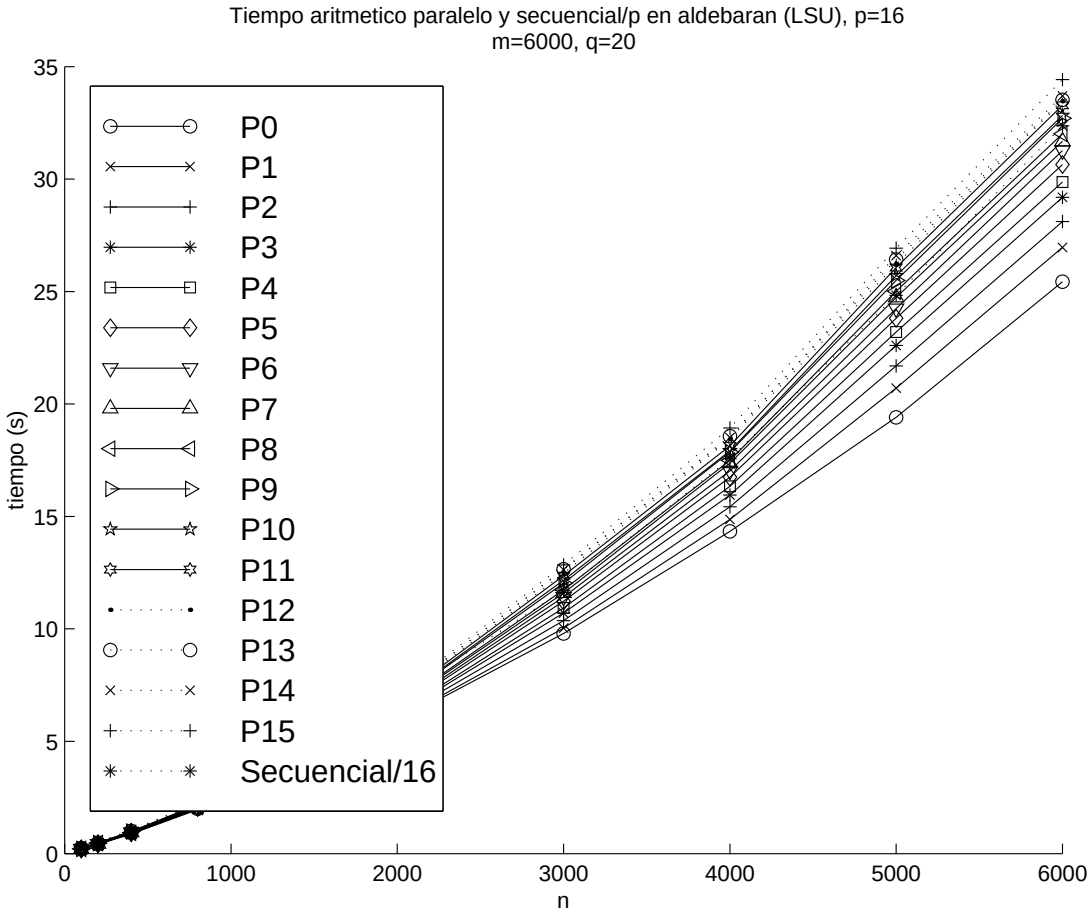

Figura 8.3: Tiempos aritméticos por procesador y secuencial dividido por el número de procesadores para la línea segmentada unidireccional con $q = 20$.

q con el valor $q = 50$, lo cual indica que existe un mínimo en el tiempo de ejecución en función del parámetro q, que ha sido superado para $q = 50$.

Equilibrado de carga

Las figuras 8.3 y 8.4 muestran los tiempos aritméticos paralelos por procesador y el tiempo secuencial dividido por el número de procesadores para la línea segmentada unidireccional y bidireccional respectivamente para $q = 20$, observando un buen equilibrado para este último y cierto desequilibrio para el primero, debido, por tanto a la diferencia que existe entre el modelo de tiempo de ejecución y la realidad.

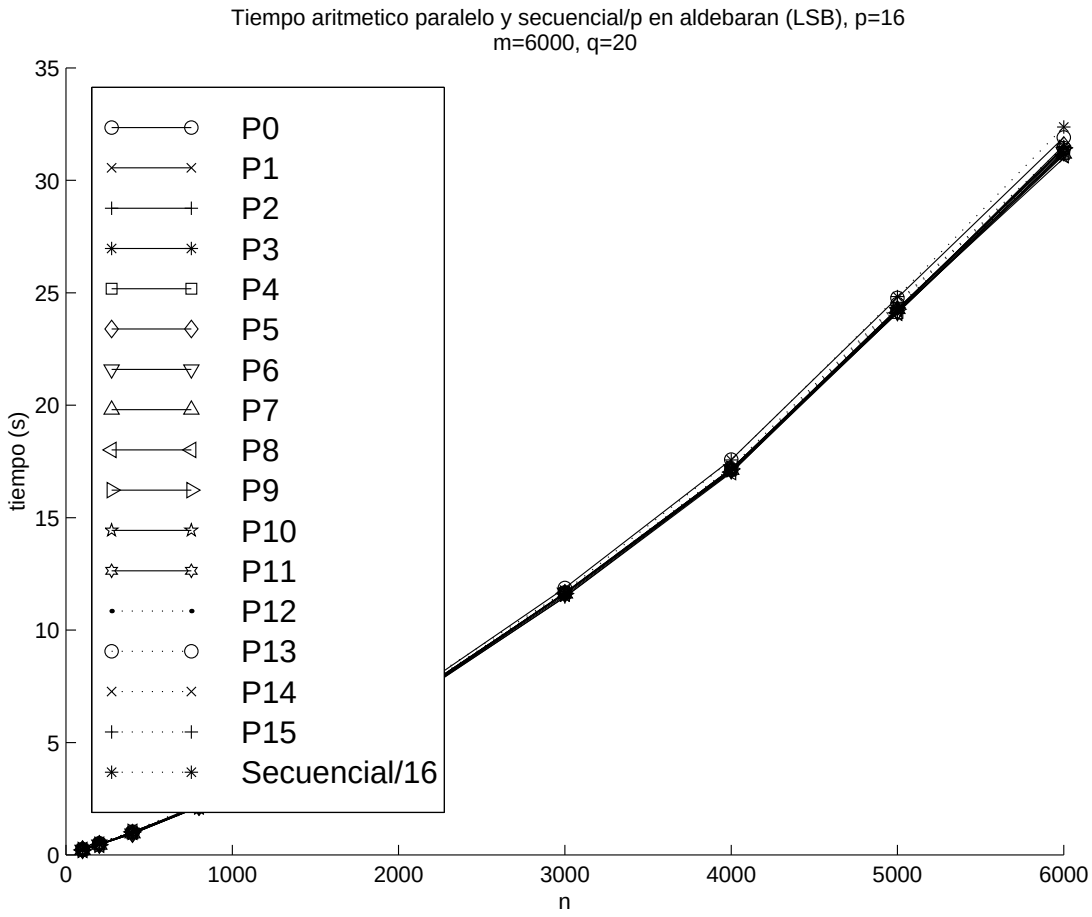

Figura 8.4: Tiempos aritméticos por procesador y secuencial dividido por el número de procesadores para la línea segmentada bidireccional con $q = 20$.

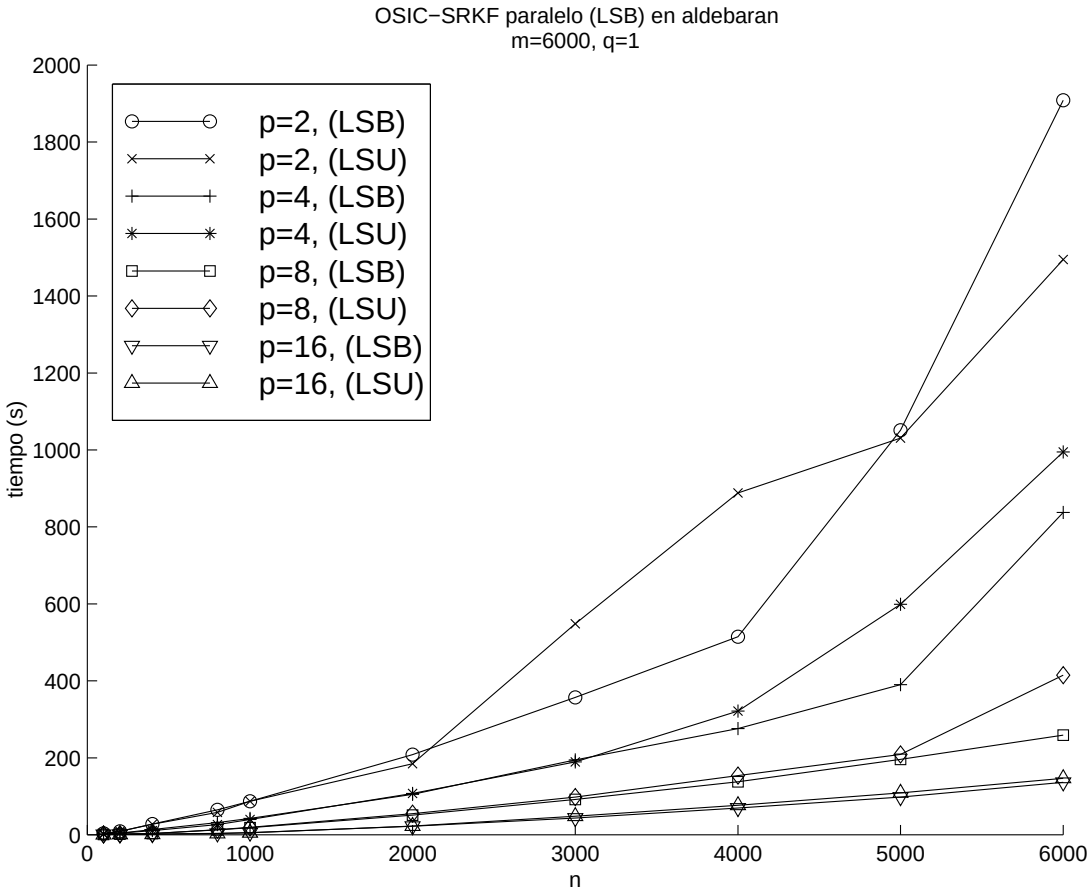

Figura 8.5: Tiempos paralelos en la línea segmentada unidireccional (LSU) y bidireccional (LSB) para $q = 1$

Tiempo de ejecución paralelo

Las figuras 8.5 y 8.6 muestran los tiempos de ejecución paralalela tanto para la versión unidireccional como la bidireccional para $q = 1$ y $q = 20$ respectivamente, apreciando cómo para $q = 1$ el caso bidireccional tiene una sutil mejor respuesta, que se invierte para valores superiores.

Eficiencia

La figura 8.7 muestra la eficiencia en la paralelización para $q = 20$, donde se observa que la línea segmentada unidireccional tiene para este valor del parámetro q un notable superior eficiencia que el caso bidireccional.

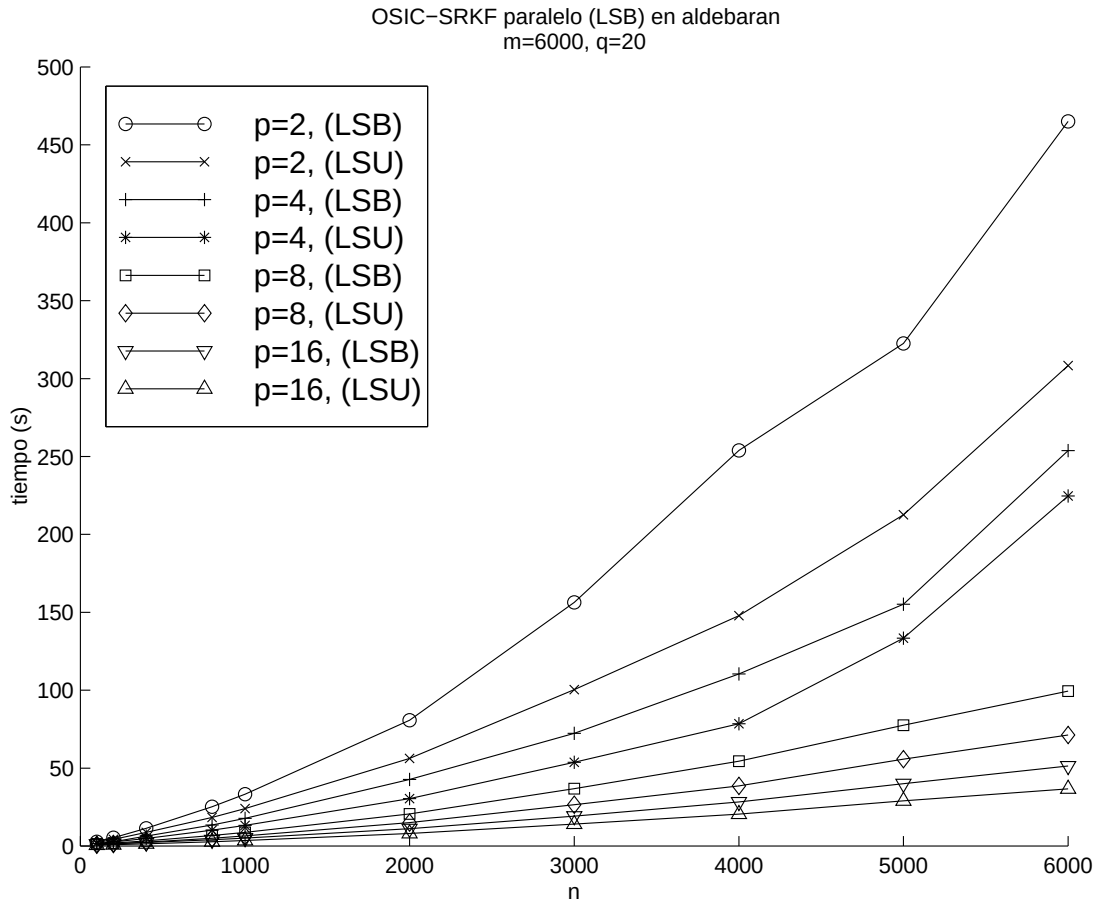

Figura 8.6: Tiempos paralelos en la línea segmentada unidireccional (LSU) y bidireccional (LSB) para $q = 20$

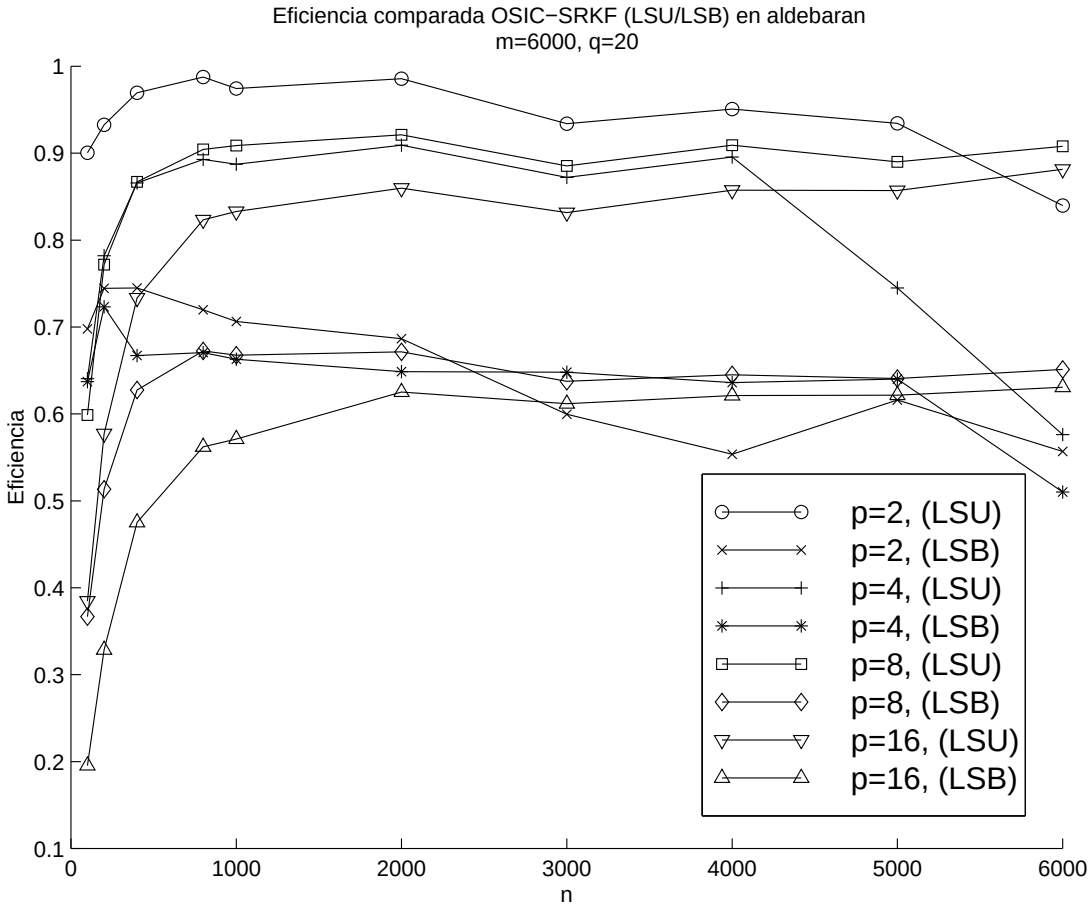

Figura 8.7: Eficiencia en la línea segmentada unidireccional (LSU) y bidireccional (LSB) para $q = 20$

8.3.3. Conclusión

Obtenemos resultados de comportamiento similares a los del algoritmo SRKF-RLS expuestos en el capítulo 4, es decir un mejor comportamiento de la versión bidireccional para $q = 1$ que se invierte para valores superiores de dicho parámetro, por los mismo motivos que los que fueron expuestos en dicho capítulo.

8.4 Modificación de la versión raíz cuadrada del filtro de información para OSIC

A continuación mostramos una variación del algoritmo *raíz cuadrada extendido del filtro de información* (véase el capítulo 5), para resolver parte del problema de decodificación V-BLAST-OSIC, [109]. En [110] se muestra adicionalmente una comparación con el método basado en SRKF.

Una solución al problema BLAST-OSIC consiste en resolver de manera óptima la secuencia (8.10) para resolver la decodificación con cancelación de la interferencia sucesiva con ordenación. El algoritmo *raíz cuadrada del filtro de Kalman o del filtro de covarianza* es utilizado en [22]. Nuestra aportación consiste en emplear el algoritmo *raíz cuadrada extendido del filtro de información* para conseguir el resultado. Como podremos comprobar posteriormente, cierta modificación de este algoritmo nos permitirá obtener un ahorro de coste respecto del método propuesto en [22].

Luego nuestro objetivo ahora es conseguir de manera iterativa la secuencia $\mathbf{Q}_{\alpha,(i)} = \mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{P}_i^{-*/2}$ en forma de algoritmo raíz cuadrada. Para ello planteemos la igualdad (5.1) ampliada

$$\mathbf{V}_i\mathbf{\Theta}_i = \begin{pmatrix} \lambda^{1/2}\mathbf{P}_i^{-*/2} & \lambda^{1/2}\mathbf{H}_i^* \\ \hat{\mathbf{x}}_i^*\mathbf{P}_i^{-*/2} & \mathbf{y}_i^* \\ \lambda^{-1/2}\mathbf{P}_i^{1/2} & \mathbf{0} \\ \mathbf{A} & \mathbf{B} \end{pmatrix} \mathbf{\Theta}_i \;=\; \begin{pmatrix} \mathbf{P}_{i+1}^{-*/2} & \mathbf{0} \\ \hat{\mathbf{x}}_{i+1}^*\mathbf{P}_{i+1}^{-*/2} & \mathbf{e}_i^*\mathbf{R}_{e,i}^{-*/2} \\ \mathbf{P}_{i+1}^{1/2} & -\overline{\mathbf{K}}_{p,i} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} = \mathbf{W}_i$$

forzando a que $\mathbf{A} = \mathbf{Q}_{\alpha,(i)} = \mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{P}_i^{-*/2}$ y $\mathbf{C} = \mathbf{Q}_{\alpha,(i+1)} = \mathbf{H}_{\alpha,(i+1)}^{\dagger *}\mathbf{P}_{i+1}^{-*/2}$. A continuación deduciremos cuáles deben ser los valores de las matrices $\mathbf{B}$ y $\mathbf{D}$ para que se satisfaga la nueva igualdad. Antes de deducir estos valores, ya que no vamos a estar intersados en las filas $\begin{pmatrix} \hat{\mathbf{x}}_i^*\mathbf{P}_i^{-*/2} & \mathbf{y}_i^* \end{pmatrix}$ de $\mathbf{V}_i$ y $\begin{pmatrix} \hat{\mathbf{x}}_{i+1}^*\mathbf{P}_{i+1}^{-*/2} & \mathbf{e}_i^*\mathbf{R}_{e,i}^{-*/2} \end{pmatrix}$ de $\mathbf{W}_i$, las omitiremos de los cálculos.

La deducción de $\mathbf{B}$ y $\mathbf{D}$ la haremos a partir de la igualdad:

$$\mathbf{V}_i\mathbf{V}_i^* = \mathbf{W}_i\mathbf{W}_i^*$$

$$\begin{pmatrix} \lambda^{1/2}\mathbf{P}_i^{-*/2} & \lambda^{1/2}\mathbf{H}_i^* \\ \lambda^{-1/2}\mathbf{P}_i^{1/2} & \mathbf{0} \\ \mathbf{A} & \mathbf{B} \end{pmatrix} \begin{pmatrix} \lambda^{1/2}\mathbf{P}_i^{-1/2} & \lambda^{-1/2}\mathbf{P}_i^{*/2} & \mathbf{A}^* \\ \lambda^{1/2}\mathbf{H}_i & \mathbf{0} & \mathbf{B}^* \end{pmatrix} =$$

$$= \begin{pmatrix} \mathbf{P}_{i+1}^{-*/2} & \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} & -\overline{\mathbf{K}}_{p,i} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{P}_{i+1}^{-1/2} & \mathbf{P}_{i+1}^{*/2} & \mathbf{C}^* \\ \mathbf{0} & -\overline{\mathbf{K}}_{p,i}^* & \mathbf{D}^* \end{pmatrix}$$

La submatriz $\mathbf{B}$ podremos deducirla cómodamente a partir de la igualación de los productos internos de la primera fila y de la tercera columna de submatrices por una parte de $\mathbf{V}_i$ y $\mathbf{V}_i^*$ y por la otra de $\mathbf{W}_i$ y de $\mathbf{W}_i^*$:

$$\begin{aligned}
\lambda^{1/2}\mathbf{P}_i^{-*/2}\mathbf{A}^* + \lambda^{1/2}\mathbf{H}_i^*\mathbf{B}^* &= \mathbf{P}_{i+1}^{-*/2}\mathbf{C}^* \\
\lambda^{1/2}\mathbf{P}_i^{-*/2}\left(\mathbf{P}_i^{-1/2}\mathbf{H}_{\alpha,(i)}^\dagger\right) + \left(\lambda^{1/2}\mathbf{H}_i^*\mathbf{B}^*\right) &= \mathbf{P}_{i+1}^{-*/2}\left(\mathbf{P}_{i+1}^{-1/2}\mathbf{H}_{\alpha,(i+1)}^\dagger\right) \\
\lambda^{1/2}\mathbf{P}_i^{-1}\mathbf{H}_{\alpha,(i)}^\dagger + \lambda^{1/2}\mathbf{H}_i^*\mathbf{B}^* &= \mathbf{P}_{i+1}^{-1}\mathbf{H}_{\alpha,(i+1)}^\dagger
\end{aligned}$$

A continuación, utilizando (5.5) para $\mathbf{P}_{i+1}^{-1}$ y (8.10) para $\mathbf{H}_{\alpha,(i+1)}^\dagger$:

$$\begin{aligned}
\lambda^{1/2}\mathbf{P}_i^{-1}\mathbf{H}_{\alpha,(i)}^\dagger + \lambda^{1/2}\mathbf{H}_i^*\mathbf{B}^* &= \lambda\left(\mathbf{P}_i^{-1} + \mathbf{H}_i^*\mathbf{H}_i\right)\cdot \\
&\quad \cdot\left\{\lambda^{-1/2}\mathbf{H}_{\alpha,(i)}^\dagger + \overline{\mathbf{K}}_{p,i}\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{\Gamma}_{i+1}^* - \mathbf{H}_i\mathbf{H}_{\alpha,(i)}^\dagger\right)\right\}
\end{aligned}$$

Evaluando y sustituyendo $\overline{\mathbf{K}}_{p,i}$ por su expresión (5.4):

$$\begin{aligned}
\lambda^{1/2}\mathbf{P}_i^{-1}\mathbf{H}_{\alpha,(i)}^\dagger + \lambda^{1/2}\mathbf{H}_i^*\mathbf{B}^* &= \lambda^{1/2}\mathbf{P}_i^{-1}\mathbf{H}_{\alpha,(i)}^\dagger \\
&\quad +\lambda^{1/2}\mathbf{P}_i^{-1}\mathbf{P}_i\mathbf{H}_i^*\mathbf{R}_{e,i}^{-*/2}\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{E}_{i+1}^* - \mathbf{H}_i\mathbf{H}_{\alpha,(i)}^\dagger\right) \\
&\quad +\lambda^{1/2}\mathbf{H}_i^*\mathbf{H}_i\mathbf{H}_{\alpha,(i)}^\dagger \\
&\quad +\lambda^{1/2}\mathbf{H}_i^*\mathbf{H}_i\mathbf{P}_i\mathbf{H}_i^*\mathbf{R}_{e,i}^{-*/2}\mathbf{R}_{e,i}^{-1/2}\left(\mathbf{\Gamma}_{i+1}^* - \mathbf{H}_i\mathbf{H}_{\alpha,(i)}^\dagger\right)
\end{aligned}$$

por lo que

$$
\begin{aligned}
\mathbf{H}_i^*\mathbf{B}^* &= \mathbf{H}_i^*\mathbf{R}_{e,i}^{-1}\left(\mathbf{\Gamma}_{i+1}^* - \mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger}\right) \\
&\quad +\mathbf{H}_i^*\mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger} \\
&\quad +\mathbf{H}_i^*\mathbf{H}_i\mathbf{P}_i\mathbf{H}_i^*\mathbf{R}_{e,i}^{-1}\left(\mathbf{\Gamma}_{i+1}^* - \mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger}\right) \\
&= \mathbf{H}_i^*\mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger} + \mathbf{H}_i^*\left(\mathbf{I} + \mathbf{H}_i\mathbf{P}_i\mathbf{H}_i^*\right)\mathbf{R}_{e,i}^{-1}\left(\mathbf{\Gamma}_{i+1}^* - \mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger}\right) \\
&= \mathbf{H}_i^*\mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger} + \mathbf{H}_i^*\mathbf{R}_{e,i}\mathbf{R}_{e,i}^{-1}\left(\mathbf{\Gamma}_{i+1}^* - \mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger}\right) \quad \text{—según (5.3)—} \\
&= \mathbf{H}_i^*\mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger} + \mathbf{H}_i^*\left(\mathbf{\Gamma}_{i+1}^* - \mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger}\right) \\
&= \mathbf{H}_i^*\mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger} + \mathbf{H}_i^*\mathbf{\Gamma}_{i+1}^* - \mathbf{H}_i^*\mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger} \\
&= \mathbf{H}_i^*\mathbf{\Gamma}_{i+1}^*
\end{aligned}
$$

por tanto, una solución para $\mathbf{B}$ es $\mathbf{B} = \mathbf{\Gamma}_{i+1}$.

De forma similar podemos deducir $\mathbf{D}$, esta vez, a partir de la igualación de los productos internos de la fila tercera y columna segunda de, por una parte $\mathbf{V}_i$ y $\mathbf{V}_i^*$, y por la otra de $\mathbf{W}_i$ y $\mathbf{W}_i^*$:

$$
\begin{aligned}
\mathbf{A}\lambda^{-1/2}\mathbf{P}_i^{*/2} &= \mathbf{C}\mathbf{P}_{i+1}^{*/2} - \mathbf{D}\overline{\mathbf{K}}_{p,i}^* \\
\left(\mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{P}_i^{-*/2}\right)\lambda^{-1/2}\mathbf{P}_i^{*/2} &= \left(\mathbf{H}_{\alpha,(i+1)}^{\dagger *}\mathbf{P}_{i+1}^{-*/2}\right)\mathbf{P}_{i+1}^{*/2} - \mathbf{D}\overline{\mathbf{K}}_{p,i}^* \\
\lambda^{-1/2}\mathbf{H}_{\alpha,(i)}^{\dagger *} &= \mathbf{H}_{\alpha,(i+1)}^{\dagger *} - \mathbf{D}\overline{\mathbf{K}}_{p,i}^* \\
&= \lambda^{-1/2}\mathbf{H}_{\alpha,(i)}^{\dagger *} + \left(\mathbf{\Gamma}_{i+1} - \mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{H}_i^*\right)\mathbf{R}_{e,i}^{-*/2}\overline{\mathbf{K}}_{p,i}^* \\
&\quad -\mathbf{D}\overline{\mathbf{K}}_{p,i}^* \\
\mathbf{D}\overline{\mathbf{K}}_{p,i}^* &= \left(\mathbf{\Gamma}_{i+1} - \mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{H}_i^*\right)\mathbf{R}_{e,i}^{-*/2}\overline{\mathbf{K}}_{p,i}^*
\end{aligned}
$$

Luego una solución para $\mathbf{D}$ es $\mathbf{D} = \left(\mathbf{\Gamma}_{i+1} - \mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{H}_i^*\right)\mathbf{R}_{e,i}^{-*/2}$.

Hemos deducido ya los dos valores $\mathbf{B}$ y $\mathbf{D}$ necesarios para aprovechar la iteración de la versión cuadrada y extendida del filtro de información, pero deberíamos asegurarnos

de que todo el algoritmo raíz cuadrada es coherente, es decir verificar que efectivamente $\mathbf{V}_i\mathbf{V}_i^* = \mathbf{W}_i\mathbf{W}_i^*$. Suponiendo correcta la expresión (5.1), sólo nos restaría verificar que efectivamente:

$$\mathbf{A}\mathbf{A}^* + \mathbf{B}\mathbf{B}^* = \mathbf{C}\mathbf{C}^* + \mathbf{D}\mathbf{D}^*$$

es decir

$$\begin{aligned}
\mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{P}_i^{-*/2}\mathbf{P}^{-1/2}\mathbf{H}_{\alpha,(i)}^{\dagger} + \mathbf{\Gamma}_{i+1}\mathbf{\Gamma}_{i+1}^* &= \mathbf{H}_{\alpha,(i+1)}^{\dagger *}\mathbf{P}_{i+1}^{-*/2}\mathbf{P}_{i+1}^{-1/2}\mathbf{H}_{\alpha,(i+1)}^{\dagger} \\
&\quad + \left(\mathbf{\Gamma}_{i+1} - \mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{H}_i\right)\mathbf{R}_{e,i}^{-*/2}\mathbf{R}_{e,i}^{-1/2} \cdot \\
&\quad \cdot \left(\mathbf{\Gamma}_{i+1}^* - \mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger *}\right) \\
\mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{P}_i^{-1}\mathbf{H}_{\alpha,(i)}^{\dagger} + \mathbf{\Gamma}_{i+1}\mathbf{\Gamma}_{i+1}^* &= \mathbf{H}_{\alpha,(i+1)}^{\dagger *}\mathbf{P}_{i+1}^{-1}\mathbf{H}_{\alpha,(i+1)}^{\dagger} \\
&\quad + \left(\mathbf{\Gamma}_{i+1} - \mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{H}_i\right)\mathbf{R}_{e,i}^{-1} \cdot \\
&\quad \cdot \left(\mathbf{\Gamma}_{i+1}^* - \mathbf{H}_i\mathbf{H}_{\alpha,(i)}^{\dagger *}\right)
\end{aligned} \tag{8.15}$$

Abordaremos de momento el primer sumando de la parte derecha de la última igualdad. Utilizando de nuevo (5.5) para $\mathbf{P}_{i+1}^{-1}$ y (8.10) para $\mathbf{H}_{\alpha,(i+1)}^{\dagger}$:

$$
\begin{aligned}
\mathbf{H}^{\dagger *}_{\alpha,(i+1)}\mathbf{P}^{-1}_{i+1}\mathbf{H}^{\dagger}_{\alpha,(i+1)} &= \left\{\lambda^{-1/2}\mathbf{H}^{\dagger *}_{\alpha,(i)}+\right. \\
&\quad \left.\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-*/2}_{e,i}\lambda^{-1/2}\mathbf{R}^{-1/2}_{e,i}\mathbf{H}_{i}\mathbf{P}^{*}_{i}\right\}\cdot \\
&\quad \cdot\lambda\left(\mathbf{P}^{-1}_{i}+\mathbf{H}^{*}_{i}\mathbf{H}_{i}\right) \\
&\quad \cdot\left\{\lambda^{-1/2}\mathbf{H}^{\dagger}_{\alpha,(i)}+\right. \\
&\quad \left.\lambda^{-1/2}\mathbf{P}_{i}\mathbf{H}^{*}_{i}\mathbf{R}^{-*/2}_{e,i}\mathbf{R}^{-1/2}_{e,i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right)\right\} \\
&= \left\{\mathbf{H}^{\dagger *}_{\alpha,(i)}+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\mathbf{H}_{i}\mathbf{P}^{*}_{i}\right\}\cdot \\
&\quad \cdot\left(\mathbf{P}^{-1}_{i}+\mathbf{H}^{*}_{i}\mathbf{H}_{i}\right) \\
&\quad \cdot\left\{\mathbf{H}^{\dagger}_{\alpha,(i)}+\mathbf{P}_{i}\mathbf{H}^{*}_{i}\mathbf{R}^{-1}_{e,i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right)\right\}
\end{aligned}
$$

Distribuyendo los sumandos:

$$
\begin{aligned}
\mathbf{H}^{\dagger *}_{\alpha,(i+1)}\mathbf{P}^{-1}_{i+1}\mathbf{H}^{\dagger}_{\alpha,(i+1)} &= \mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{P}^{-1}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\
&\quad +\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{P}^{-1}_{i}\mathbf{P}_{i}\mathbf{H}^{*}\mathbf{R}^{-1}_{e,i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right) \\
&\quad +\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\
&\quad +\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\mathbf{H}_{i}\mathbf{P}_{i}\mathbf{H}^{*}_{i}\mathbf{R}^{-1}_{e,i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right) \\
&\quad +\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\mathbf{H}_{i}\mathbf{P}^{*}_{i}\mathbf{P}^{-1}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\
&\quad +\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\mathbf{H}_{i}\mathbf{P}^{*}_{i}\mathbf{P}^{-1}_{i}\mathbf{P}_{i}\mathbf{H}^{*}_{i}\mathbf{R}^{-1}_{e,i}\cdot \\
&\quad \cdot\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right) \\
&\quad +\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\mathbf{H}_{i}\mathbf{P}^{*}_{i}\mathbf{H}^{*}_{i}\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\
&\quad +\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\mathbf{H}_{i}\mathbf{P}^{*}_{i}\mathbf{H}^{*}_{i}\mathbf{H}_{i}\mathbf{P}_{i}\mathbf{H}^{*}_{i}\mathbf{R}^{-1}_{e,i}\cdot \\
&\quad \cdot\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right)
\end{aligned}
$$

Simplificando y obteniendo factores comunes:

$$
\begin{aligned}
\mathbf{H}^{\dagger *}_{\alpha,(i+1)}\mathbf{P}^{-1}_{i+1}\mathbf{H}^{\dagger}_{\alpha,(i+1)} &= \mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{P}^{-1}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\
&+\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\left(\mathbf{I}_q+\mathbf{H}_i\mathbf{P}_i\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_i\mathbf{H}^{\dagger}_{\alpha,(i)}\right) \\
&+\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\mathbf{H}_i\mathbf{H}^{\dagger}_{\alpha,(i)} \\
&+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\left(\mathbf{I}_q+\mathbf{H}_i\mathbf{P}_i\mathbf{H}^{*}_{i}\right)\mathbf{H}_i\mathbf{H}^{\dagger}_{\alpha,(i)} \\
&+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\mathbf{H}_i\mathbf{P}_i\mathbf{H}^{*}_{i}\mathbf{R}^{-1}_{e,i}\cdot \\
&\cdot\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_i\mathbf{H}^{\dagger}_{\alpha,(i)}\right) \\
&+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\mathbf{H}_i\mathbf{P}_i\mathbf{H}^{*}_{i}\mathbf{H}_i\mathbf{P}_i\mathbf{H}^{*}_{i}\mathbf{R}^{-1}_{e,i}\cdot \\
&\cdot\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_i\mathbf{H}^{\dagger}_{\alpha,(i)}\right)
\end{aligned}
$$

Utilizando (5.3) y obteniendo de nuevo más factores comunes:

$$
\begin{aligned}
\mathbf{H}^{*}_{\alpha,(i+1)}\mathbf{P}^{-1}_{i+1}\mathbf{H}^{\dagger}_{\alpha,(i+1)} &= \mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{P}^{-1}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\
&+\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_i\mathbf{H}^{\dagger}_{\alpha,(i)}\right) \\
&+\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\mathbf{H}_i\mathbf{H}^{\dagger}_{\alpha,(i)} \\
&+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{H}_i\mathbf{H}^{\dagger}_{\alpha,(i)} \\
&+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\mathbf{H}_i\mathbf{P}_i\mathbf{H}^{*}_{i}\left(\mathbf{I}_q+\mathbf{H}_i\mathbf{P}\mathbf{H}^{*}_{i}\right)\cdot \\
&\cdot\mathbf{R}^{-1}_{e,i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_i\mathbf{H}^{\dagger}_{\alpha,(i)}\right)
\end{aligned}
$$

y del mismo modo

$$\begin{aligned}\mathbf{H}^{*}_{\alpha,(i+1)}\mathbf{P}^{-1}_{i+1}\mathbf{H}^{\dagger}_{\alpha,(i+1)} &= \mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{P}^{-1}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\ &+\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right) \\ &+\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\ &+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\ &+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\mathbf{H}_{i}\mathbf{P}_{i}\mathbf{H}^{*}_{i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right)\end{aligned}$$

Ahora, si reescribimos (8.15):

$$\begin{aligned}\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{P}^{-1}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}+\mathbf{\Gamma}_{i+1}\mathbf{\Gamma}^{*}_{i+1} &= \mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{P}^{-1}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\ &+\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right) \\ &+\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\ &\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\ &+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\mathbf{H}_{i}\mathbf{P}_{i}\mathbf{H}^{*}_{i}\cdot \\ &\cdot\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right) \\ &+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right)\end{aligned}$$

y simplificamos algunas expresiones:

$$\begin{aligned}\mathbf{\Gamma}_{i+1}\mathbf{\Gamma}^{*}_{i+1} &= \mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\mathbf{\Gamma}^{*}_{i+1} \\ &+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right) \\ &+\left(\mathbf{\Gamma}_{i+1}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\right)\mathbf{R}^{-1}_{e,i}\left(\mathbf{\Gamma}^{*}_{i+1}-\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)}\right) \\ &= \mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\mathbf{\Gamma}^{*}_{i+1}+\mathbf{\Gamma}_{i+1}\mathbf{H}\mathbf{H}^{\dagger}_{\alpha,(i)}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\mathbf{H}_{i}\mathbf{H}^{\dagger}_{\alpha,(i)} \\ &+\mathbf{\Gamma}_{i+1}\mathbf{\Gamma}^{*}_{i+1}-\mathbf{\Gamma}_{i+1}\mathbf{H}\mathbf{H}^{\dagger}_{\alpha,(i)}-\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\mathbf{\Gamma}^{*}_{i+1}+\mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}^{*}_{i}\mathbf{H}_{i}\mathbf{H}_{\alpha,(i)}\dagger\end{aligned}$$

con lo que verificamos la validez de la modificación propuesta.

El algoritmo puede expresarse como se muestra a continuación:

Entrada: $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$, $\mathbf{P}_0^{1/2} = \frac{1}{\sqrt{\alpha}}\mathbf{I}$, $\mathbf{H}_{\alpha,(0)}^{\dagger} = \mathbf{0}$

Salida: $\mathbf{P}^{1/2} = \mathbf{P}_{m/q}^{1/2}$ and $\mathbf{Q}_\alpha = \mathbf{H}_{\alpha,(m/q)}^{\dagger *}\mathbf{P}_{m/q}^{-*/2}$

para $i = 0, \ldots, m/q - 1$ **hacer**

$$
\begin{aligned}
\mathbf{V}_i\mathbf{\Theta}_i &= \begin{pmatrix} \mathbf{P}_i^{-*/2} & \mathbf{H}_i^* \\ \mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{P}_i^{-*/2} & \mathbf{\Gamma}_{i+1} \\ \mathbf{P}_i^{1/2} & \mathbf{0} \end{pmatrix} \mathbf{\Theta}_i \\
&= \begin{pmatrix} \mathbf{P}_{i+1}^{-*/2} & \mathbf{0} \\ \mathbf{H}_{\alpha,(i+1)}^{\dagger *}\mathbf{P}_{i+1}^{-*/2} & \left(\mathbf{\Gamma}_{i+1} - \mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{H}_i^*\right)\mathbf{R}_{e,i}^{-*/2} \\ \mathbf{P}_{i+1}^{1/2} & -\overline{\mathbf{K}}_{p,i} \end{pmatrix} = \mathbf{W}_i
\end{aligned}
\tag{8.16}
$$

fin para

Podemos verificar que $\mathbf{V}_i\mathbf{V}_i^* = \mathbf{W}_i\mathbf{W}_i^*$ de manera completa, como en (5.1). La manera en la que las entradas de las matrices han sido organizadas no altera el algoritmo paralelo propuesto (o secuencial). La diferencia radica en que la $(n+q)$-ésima fila $(\hat{\mathbf{x}}_i^*\mathbf{P}_i^{-*/2} \;\; \mathbf{y}_i^*)$ de $\mathbf{E}_i$ en (5.1) ha sido cambiada por la submatriz $(\mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{P}_i^{-*/2} \;\; \mathbf{\Gamma}_{i+1})$ de dimensiones $m \times (n+q)$ de $\mathbf{V}_i$ en (8.16). Así pues, las rotaciones de Givens aplicadas a los pares de vectores de dimensión $n+2$ deben ahora ser aplicadas a pares de vectores de dimensión $n+2+m$, por lo que el coste total del *algoritmo paralelo raíz cuadrada extendido del filtro de información* debería incrementarse aproximadamente en $6m^2n/p$ flops para la versión del sistema paralelo homogéneo.

Existe un refinamiento adicional que hace que el método definitivo sea más rápido que

el propuesto en [22]. Podemos observar que no es necesaria la propagación a lo largo de las iteraciones de los datos de la última fila, ya que no son necesarios para conseguir la submatriz resultado $\mathbf{H}^{\dagger *}_{\alpha,(i+1)}\mathbf{P}^{-*/2}_{i+1}$. En el **Algoritmo** 7 se muestra la versión definitiva.

Algoritmo 7 SRIF-OSIC

Entrada: $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots)^*$, $\mathbf{P}_0^{-*/2} = \sqrt{\alpha}\mathbf{I}$, $\mathbf{Q}_{\alpha,(0)} = \mathbf{0}$
Salida: $\mathbf{P}^{-*/2} = \mathbf{P}^{-*/2}_{m/q}$ y $\mathbf{Q}_\alpha = \mathbf{Q}_{\alpha,(m/q)}$
para $i = 0, \ldots, m/q - 1$ **hacer**
Calcular y aplicar $\mathbf{\Theta}_i$ unitaria tal que:

$$\begin{aligned} \mathbf{V}_i\mathbf{\Theta}_i &= \begin{pmatrix} \mathbf{P}_i^{-*/2} & \mathbf{H}_i^* \\ \mathbf{Q}_{\alpha,(i)} & \mathbf{\Gamma}_{i+1} \end{pmatrix}\mathbf{\Theta}_i \\ &= \begin{pmatrix} \mathbf{P}_{i+1}^{-*/2} & \mathbf{0} \\ \mathbf{Q}_{\alpha,(i+1)} & \left(\mathbf{\Gamma}_{i+1} - \mathbf{H}^{\dagger *}_{\alpha,(i)}\mathbf{H}_i^*\right)\mathbf{R}_{e,i}^{-*/2} \end{pmatrix} = \mathbf{W}_i \end{aligned} \tag{8.17}$$

fin para

8.4.1. Estimación de la componente con mayor relación señal-ruido

Cada vez que se decodifica una componente, es necesario conocer cuál es el índice del menor valor diagonal de la matriz $\mathbf{P}$ proveniente de la deflación de la matriz de canal ampliada. En [22] dicho valor se extrae del índice de la fila de $\mathbf{P}^{1/2}$ (o alternativamente del índice de la columna de $\mathbf{P}^{*/2}$) que posee una menor norma euclídea. Si dicho índice no es el último de la matriz que se está tratando deberemos realizar la reordenación comentada en el subapartado 8.2.

El método propuesto para resolver el problema OSIC obtiene $\mathbf{P}^{-*/2}$ y no $\mathbf{P}^{1/2}$ o $\mathbf{P}^{*/2}$. La solución más inmediata, pero que aporta mayor coste, es conseguir $\mathbf{P}^{*/2} = \left(\mathbf{P}^{-*/2}\right)^{-1}$. Este coste es aproximadamente $n^3/3$, pero nuestro objetivo se centra en intentar disminuirlo.

La alternativa que planteamos consiste en determinar una aproximación (un intervalo) a la norma euclídea de cada columna de $\mathbf{P}^{*/2}$ a partir de $\mathbf{P}^{-*/2}$ sin llegar a invertirla,

si es posible, al menos completamente. Si conseguimos dicho intervalo de norma euclídea para cada columna de $\mathbf{P}^{*/2}$, ciertas columnas serán descartadas como candidatas a tener la menor norma; para el resto será necesario proceder a la inversión y así lograr determinar definitivamente su 2-norma, pero ahora en la inversión intervendrán un menor número de columnas, con lo que el coste será inferior.

La matriz $\mathbf{P}^{-*/2}$ ha mantenido estructura de matriz triangular a lo largo de todas las iteraciones del algoritmo derivado del filtro de información. Una matriz triangular inferior podemos definirla recursivamente de la siguiente forma:

$$\mathbf{L}_i = \begin{pmatrix} \alpha_i & \mathbf{0} \\ \mathbf{a}_i & \mathbf{L}_{i-1} \end{pmatrix}$$

con $\mathbf{L}_i \in \mathbb{R}^{i\times i}$, $\alpha_i \in \mathbb{R}$, $\mathbf{a}_i \in \mathbb{R}^{i-1}$ y $\mathbf{L}_{i-1} \in \mathbb{R}^{(i-1)\times(i-1)}$ también triangular inferior, por lo que si $\mathbf{P}^{-*/2} \equiv \mathbf{L}_n$:

$$\begin{aligned} \mathbf{P}^{*/2} &= \left(\mathbf{P}^{-*/2}\right)^{-1} \\ &= (\mathbf{L}_n)^{-1} \\ &= \begin{pmatrix} \alpha_n & \mathbf{0} \\ \mathbf{a}_n & \mathbf{L}_{n-1} \end{pmatrix}^{-1} \\ &= \begin{pmatrix} \alpha_n^{-1} & \mathbf{0} \\ -\mathbf{L}_{n-1}^{-1}\mathbf{a}_n\alpha_n^{-1} & \mathbf{L}_{n-1}^{-1} \end{pmatrix} \end{aligned}$$

De esta forma, la columna j-ésima de $\mathbf{P}^{*/2}$, con $j = 1, \ldots, n$, puede hallarse como la primera columna de la matriz $\mathbf{L}_{n-j+1}$:

$$\mathbf{L}_{n-j+1}^{-1} = \begin{pmatrix} \alpha_{n-j+1}^{-1} & \mathbf{0} \\ -\mathbf{L}_{n-j}^{-1}\mathbf{a}_{n-j+1}\alpha_{n-j+1}^{-1} & \mathbf{L}_{n-j}^{-1} \end{pmatrix}^{-1}$$

cuya norma euclídea, r_j, al cuadrado es

$$r_j^2 = \left|\left| \begin{array}{c} \alpha_{n-j+1}^{-1} \\ -\mathbf{L}_{n-j}^{-1}\mathbf{a}_{n-j+1}\alpha_{n-j+1}^{-1} \end{array} \right|\right|_2^2 = \alpha_{n-j+1}^{-2}\left(1 + \left|\left|\mathbf{L}_{n-j}^{-1}\mathbf{a}_{n-j+1}\right|\right|_2^2\right) \tag{8.18}$$

A continuación intentaremos calcular un intervalo de valores entre los cuales se encuentre la norma de la columna j-ésima de $\mathbf{P}^{*/2}$, $r_{j_{\min}} \leq r_j \leq r_{j_{\max}}$.

Emplearemos algunas de las desigualdades que aparecen en [87] respecto a normas vectoriales y matriciales. Si, por ejemplo, $\mathbf{A} \in \mathbb{R}^{m\times n}$ y $\mathbf{x} \in \mathbb{R}^n$, entonces

$$||\mathbf{x}||_2 \leq ||\mathbf{x}||_1 \leq \sqrt{n}||\mathbf{x}||_2 \tag{8.19}$$

$$||\mathbf{A}||_2 \leq \sqrt{m}||\mathbf{A}||_\infty \tag{8.20}$$

$$||\mathbf{A}||_2 \leq \sqrt{n}||\mathbf{A}||_1 \tag{8.21}$$

$$||\mathbf{A}\mathbf{x}||_p \leq ||\mathbf{A}||_p||\mathbf{x}||_p \tag{8.22}$$

donde p denota la 1, 2 ó ∞-norma.

Centrándonos en el factor $\left|\left|\mathbf{L}_{n-j}^{-1}\mathbf{a}_{n-j+1}\right|\right|_2$ de (8.18), una cota inferior puede deducirse a partir de (8.22):

$$||\mathbf{a}_{n-j+1}||_2 = \left|\left|\mathbf{L}_{n-j}\mathbf{L}_{n-j}^{-1}\mathbf{a}_{n-j+1}\right|\right|_2 \leq ||\mathbf{L}_{n-j}||_2 \left|\left|\mathbf{L}_{n-j}^{-1}\mathbf{a}_{n-j+1}\right|\right|_2$$

por lo que

$$\frac{||\mathbf{a}_{n-j+1}||_2}{||\mathbf{L}_{n-j}||_2} \leq \left|\left|\mathbf{L}_{n-j}^{-1}\mathbf{a}_{n-j+1}\right|\right|_2$$

y por tanto

$$r_{j_{\min}}^2 = \alpha_{n-j+1}^{-2}\left(1 + \frac{||\mathbf{a}_{n-j+1}||_2^2}{||\mathbf{L}_{n-j}||_2^2}\right) \leq r_j^2$$

Podemos encontrar una cota menos costosa computacionalmente para $||\mathbf{L}_{n-j}||_2$, basándo-

nos en (8.21) y (8.20) por lo que

$$||\mathbf{L}_{n-j}||_2 \leq \sqrt{n-j}\,||\mathbf{L}_{n-j}||_{p'}$$

donde p' denota la 1 ó ∞-norma, y $\mathbf{L}_{n-j} \in \mathbb{R}^{(n-j)\times(n-j)}$. Con el objetivo de ajustar lo más posible la cota, $||\mathbf{L}_{n-j}||_{p''}$ denotará

$$||\mathbf{L}_{n-j}||_{p''} = \text{mín}\{||\mathbf{L}_{n-j}||_1, ||\mathbf{L}_{n-j}||_\infty\}$$

Por lo que un valor práctico para $r_{j_{\min}}$ será:

$$r^2_{j_{\min}} = \alpha^{-2}_{n-j+1}\left(1 + \frac{||\mathbf{a}_{n-j+1}||_2^2}{(n-j)\,||\mathbf{L}_{n-j}||^2_{p''}}\right)$$

Para calcular una cota superior nos basaremos en (8.21) y (8.22):

$$\left|\left|\mathbf{L}^{-1}_{n-j}\mathbf{a}_{n-j+1}\right|\right|_2 \leq \left|\left|\mathbf{L}^{-1}_{n-j}\mathbf{a}_{n-j+1}\right|\right|_1 \leq \left|\left|\mathbf{L}^{-1}_{n-j}\right|\right|_1 ||\mathbf{a}_{n-j+1}||_1 \tag{8.23}$$

donde según la definición de la 1-norma, y denotando por $\mathbf{L}^{-1}_{n-j,k}$ la k-ésima columna de $\mathbf{L}^{-1}_{n-j}$, con $k = 1, \ldots, n-j$, ésta será

$$\left|\left|\mathbf{L}^{-1}_{n-j}\right|\right|_1 = \max_{k=1,\ldots,n-j}\left\{\left|\left|\mathbf{L}^{-1}_{n-j,k}\right|\right|_1\right\}$$

Utilizando (8.19), relacionando la 1-norma de la columna k-ésima de $\mathbf{L}^{-1}_{n-j}$ con la 2-norma de la misma, y teniendo en cuenta que la columa k-ésima de $\mathbf{L}^{-1}_{n-j}$ tiene $n-j-k+1$ elementos no nulos, obtenemos:

$$\left|\left|\mathbf{L}^{-1}_{n-j,k}\right|\right|_1 \leq \sqrt{n-j-k+1}\left|\left|\mathbf{L}^{-1}_{n-j,k}\right|\right|_2$$

y ya que la columna k-ésima de $\mathbf{L}^{-1}_{n-j,k}$ es la columna $(j+k)$-ésima de $\mathbf{L}^{-1}_n$, su 2-norma es por definición $\left|\left|\mathbf{L}^{-1}_{n-j,k}\right|\right|_2 = r_{j+k}$, entonces

$$\begin{aligned}\left|\left|\mathbf{L}_{n-j,k}^{-1}\right|\right|_1 &\leq \sqrt{n-j-k+1}\cdot r_{j+k}\\ &\leq \sqrt{n-j-k+1}\cdot r_{(j+k)_{\max}}\end{aligned}$$

Por lo tanto:

$$\begin{aligned}\left|\left|\mathbf{L}_{n-j}^{-1}\right|\right|_1 &= \max_{k=1,\ldots,n-j}\left\{\left|\left|\mathbf{L}_{n-j,k}^{-1}\right|\right|_1\right\}\\ &\leq \max_{k=1,\ldots,n-j}\left\{\sqrt{n-j-k+1}\cdot\left|\left|\mathbf{L}_{n-j,k}^{-1}\right|\right|_2\right\}\\ &\leq \max_{k=1,\ldots,n-j}\left\{\sqrt{n-j-k+1}\cdot r_{(j+k)_{\max}}\right\}\\ &\leq \max_{s=1,\ldots,n-j}\left\{\sqrt{s}\cdot r_{(n-s+1)_{\max}}\right\}\end{aligned}$$

De modo que podemos reescribir (8.23) como:

$$\left|\left|\mathbf{L}_{n-j}^{-1}\mathbf{a}_{n-j+1}\right|\right|_2 \leq \max_{s=1,\ldots,n-j}\left\{\sqrt{s}\cdot r_{(n-s+1)_{\max}}\right\}||a_{n-j+1}||_1$$

Por lo que la cota superior obtenida es:

$$r_{j_{\max}}^2 = \alpha_{n-j+1}^2\left(1+||a_{n-j+1}||_1^2\left(\max_{s=1,\ldots,n-j}\{\sqrt{s}\cdot r_{(n-s+1)_{\max}}\}\right)^2\right)$$

El cómputo de los resultados implica un coste que no es determinista, ya que depende de cuál sea los valores de la matriz $\mathbf{P}^{-*/2}$. Dicho coste puede ir desde un valor despreciable hasta un valor del mismo orden de magnitud que la inversión total de la matriz $\mathbf{P}^{-*/2}$.

8.4.2. Algoritmo secuencial

En cada iteración i-ésima se deben conseguir ceros en la submatriz $\mathbf{H}_i^*$ de $\mathbf{V}_i$. Si empleamos rotaciones de Givens, podemos llevarlo a cabo anulando bien las filas, bien las columnas con idéntico coste. Si plantemos la anulación por filas, el algoritmo paralelo

que puede derivarse resulta ser mucho más eficiente, por lo que tomaremos esta estrategia como la de referencia. Con este esquema vuelve a surgir una nueva alternativa: el emplear una transformación de Householder para anular las últimas $q-1$ componentes de la fila en cuestión, reflejando el resultado en la primera, y a continuación rotar este elemento respecto al elemento diagonal de la misma fila de $\mathbf{P}_i^{-*/2}$.

8.4.3. Costes

A continuación expondremos los costes de tanto la versión híbrida (Householder + Givens) como de la versión basada exclusivamente en las rotaciones de Givens, realizando las comparaciones oportunas. El coste de la estimación del índice con mayor relación señal-ruido no es determinista y tendrá un valor comprendido entre $n^3/3$ flops y un coste que podemos considerar despreciable.

Coste de la versión híbrida

La anulación de la fila k-ésima de $\mathbf{H}_i^*$ implica el cálculo de una transformación de Householder de orden q y la aplicación de la misma a un total de $n-k+1$ filas (fila a anular y las inferiores de $\mathbf{H}_i^*$) y también a las filas no nulas de $\mathbf{\Gamma}_{i+1}$, es decir $(i+1)q$ filas; adicionalmente hay que calcular una rotación de Givens y aplicarla a un par de columnas de dimensión también $n-k+1+(i+1)q$.

El coste de calcular ambas transformaciones será $w_{CH}(q)+w_{CG}$ y el de aplicarlas a una fila de las $n-k+1+(i+1)q$, para anular la fila k-ésima de $\mathbf{H}_i^*$, será $(w_{AH}(q)+w_{AG})$, por lo que el coste de la anulación de la fila k-ésima es:

$$W_{\text{sec},i,k}(z) = (w_{CH}(q)+w_{CG}) + (w_{AH}(q)+w_{AG})(n-k+1+(i+1)q) \quad \text{flops}$$

y el coste de la anulación de todas las $k=1,\dots n$ filas de $\mathbf{H}_i^*$, es decir, el coste por iteración será:

$$\begin{aligned} W_{\mathrm{sec},i}(z) &= \sum_{k=1}^{n} W_{\mathrm{sec},i,k}(z) \\ &= \sum_{k=1}^{n} (w_{CH}(q) + w_{CG}) + (w_{AH}(q) + w_{AG})(n - k + 1 + (i+1)q) \\ &= n(w_{CH}(q) + w_{CG}) + (w_{AH}(q) + w_{AG}) \left(\frac{1}{2} n + \frac{1}{2} + (i+1)q \right) n \end{aligned}$$

y el de todo el algoritmo, despreciando los costes de orden inferior:

$$\begin{aligned} W_{\mathrm{sec}}(z) &= \sum_{i=0}^{m/q-1} W_{\mathrm{sec},i}(z) \\ &\approx \sum_{i=0}^{m/q-1} \left\{ (w_{AH}(q) + w_{AG}) \left(\frac{1}{2} n + (i+1)q \right) n \right\} \\ &\approx \frac{1}{2q} (w_{AH}(q) + w_{AG})(n^2 m + n m^2) \quad \text{flops} \end{aligned} \tag{8.24}$$

Adicionalmente, el coste basado en el modelo de llamadas, para aquéllas que son más siginificativas, es:

$$\begin{aligned} W_{\mathrm{sec}}(z) &= \sum_{i=0}^{m/q-1} \left\{ \sum_{r=1}^{n} \Big[\mathrm{w}_{\mathrm{LARF}}(q, n - r + 1 + (i+1)q) \right. \\ &\qquad \left. + \mathrm{w}_{\mathrm{ROT}}(n - r + 1 + (i+1)q) \Big] \right\} \\ &= \sum_{i=0}^{m/q-1} \sum_{r=1}^{n} \mathrm{w}_{\mathrm{LARF}}(q, r + (i+1)q) + \sum_{i=0}^{m/q-1} \sum_{r=1}^{n} \mathrm{w}_{\mathrm{ROT}}(r + (i+1)q) \end{aligned}$$

La figura 8.8 muestra el tiempo de ejecución del algoritmo secuencial para distintos valores del parámetro q.

En la gráfica volvemos a observar la misma peculiaridad que en el algoritmo OSIC-SRKF en cuanto a la curva correspondiente a $q = 50$, por lo que de nuevo, en este algoritmo

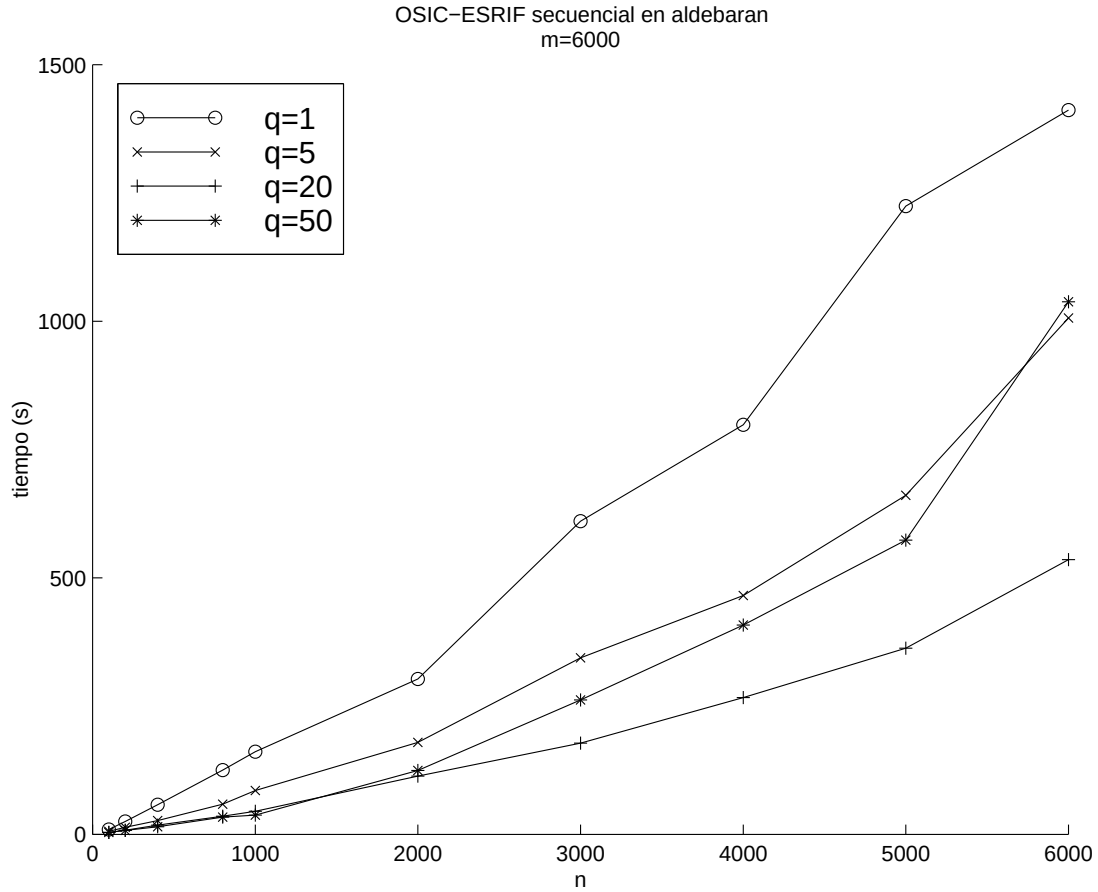

Figura 8.8: Tiempo secuencial de OSIC-SRIF para distintos valores del parámetro q

existe un mínimo de tiempo de ejecución que ha sido superado cuando el parámetro q toma este valor.

Coste de la versión Givens

En la versión basada en la aplicación exclusiva de rotaciones de Givens, cada aplicación de una transformación de Householder es sustituida por una secuencia de $q-1$ aplicaciones de rotaciones de Givens, por lo que el factor $(w_{AH}(q) + w_{AG})$ viene modificado por qw_{AG}, de modo que

$$\begin{aligned} W_{\text{sec},i}(z) &= \sum_{k=1}^{n} W_{\text{sec},i,k}(z) \\ &= \sum_{k=1}^{n} qw_{CG} + qw_{AG}(n-k+1+(i+1)q) \\ &= nqw_{CG} + qw_{AG}\left(\frac{1}{2}n + \frac{1}{2} + (i+1)q\right)n \end{aligned}$$

siendo el coste total del algoritmo:

$$\begin{aligned} W_{\text{sec}}(z) &= \sum_{i=0}^{m/q-1} W_{\text{sec},i}(z) \\ &\approx \frac{1}{2}w_{AG}(n^2m + nm^2) \quad \text{flops} \end{aligned}$$

El coste basado en el modelo de llamadas es:

$$\begin{aligned} W_{\text{sec}}(z) &= \sum_{i=0}^{m/q-1}\left\{\sum_{r=1}^{n}\left[\sum_{c=1}^{q} \text{w}_{\text{ROT}}(n-r+1+(i+1)q)\right]\right\} \\ &= q\sum_{i=1}^{m/q}\sum_{r=1}^{n} \text{w}_{\text{ROT}}(r+iq) \end{aligned} \tag{8.25}$$

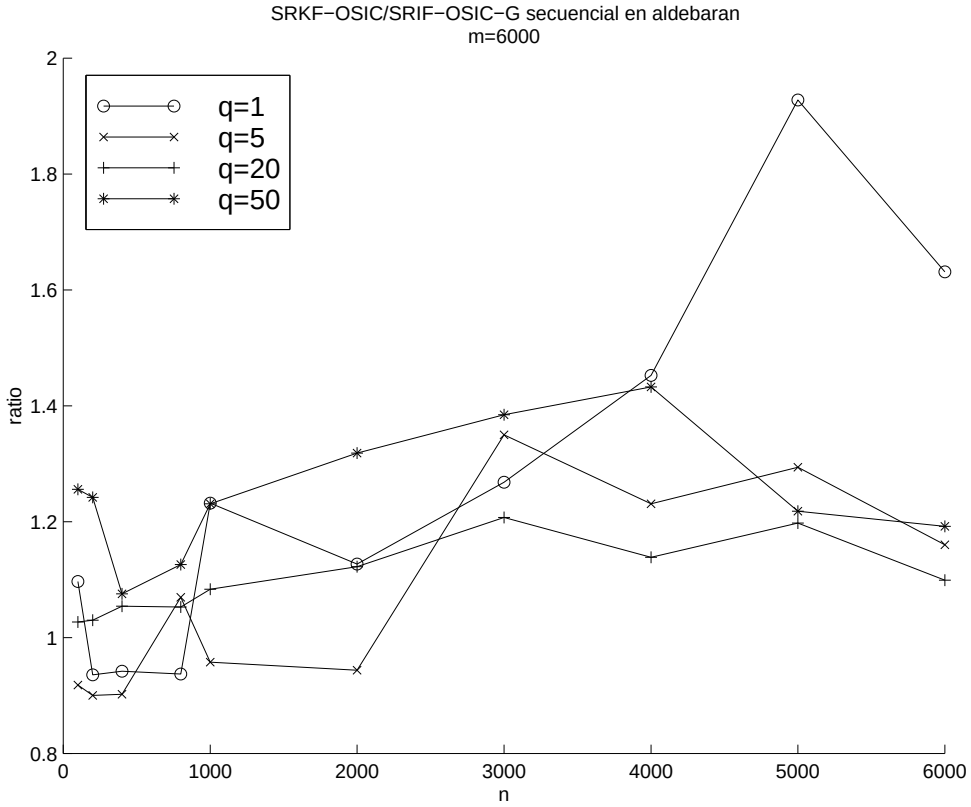

Figura 8.9: Ratio entre el tiempo de ejecución secuencial de la versión SRKF-OSIC y SRIF-OSIC-G

Comparación entre versiones OSIC SRKF y SRIF

Si comparamos las expresiones del coste basado en el modelo de llamadas del algoritmo SRKF-OSIC, (8.12) y el de SRIF-OSIC-G, versión Givens, (8.25) podemos comprobar fácilmente que la versión SRIF-OSIC-G tiene un tiempo de ejecución inferior a la SRKF-OSIC. El ahorro radica en la **no necesidad** de realizar una multiplicación matricial y la aplicación de ciertas rotaciones. La figura 8.9 muestra que dicho ratio podemos considerarlo superior a la unidad entorno al 20 %.

La comparación entre SRKF-OSIC y SRIF-OSIC-HG (versión híbrida) no es tan inmediata, debido a que aparecen operaciones distintas y el modelo oculta el comportamiento de la arquitectura con el algoritmo:

$$W_{\text{sec-SRKF-OSIC}}(z) >?< W_{\text{sec-SRIF-OSIC-HG}}(z)$$
$$\left(1+\tfrac{1}{2}w_{AG}\right)n^2m+\tfrac{1}{2}w_{AG}nm^2 >?< \tfrac{1}{2q}(w_{AH}(q)+w_{AG})(n^2m+nm^2)$$
$$\tfrac{m}{q}\mathrm{w}_{\mathrm{TRMM}}(q,n)+\tfrac{m}{q}n\sum_{r=1}^{q}\mathrm{w}_{\mathrm{ROT}}(r)+q\sum_{i=0}^{m/q-1}\sum_{c=1}^{n}\mathrm{w}_{\mathrm{ROT}}(c+(i+1)q)$$
$$>?<$$
$$\sum_{i=0}^{m/q-1}\sum_{r=1}^{n}\mathrm{w}_{\mathrm{LARF}}(q,r+(i+1)q)+\sum_{i=0}^{m/q-1}\sum_{r=1}^{n}\mathrm{w}_{\mathrm{ROT}}(r+(i+1)q)$$

La figura 8.10 muestra que dependiendo del valor del parámetro q podemos encontrarnos con que el algoritmo SRKF-OSIC es mejor que el SRIF-OSIC-HG y viceversa. Podemos observar que, a excepción de cuando $q=1$ y en términos generales, el algoritmo SRIF-OSIC-HG se va comportando mejor que el SRKF-OSIC, conforme mayor es el valor del parámetro q, debido al comportamiento asintótico del tiempo de ejecución en función de q, según se desprende de (8.24).

8.4.4. Algoritmo paralelo

La versión algorítmica de la que vamos a mostrar los detalles de la paralelización será la versión híbrida. El algoritmo paralelo está inspirado en el algoritmo paralelo descrito para la versión raíz cuadrada del filtro de información para el problema RLS, con una diferencia importante respecto al equilibrado. En aquél, la distribución por columnas implica un esquema de equilibrado perfecto dada la estructura de la matriz problema, pero en éste, conforme el índice del procesador aumenta, la carga va disminuyendo, lo cual se muestra simbólicamente en la figura 8.11.

Cada procesador anulará, cuando lo hayan hecho los procesadores de índice inferior, sus n_j filas asignadas de $\mathbf{H}_i^*$, empleando para ello sus n_j columnas asignadas de $\mathbf{P}_i^{-*/2}$ y de $\mathbf{H}_{\alpha,(i)}^{\dagger *}\mathbf{P}_i^{-*/2}$, denotando por c_{0_j} el índice de la primera de ellas. En la versión híbrida, para cada una de estas filas de $\mathbf{H}_i^*$, P_j debe calcular y aplicar una transformación de Householder a un total de $n-c_{0_j}+1-k+1+(i+1)q$ filas, y calcular y aplicar una

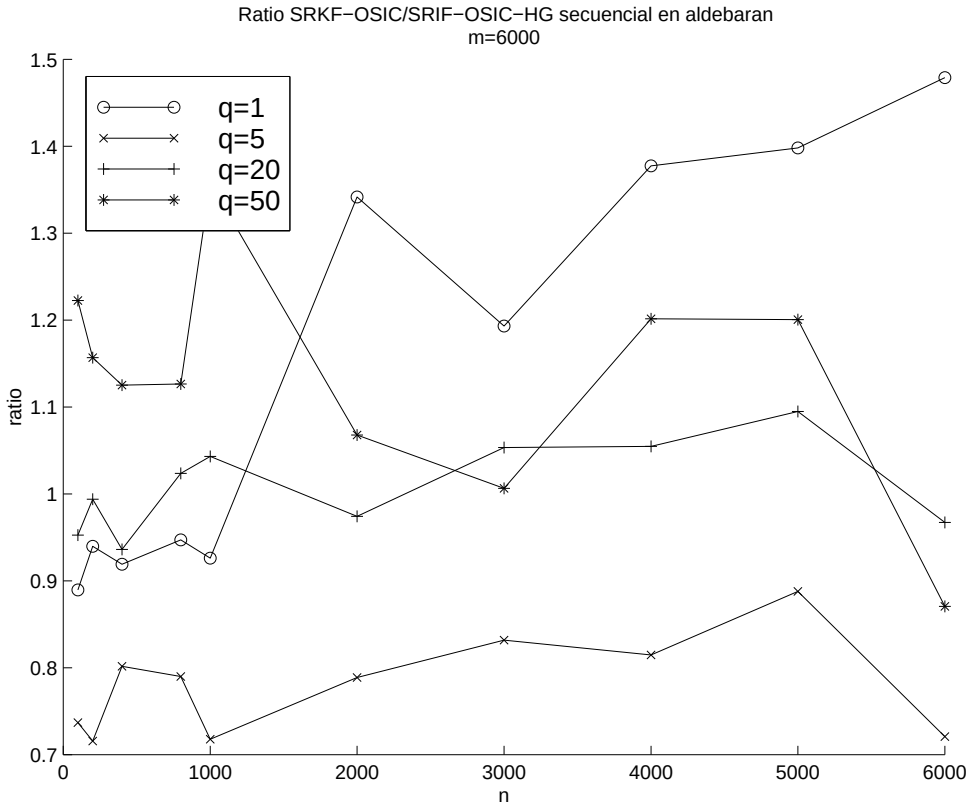

Figura 8.10: Ratio entre el tiempo de ejecución secuencial de la versión SRKF-OSIC y SRIF-OSIC-HG

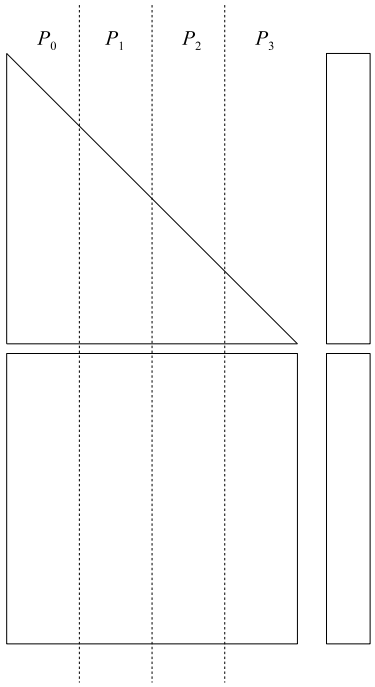

Figura 8.11: Particiones y carga para OSIC con SRIF modificado.

rotación de Givens y aplicarla a un par de vectores con la misma cantidad de filas. Para la versión de Givens, se deben calcular y aplicar un total de q rotaciones de Givens y aplicarlas todas ellas a pares de vectores con la misma dimensión que en el caso anterior.

Dada la gran similitud de ambas versiones, denotaremos por $w_{AHG/G}(q)$ la carga resultante de aplicar las transformaciones a la cantidad de filas indicadas en el párrafo anterior. De modo que para la versión híbrida $w_{AHG/G}(q) = w_{AH}(q)+w_G$ y para la versión de Givens, $w_{AHG/G}(q) = qw_G$.

El coste necesario para anular todas las k filas, $k = 1, \ldots, n_j$, será:

$$\begin{aligned} W_{P_j,i}(z) &= \sum_{k=1}^{n_j} w_{AHG/G}(q) \left[n - c_{0_j} + 1 - k + 1 + (i+1)q\right] \\ &= w_{AHG/G}(q) \left\{ \left[n - c_{0_j} + 2 + (i+1)q\right] n_j - \frac{1}{2}n_j^2 - \frac{1}{2}n_j \right\} \\ &= w_{AHG/G}(q) \left\{ \left[n - c_{0_j} + \frac{3}{2} + (i+1)q\right] n_j - \frac{1}{2}n_j^2 \right\} \quad \text{flops} \end{aligned} \tag{8.26}$$

Podemos comprobar que de esta forma, la carga de trabajo en cada iteración se reparte completamente, es decir, la sobrecarga aritmética por paralelización es nula:

$$\sum_{j=0}^{p-1} W_{P_j,i}(z) = W_{\text{sec},i}(z)$$

ya que, teniendo en cuenta que $n = \sum_{j=0}^{p-1} n_j$ y $c_{0_j} = 1 + \sum_{k=0}^{j-1} n_k$:

$$\begin{aligned}
\sum_{j=0}^{p-1} W_{P_j,i}(z) &= w_{AHG/G}(q) \sum_{j=0}^{p-1} \left\{ \left[n - c_{0_j} + \frac{3}{2} + (i+1)q \right] n_j - \frac{1}{2} n_j^2 \right\} \\
&= w_{AHG/G}(q) \sum_{j=0}^{p-1} \left\{ \left[n - 1 - \sum_{k=0}^{j-1} n_k + \frac{3}{2} + (i+1)q \right] n_j - \frac{1}{2} n_j^2 \right\} \\
&= w_{AHG/G}(q) \left\{ \left[n + \frac{1}{2} + (i+1)q \right] \sum_{j=0}^{p-1} n_j - \left(\frac{1}{2} \sum_{j=0}^{p-1} n_j^2 + n_j \sum_{k=0}^{j-1} n_k \right) \right\} \\
&= w_{AHG/G}(q) \left\{ \left[n + \frac{1}{2} + (i+1)q \right] \sum_{j=0}^{p-1} n_j - \frac{1}{2} \left(\sum_{j=0}^{p-1} n_j \right)^2 \right\} \\
&= w_{AHG/G}(q) \left\{ \left[n + \frac{1}{2} + (i+1)q \right] n - \frac{1}{2} n^2 \right\} \\
&= W_{\text{sec},i}(z)
\end{aligned}$$

El coste de llevar a cabo las m/q iteraciones será:

$$\begin{aligned}
W_{P_j}(z) &= \sum_{i=0}^{m/q-1} W_{P_j,i}(z) \\
&= \sum_{i=0}^{m/q-1} w_{AHG/G}(q) \left\{ \left[n - c_{0_j} + \frac{3}{2} + (i+1)q \right] n_j - \frac{1}{2} n_j^2 \right\} \quad \text{flops}
\end{aligned} \tag{8.27}$$

Consecuentemente:

$$\sum_{j=0}^{p-1} W_{P_j}(z) = W_{\text{sec}}(z) \tag{8.28}$$

8.4.5. Equilibrado de la carga

Para equilibrar la carga entre todos los procesadores debemos forzar a que se cumpla, en primera instancia, la expresión de la ecuación (2.8), que deberíamos traducir a nivel de iteración, debido a la fuerte dependencia temporal que van a tener los procesadores en la

línea segmentada (una vez que se ha superado el transitorio del llenado de la línea segmentada) para hacer mínimo el tiempo de espera o sincronización entre los procesadores:

$$W_{P_j,i}(z) = \alpha_j W_{\text{sec},i}(z), \quad \forall\, i, \forall\, j \tag{8.29}$$

En cierto instante de tiempo, el procesador P_j estará procesando la iteración i-ésima, mientras que P_k estará procesando la iteración $i-(k-j)$-ésima, por tanto el equilibrado perfecto se consigue cuando:

$$W_{P_j,i}(z)t_{w_j} = W_{P_k,i-(k-j)}(z)t_{w_k} = \frac{1}{S_{\text{máx}}(p)} W_{\text{sec},i}(z)t_{w_{\text{sec}}}, \forall\, i, \forall\, j \neq k$$

es decir, según (8.26) y (2.6)

$$\begin{aligned} w_{AHG/G}(q)\left\{\left[n - c_{0_j} + \tfrac{3}{2} + (i+1)q\right] n_j - \tfrac{1}{2}n_j^2\right\} t_{w_j} &= \\ w_{AHG/G}(q)\left\{\left[n - c_{0_k} + \tfrac{3}{2} + (i-(k-j)+1)q\right] n_k - \tfrac{1}{2}n_k^2\right\} t_{w_k} &= \\ \alpha_f w_{AHG/G}(q)\left\{\left[n + \tfrac{1}{2} + (i+1)q\right] n - \tfrac{1}{2}n^2\right\} t_{w_f}, &\qquad \forall\, i, \forall\, j \neq k \end{aligned}$$

De las anteriores expresiones, es complejo extraer la relación entre n_j y n_k para conseguir un equilibrado perfecto debido, no sólo a que podemos estar en un entorno heterogéneo ($t_{w_j} \neq t_{w_k}$), si no que en cada iteración, la carga de trabajo es distinta, tal y como puede observarse por la presencia del factor i en las expresiones. El intentar conseguir el equilibrado según las expresiones anteriores, desembocaría en un esquema dinámico de equilibrado de carga. Por ello, relajaremos las condiciones de equilibrado para conseguir un resultado práctico, aún a sabiendas de que no se obtendrá un resultado perfecto. Primeramente supondremos despreciable la diferencia de carga entre los procesadores respecto a su índice, debida a su vez a la diferencia entre los índices de iteraciones empleados por dos procesadores en el mismo instante de tiempo, habida cuenta que la expresión $|k-j| < p$ dejará de tener influencia cuando el índice de iteración $i \gg p$, es

decir $W_{P_j,i}(z)t_{w_j} = W_{P_k,i}(z)t_{w_k}, \forall\, i, \forall\, j \neq k$, y para eliminar de alguna forma, la influencia que tiene el índice de la iteración sobre la carga, supondremos que deseamos el equilibrado para el caso más desfavorable que es cuando más carga hay en el sistema ($i = m/q - 1$), es decir $W_{P_j,m/q-1}(z)t_{w_j} = W_{\mathrm{sec},m/q-1}(z)t_{w_f}, \forall\, j$, por lo tanto

$$\begin{aligned} &\left\{\left[n - c_{0_j} + \tfrac{3}{2} + m\right] n_j - \tfrac{1}{2}n_j^2\right\} t_{w_j} = \\ &\quad \alpha_f \left\{\left[n + \tfrac{1}{2} + m\right] n - \tfrac{1}{2}n^2\right\} t_{w_f}, \qquad \forall\, j \end{aligned}$$

A partir de aquí podemos resolver el cálculo de n_0, siendo $c_{0_0} = 1$, resolviendo una ecuación de segundo grado. Podríamos continuar con n_1, siendo ahora $c_{0_1} = 1 + n_0$, y así sucesivamente.

De la anterior igualdad se desprende que las ecuaciones de equilibrado son las mismas para la versión híbrida que para la versión de Givens.

La solución obtenida para los distintos n_j, nos proporcionará resultados reales y no enteros, generalmente, por lo que habrá que realizar algún tipo de operación de tipo *redondeo hacia menos infinito* o simplemente truncado, ya que los valores implicados son positivos. Residualmente, puede haber como máximo un total de $p - 1$ columnas sin asignar como consecuencia del redondeo anterior, que deben ser repartidas. Habida cuenta que el primer procesador tendrá cierta sobrecarga adicional (carga de $\mathbf{H}_i^*$ e inicialización de la variable $\mathbf{\Gamma}_{i+1}$) parece razonable realizar el reparto entre los $p - 1$ procesadores restantes de mayor índice. El reparto de columnas *sobrantes* por el efecto del truncado no tendrá demasiada influencia si n es grande, debido a que la cantidad de columnas residuales a añadir a un procesador es de sólamente una.

Este planteamiento del equilibrado de la carga en una línea segmentada unidireccional presenta ciertos inconvenientes: el equilibrado sólo ha sido calculado para la última iteración ($i = m/q - 1$) y sobre todo, está basado en el modelo analítico de tiempo de ejecución que hemos planteado y que quizá no coincida con el tiempo de ejecución real en la máquina, por lo que la consecuencia más inmediata será cierto desequilibrio residual.

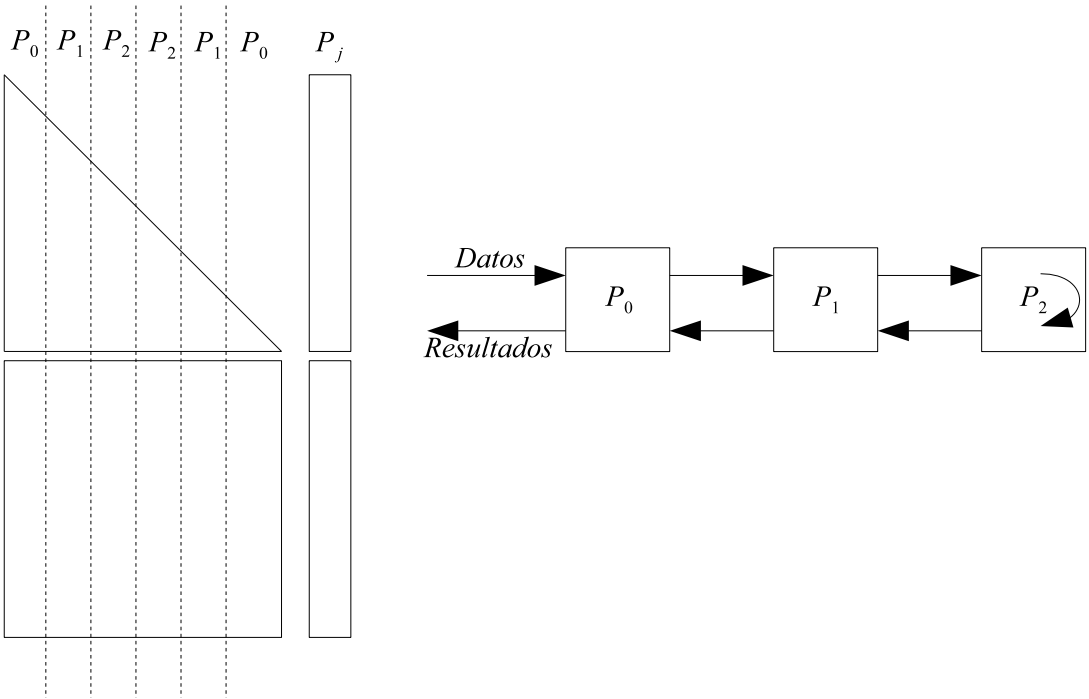

Figura 8.12: Línea segmentada bidireccional.

Podemos plantear una alternativa de equilibrado más independiente del modelo analítico y que consiste en realizar una asignación de carga de tipo *especular*, dando lugar a una línea segmentada en dos sentidos (ida y vuelta). Ésto no altera las semejanzas entre la versión híbrida y la de Givens.

8.4.6. Línea segmentada bidireccional

Para este algoritmo paralelo, tal y como ocurría con algunos algoritmos anteriores, podemos plantear como alternativa de equilibrado la basada en una línea segmentada bidireccional. Supongamos que tenemos t particiones, es decir, t grupos de columnas consecutivas de manera que $\sum_{k=0}^{t-1} n_k = n$. Si relacionamos especularmente los índices de las particiones, es decir, si $0 \leq j \leq t/2 - 1$ y $t/2 \leq j' \leq t - 1$, entonces $j' = t - j - 1$. Deberíamos forzar a que $n_{j'} = n_j$; por ejemplo, con $t = 6$ particiones, $n_0 = n_5$, $n_1 = n_4$ y $n_2 = n_3$. Gráficamente se muestra en la figura 8.12.

De (8.26) puede observarse que el tiempo de ejecución cumple las condiciones de la función (4.16) expuestas en el apartado 4.2.6, donde:

- $\beta = \rho = -1$

- $f(n,q)$ es en este caso $f(n,q,i) = n + \frac{3}{2} + (i+1)q$

- $c_{0_j} + c_{0_{j'}}$ tiene la siguiente expresión: el valor de c_{0_j} es $1 + \sum_{k=0}^{j-1} n_k$, o alternativamente $n + 1 - \sum_{k=j}^{t-1} n_k$, y el de $c_{0_{j'}} = n + 1 - \sum_{k=j'}^{t-1} n_k$, para el cual, el rango de los índices del sumatorio es $t/2 \leq k \leq t-1$, por lo que el límite superior del sumatorio es equivalente a 0 y el inferior a j, que reordenándolo podemos reescribirlo como

$$c_{0_{j'}} = n + 1 - \sum_{k=0}^{j} n_k$$

entonces

$$\begin{aligned} c_{0_j} + c_{0_{j'}} &= 1 + \sum_{k=0}^{j-1} n_k + n + 1 - \sum_{k=0}^{j} n_k \\ &= 2 + n - n_j \\ &= f_1(n) - \frac{\beta}{\rho} n_j \\ f_1(n) &= 2 + n \end{aligned}$$

La carga suma de las particiones j y j' sería:

$$\begin{aligned}
W_{j,i}(z) + W_{j',i}(z) &= w_{AHG/G}(q)\left\{\left[n - c_{0_j} + \frac{3}{2} + (i+1)q\right] n_j - \frac{1}{2}n_j^2\right\} \\
&\quad + w_{AHG/G}(q)\left\{\left[n - c_{0_{j'}} + \frac{3}{2} + (i+1)q\right] n_{j'} - \frac{1}{2}n_{j'}^2\right\} \\
&= w_{AHG/G}(q)\left\{\left[2n - c_{0_j} - c_{0_{j'}} + 3 + 2(i+1)q\right] n_j - n_j^2\right\} \\
&= w_{AHG/G}(q)\left\{[2n - 2 - n + n_j + 3 + 2(i+1)q]\, n_j - n_j^2\right\} \\
&= w_{AHG/G}(q)\left\{[n + n_j + 1 + 2(i+1)q]\, n_j - n_j^2\right\} \\
&= w_{AHG/G}(q)\left\{[n + 1 + 2(i+1)q]\, n_j\right\}
\end{aligned}$$

De esta forma, las particiones j y $j' = t - j - 1$ tienen conjuntamente una carga independiente de los factores c_{0_j} y $c_{0_{j'}}$, dependiendo sólo de n_j, n, i y q. Por lo que en un sistema con p procesadores tendríamos $t = 2p$ particiones, asignándole al procesador P_j, $0 \leq j \leq p-1$, la carga de las particiones j y $j' = t - j - 1$:

$$W_{P_j,i}(z) = W_{j,i}(z) + W_{j',i}(z)$$

De nuevo, podemos verificar fácilmente que $\sum_{j=0}^{p-1} W_{P_j,i}(z) = W_{\text{seq},i}(z)$.

El equilibrado se consigue igualando el tiempo de ejecución en P_j, $\forall\, j$, a la mínima fracción del tiempo secuencial que permite el incremento de velocidad, es decir, reescribiendo (2.8):

$$w_{AHG/G}(q)\left\{[n + 1 + 2(i+1)q]\, n_j\right\} t_{w_j} = \alpha_f w_{AHG/G}(q)\left\{\frac{n}{2} + \frac{1}{2} + (i+1)q\right\} n t_{w_f}$$

por lo que

$$n_j = \frac{\alpha_f t_{w_f}}{t_{w_j}} \frac{n}{2} = \frac{n}{2}\alpha_j$$

Y ya que $n_{j'} = n_j$, el total de columnas que trabajará P_j será $n\alpha_j$.

Si no se cumpliera la hipótesis de que la función del modelo de tiempo de ejecución no coincidiera con la expuesta en (4.16), son aplicables las mismas consideraciones en cuanto al desequilibrio que la expuesta en la expresión (4.18).

Recordemos que la propuesta de equilibrado no tiene en cuenta que en las dos particiones j y j' se está trabajando en iteraciones diferentes, por lo que va a existir cierto desequilibrio residual, cuyo cómputo viene a continuación. Las ecuaciones de equilibrado se han basado en que

$$W_{P_j,i}(z) = W_{j,i}(z) + W_{j',i}(z)$$

cuando, para ser más precisos, debería haber sido

$$W_{P_j,i}(z) = W_{j,i}(z) + W_{j',i+2j-(2p-1)}(z)$$

ya que en cierto instante de tiempo, P_j está trabajando la partición j en la iteración i —*ida*—, y además, la partición $j' = 2p - j - 1$ en la iteración $i + 2j - (2p - 1)$ —*vuelta*—, como máximo (si el sistema estuviera funcionando de manera perfectamente síncrona). Esta falta de ajuste es origen de desequilibrio, ya que la carga real para P_j es:

$$\begin{aligned} W_{P_j,i}(z) &= \left[n - c_{0_j} + \frac{3}{2} + (i+1)q\right] n_j - \frac{1}{2}n_j^2 \\ &\quad + \left[n - c_{0_{j'}} + \frac{3}{2} + (i + 2j - (2p-1) + 1)q\right] n_j - \frac{1}{2}n_j^2 \\ &= [2n - (2 + n - n_j) + 3 + 2(i+1)q + (2j - 2p + 1)q]\, n_j - n_j^2 \\ &= [n + 1 + 2(i+1)q + (2j - 2p + 1)q]\, n_j \end{aligned}$$

cuando la prevista fue $[n + 1 + 2(i + 1)q]\, n_j$. La diferencia respecto a lo previsto es $(2j -$

$2p+1)qn_j$, que resulta ser negativa, es decir, los procesadores estan menos cargados que lo previsto, pero entre ellos existe una diferencia de carga proporcional a los parámetros q, n_j y a su índice, es decir, a mayor índice de procesador, mayor carga. De todas formas, esta cantidad, podemos considerarla despreciable en la mayor parte de las situaciones prácticas.

8.4.7. Análisis de las comunicaciones

Independientemente de si utilizamos la versión híbrida o la de Givens, en cada iteración i-ésima, un procesador P_j, $0 \leq j < p-1$, debe transferir al siguiente en la cadena la parte todavía no nula de $\mathbf{H}_i^*$, es decir, $n-c_{0_{j+1}}+1$ filas de q elementos, con $c_{0_{j+1}} = 1+\sum_{k=0}^{j} n_k = 1+n-\sum_{k=j+1}^{p-1}$, y también la parte todavía no nula de $\mathbf{\Gamma}_{i+1}$, es decir $i+1$ filas también de q elementos, es decir, un total de $((i+1)+\sum_{k=j+1}^{p-1} n_k)q$ elementos. Podemos observar, por tanto, que conforme crece el índice del procesador, disminuye la cantidad de información a transferir al siguiente, y lo opuesto respecto al índice de la iteración i.

Suponiendo un equilibrado perfecto, cuando todos los procesadores acaben su segmento de iteración, entonces P_0 transmitirá a P_1 los resultados de su iteración i-ésima, P_1 transmitirá a P_2 los de su iteración $i-1$-ésima, etc. (por simplicidad en las expresiones, obviaremos los intervalos temporales en los que la línea segmentada se está llenando o vaciando). La figura 8.13 muestra esta situación para una línea segmentada unidireccional; para el caso bidireccional, la organización de las comunicaciones sigue siendo la misma, sólo que los procesadores están ubicados de manera especular.

De manera que en ese intervalo temporal destinado a las comunicaciones tendremos $p-1$ transferencias que podrá realizarse simultáneamente, en serie o con cierto grado de solapamiento, dependiendo de las características de la red subyacente. Supongamos que el tiempo de comunicación punto a punto sigue una función afín de la forma

$$T_{C,P_j,i}(z,p) = \beta + \tau \left((i+1) + \sum_{k=j+1}^{p-1} n_k \right) q$$

donde β es el tiempo de establecimiento de la comunicaición, y τ el tiempo necesario en

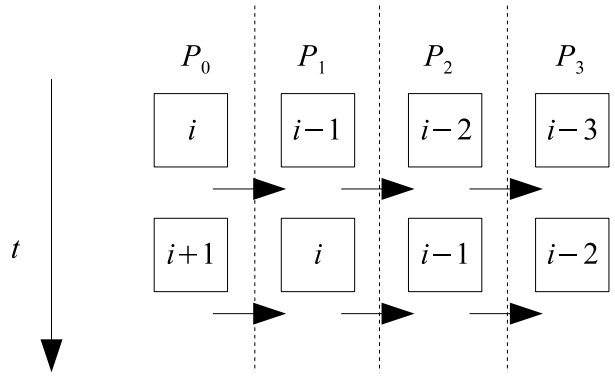

Figura 8.13: Comunicaciones en la línea segmentada unidireccional.

transmitir un elemento.

Si las comunicaciones pueden realizarse simultáneamente, el tiempo de comunicación será:

$$\begin{aligned} T_{C,i}^{(A)}(z,p) &= \text{máx}\{T_{C,P_0,i}(z), T_{C,P_1,i-1}(z), \cdots, T_{C,P_{p-2},i-(p-1)}(z)\} \\ &= T_{C,P_0,i}(z) \end{aligned}$$

ya que como hemos comentado, la cantidad de elementos a transferir aumenta conforme menor es el índice de procesador y mayor el índice de iteración. Por lo tanto:

$$T_{C,i}^{(A)}(z,p) = \beta + \tau(n - n_0 + i + 1)q$$

Y el tiempo total destinado a las comunicaciones en todas las iteraciones del algoritmo:

$$\begin{aligned} T_C^{(A)}(z,p) &= \sum_{i=0}^{m/q-1} T_{C,i}^{(A)}(z,p) \\ &= \sum_{i=0}^{m/q-1} [\beta + \tau(n-n_0+i+1)q] \\ &= \beta\frac{m}{q} + \tau(n-n_0+1)m + \tau q \sum_{i=0}^{m/q-1} i \\ &= \beta\frac{m}{q} + \tau(n-n_0+1)m + \tau\frac{1}{2}\left(\frac{m^2}{q} - m\right) \\ &< \beta\frac{m}{q} + \tau(n+1)m + \tau\frac{1}{2}\left(\frac{m^2}{q} - m\right) \\ &= \Theta(nm) + \Theta\left(\frac{m^2}{q}\right) \end{aligned} \tag{8.30}$$

si n es del orden de m, entonces $T_C^{(A)}(z,p) = \Theta(mn)$.

En el caso más desfavorable, es decir, si las comunicaciones deben llevarse a cabo en serie, el tiempo empleado será:

$$T_{C,i}^{(B)}(z,p) = T_{C,P_0,i}(z) + T_{C,P_1,i-1}(z) + \cdots + T_{C,P_{p-2},i-(p-1)}(z)$$

entonces

$$\begin{aligned} T_{C,i}^{(B)}(z,p) &= \beta + \tau\left[(n_1 + n_2 + \cdots + n_{p-1}) + i + 1\right] q \\ &\quad +\beta + \tau\left[(n_2 + \cdots + n_{p-1}) + i\right] q \\ &\quad +\cdots + \\ &\quad +\beta + \tau\left[(n_{p-1}) + i - (p-3)\right] q \\ &= (p-1)\beta \\ &\quad +\tau\left[(n_1 + n_2 + \cdots + n_{p-1}) + (n_2 + \cdots + n_{p-1}) + \cdots n_{p-1}\right] q \\ &\quad +\tau\left[(i+1) + i + \cdots + (i - (p-3))\right] q \end{aligned} \tag{8.31}$$

A continuación buscaremos expresiones más simplificados para los dos últimos sumandos de (8.31). Para el primero de ellos buscaremos una cota superior: el caso más desfavorable y extremo para los sumatorios sería que $n_{p-1} = n$ y $n_k = 0$, $1 \leq k \leq p-2$, por lo que

$$\tau\left[(n_1 + n_2 + \cdots + n_{p-1}) + (n_2 + \cdots + n_{p-1}) + \cdots n_{p-1}\right] q \leq \tau(p-1)nq$$

Y para el segundo sumando de (8.31)

$$\begin{aligned} &\tau\left[(i+1) + i + (i-1) + (i-2) + \cdots + (i - (p-3))\right] q = \\ &\quad = \tau\left[(i+1) + i + (p-3)i - \textstyle\sum_{k=1}^{p-3} k\right] q \\ &\quad = \tau\left[(p-1)i + 1 - \frac{(p-3)(p-2)}{2}\right] q \end{aligned}$$

Por lo tanto, el tiempo de comunicación en la línea segmentada unidireccional, cuando el procesador P_0 está elaborando la iteración i-ésima tiene como cota superior:

$$T_{C,i}^{(B)}(z,p) < (p-1)\beta + \tau(p-1)nq + \tau\left[(p-1)i + 1 - \frac{(p-3)(p-2)}{2}\right] q$$

Por simplificar las expresiones, despreciaremos el tiempo de comunicación empleado cuando se está llenando o vaciando la línea segmentada, por lo que

$$\begin{aligned} T_C^{(B)}(z,p) &< \sum_{i=0}^{m/q-1} \{(p-1)\beta + \tau(p-1)nq \\ &\quad + \tau\left[(p-1)i + 1 - \frac{(p-3)(p-2)}{2}\right] q\Big\} \\ &< (p-1)\beta\frac{m}{q} + \tau(p-1)nm \\ &\quad +\tau\left[1 - \frac{(p-3)(p-2)}{2}\right] m + \tau q(p-1)\sum_{i=0}^{m/q-1} i \\ &< (p-1)\beta\frac{m}{q} + \tau(p-1)nm \\ &\quad +\tau\left[1 - \frac{(p-3)(p-2)}{2}\right] m + \tau(p-1)\left(\frac{m^2}{q} - m\right) \end{aligned} \tag{8.32}$$

O bien

$$T_C^{(B)}(z,p) = \Theta\left(\frac{pm}{q}\right) + \Theta\left(mnp\right) + \Theta\left(mp^2\right) + \Theta\left(\frac{pm^2}{q}\right)$$

Es complejo el obtener el tiempo destinado a las comunicaciones a partir de medidas temporales en el propio algoritmo paralelo debido a que es difícil separar el tiempo de espera por desequilibrio en la carga, del propio tiempo dedicado a la comunicación, máxime si solapamos cálculo y comunicaciones, como en el caso de línea segmentada bidireccional, para poder atender de la forma más equitativa posible a dos flujos de información. Estos tiempos deben entonces ser obtenidos a partir de tests independientes del algoritmo, pero reflejando la situación real.

Para el caso bidireccional, las conclusiones son similiares, con la diferencia de que las comunicaciones prácticamente se duplican, con el consiguiente incremento en el coste.

8.4.8. Escalabilidad

La escalabilidad por isoeficiencia puede expresarse viendo en qué orden de magnitud debe aumentarse la carga en función del aumento en la cantidad de procesadores. Ello se puede deducir forzando a que el tiempo de ejecución del algoritmo secuencial sea proporcional al tiempo de sobrecarga, [71]. En nuestro caso, a partir de (8.27):

$$T_{\text{sec}}(z) = \Theta(m^2 n) + \Theta(mn^2)$$

Según (8.28) no existe sobrecarga por cálculo, sino sólo por comunicaciones:

$$T_O(z,p) = pT_C(z,p)$$

Si las comunicaciones pueden realizarse simultáneamente, entonces, según (8.30)

$$pT_C(z,p) = \Theta(mnp) + \Theta\left(\frac{m^2 p}{q}\right)$$

Por lo que para que $T_{\text{sec}}(z) = KpT_C(z,p)$, debe cumplirse el caso más desfavorables de los siguientes:

- $\Theta(m^2 n) = \Theta(mnp) \Rightarrow m = \Theta(p)$
- $\Theta(mn^2) = \Theta(mnp) \Rightarrow n = \Theta(p)$
- $\Theta(m^2 n) = \Theta\left(\frac{m^2 p}{q}\right) \Rightarrow n = \Theta\left(\frac{p}{q}\right)$
- $\Theta(mn^2) = \Theta\left(\frac{m^2 p}{q}\right) \Rightarrow \Theta\left(\frac{n^2}{m}\right) = \Theta\left(\frac{p}{q}\right)$. En la mayor parte de las situaciones prácticas, podemos considerar que m es del orden de n, por lo que podemos concretar que n y $m = \Theta\left(\frac{p}{q}\right)$.

Por lo tanto, la condición más restrictiva para el escalado es que $m = \Theta(p)$ y $n = \Theta(p)$. Con estas consideraciones, podemos concluir que el sistema paralelo es altamente escalable.

Si por el contrario, las comunicaciones deben realizarse en serie, entonces, según (8.32):

$$pT_C(z,p) = \Theta\left(\frac{p^2m}{q}\right) + \Theta\left(mnp^2\right) + \Theta\left(mp^3\right) + \Theta\left(\frac{p^2m^2}{q}\right)$$

Para lo cual, debemos escoger el caso más desfavorable de todas las combinaciones siguientes:

- $\Theta\left(m^2n\right) = \Theta\left(\frac{p^2m}{q}\right) \Rightarrow mn = \Theta\left(\frac{p^2}{q}\right)$. Si m y n son del mismo orden de magnitud, entonces, m y $n = \Theta\left(\frac{p}{q^{1/2}}\right)$
- $\Theta\left(m^2n\right) = \Theta\left(mnp^2\right) \Rightarrow m = \Theta\left(p^2\right)$
- $\Theta\left(m^2n\right) = \Theta\left(mp^3\right) \Rightarrow mn = \Theta\left(p^3\right)$. Es decir, m y $n = \Theta\left(p^{3/2}\right)$.
- $\Theta\left(m^2n\right) = \Theta\left(\frac{p^2m^2}{q}\right) \Rightarrow n = \Theta\left(\frac{p^2}{q}\right)$
- $\Theta\left(mn^2\right) = \Theta\left(\frac{p^2m}{q}\right) \Rightarrow n = \Theta\left(\frac{p}{q^{1/2}}\right)$
- $\Theta\left(mn^2\right) = \Theta\left(mnp^2\right) \Rightarrow n = \Theta\left(p^2\right)$
- $\Theta\left(mn^2\right) = \Theta\left(mp^3\right) \Rightarrow n = \Theta\left(p^{3/2}\right)$
- $\Theta\left(mn^2\right) = \Theta\left(\frac{p^2m^2}{q}\right) \Rightarrow \frac{n^2}{m} = \Theta\left(\frac{p^2}{q}\right)$. Por lo tanto n y $m = \Theta\left(\frac{p^2}{q}\right)$.

Por lo tanto, las condiciones más restrictivas son que ambas dimensiones crezcan como $n = \Theta\left(p^2\right)$ y $m = \Theta\left(p^2\right)$.

La pobre escalabilidad obtenida es debida a la importante cantidad de información que debe transmitirse cada vez que finaliza una iteración del algoritmo paralelo, de un procesador al siguiente, cuando no es posible simultanear en el tiempo dichas transferencias. Por lo que se concluye que para que el sistema paralelo sea escalable las comunicaciones entre los procesadores contiguos deben ser tales que puedan simultanearse lo más posible en el tiempo.

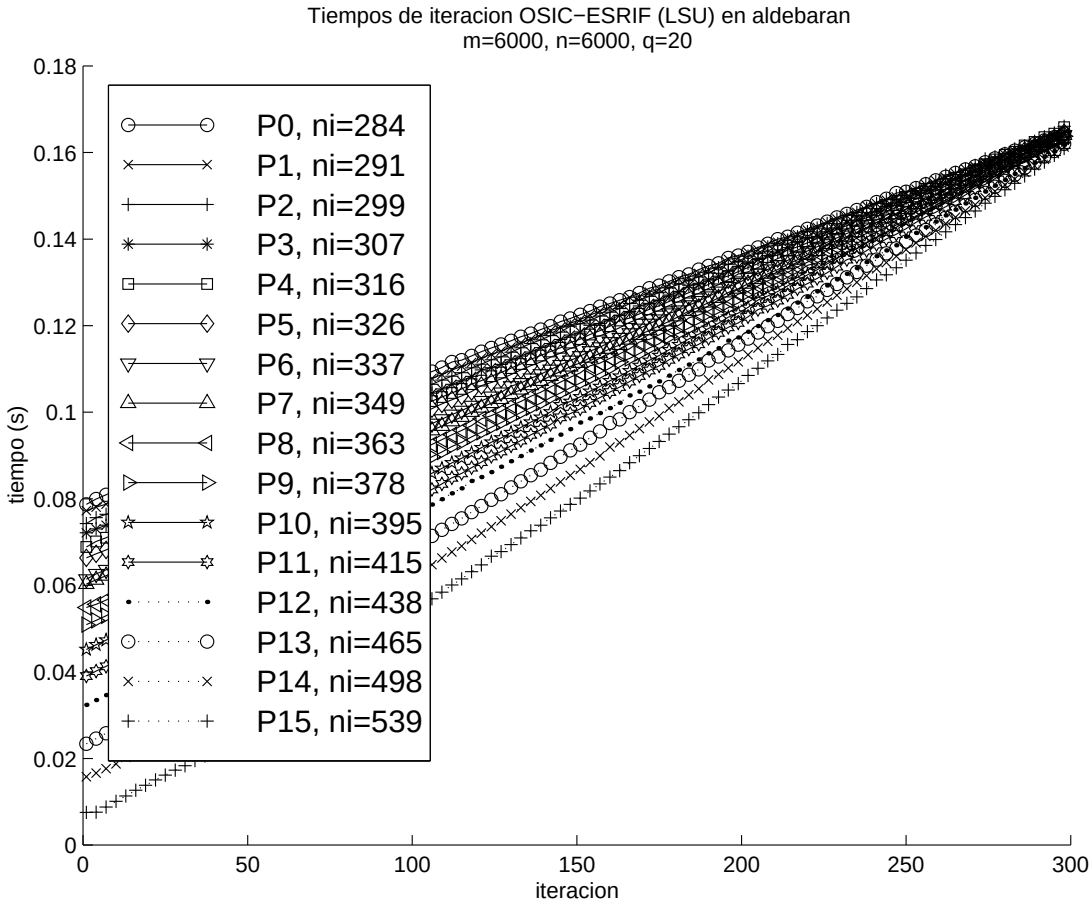

Figura 8.14: Equilibrado de carga en la línea segmentada unidireccional.

8.4.9. Resultados experimentales

Los siguientes resultados han sido obtenidos de ejecuciones en la máquina *Aldebarán*.

Equilibrado de la carga en línea segmentada unidireccional

En la figura 8.14 se muestra el tiempo aritmético que emplea cada procesador con el esquema de equilibrado propuesto para la línea segmentada unidireccional (ni denota la cantidad de columnas asignadas a cada procesador y los puntos han sido diezmados para una mejor visualización), apreciándose cómo se consigue el equilibrio de la carga en el instante previsto, es decir, en la(s) última(s) iteracion(es).

En la figura 8.15 observamos el mismo resultado, pero para un n pequeño en comparación con m, con las siguientes peculiaridades: la cantidad de columnas debe ser un

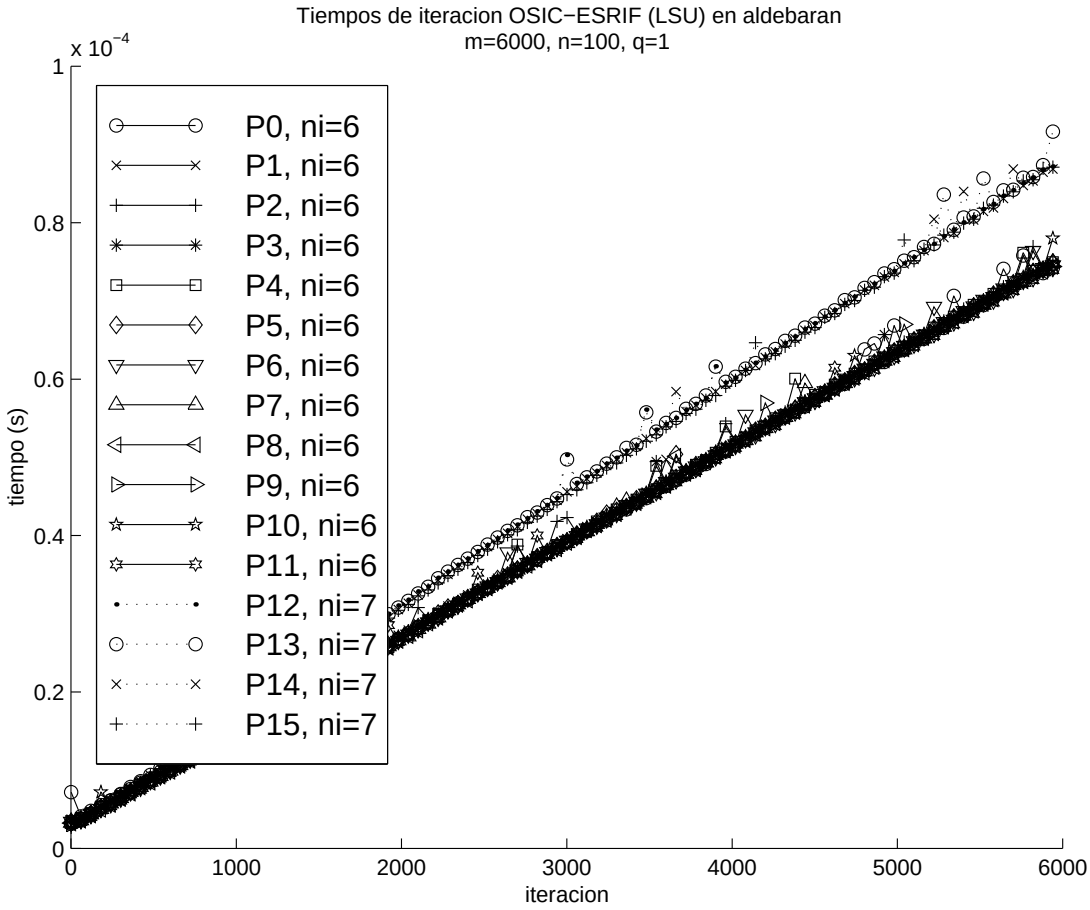

Figura 8.15: Equilibrado de carga en la línea segmentada unidireccional para valores de n pequeños.

número entero y ello implica que con un valor de n relativamente pequeño, el reparto no podrá ser equitativo; las columnas *sobrantes* se han repartido entre los procesadores de mayor índice, por lo que han confluido varias causas de desequilibrio, mostrando, además, cierta *divergencia* en la carga al final de las iteraciones. También podemos apreciar que para este valor relativamente pequeño de n la diferencia de carga entre procesadores (con el mismo número de columnas asignadas) es mínima a lo largo de las iteraciones, ya que

$$W_{P_j,i}(z)t_{w_j} - W_{P_k,i}(z)t_{w_k}\big|_{t_{w_j}=t_{w_k},n_j=n_k} \quad = \quad w_{AHG/G}(q)\left[-c_{0_j} + c_{0_k} + (k-j)q\right] n_k t_{w_j}$$

es una cantidad relativamente pequeña en comparación con la carga total, ya que el factor $(i+1)q$ tiene más peso en $W_{P_j,i}(z)t_{w_j}$ que $\left[-c_{0_j} + c_{0_k} + (k-j)q\right]$ conforme i va creciendo.

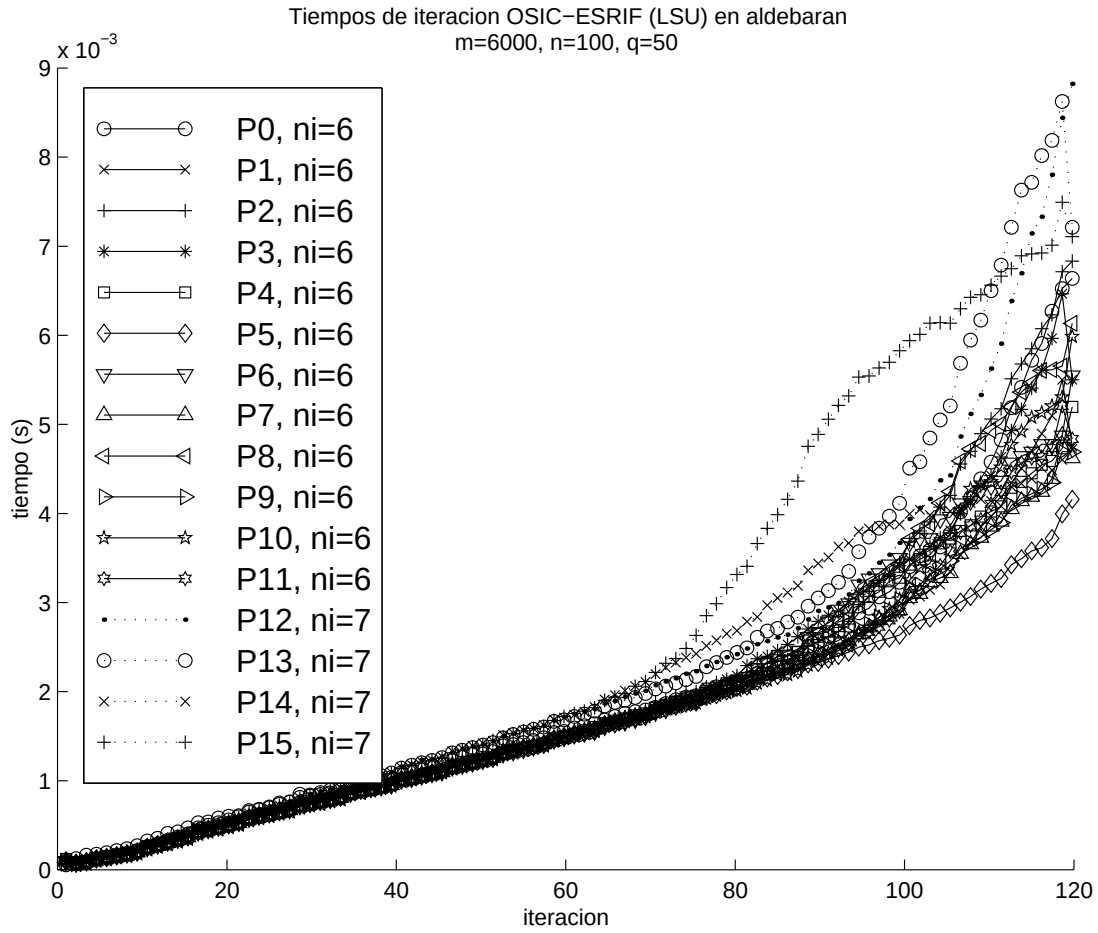

Figura 8.16: Equilibrado de carga en la línea segmentada unidireccional para valores de q grandes.

En la figura 8.16 vemos de nuevo la misma gráfica, esta vez para un valor de q relativamente grande, observando cómo al final cada procesador tiene un tiempo de iteración distinto debido a que las pequeñas diferencias en las estructuras de datos de cada uno son más sensibles a valores de q relativamente altos.

También podemos observar en la figura 8.17 la influencia de los tamaños de los datos en los tiempos de iteración, apreciando la diferencia de pendientes en los distintos procesadores para distintos valores de las iteraciones i, ya que el tamaño de los datos que están siendo tratados por cada procesador, no sólo dependen de la cantidad de columnas asignadas a cada uno, sino también de la iteración que se esté procesando. Esta situación es más apreciable para valores de n pequeños.

La figura 8.18 compara los tiempos aritméticos paralelos de cada procesador con el

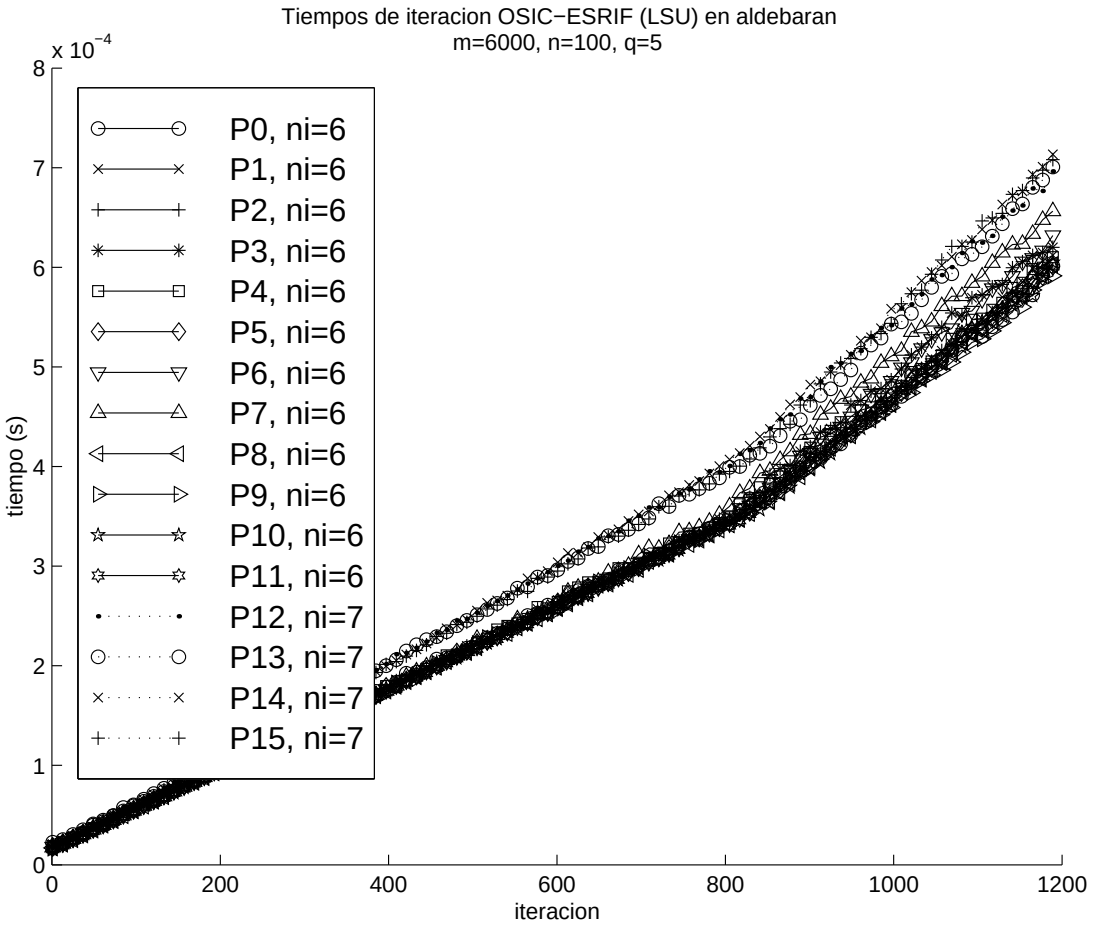

Figura 8.17: Equilibrado de carga en la línea segmentada unidireccional y la influencia del índice de iteración i en la velocidad de proceso.

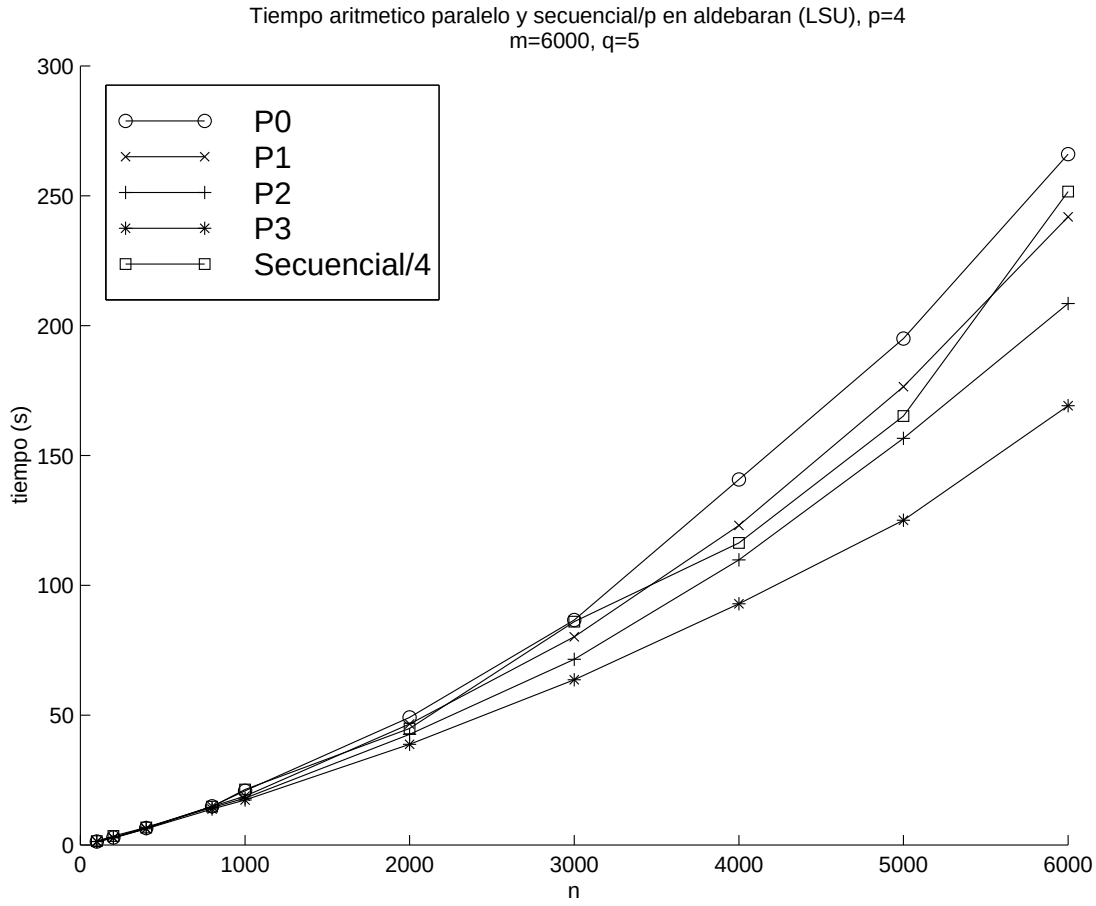

Figura 8.18: Tiempos aritméticos paralelos y tiempo secuencial dividido por el número de procesadores en línea segmentada unidireccional.

valor ideal de equilibrado para un sistema homogéneo (tiempo secuencial dividido por el número de procesadores). Se aprecia claramente cómo algunos procesadores requieren más tiempo de ejecución que otros, y que la *media* coincide aproximadamente con el valor ideal. Este desequilibrio era previsible a partir de las gráficas y conclusiones anteriores. Con el esquema de distribución de carga propuesto basado en el modelo analítico de tiempo de ejecución paralela, los procesadores de índice inferior han sido sobrecargados respecto a los de índice superior. Ello da lugar a concluir que el modelo analítico en el que nos hemos basado no coincide siempre y bajo cualquier circunstancia con el real de ejecución.

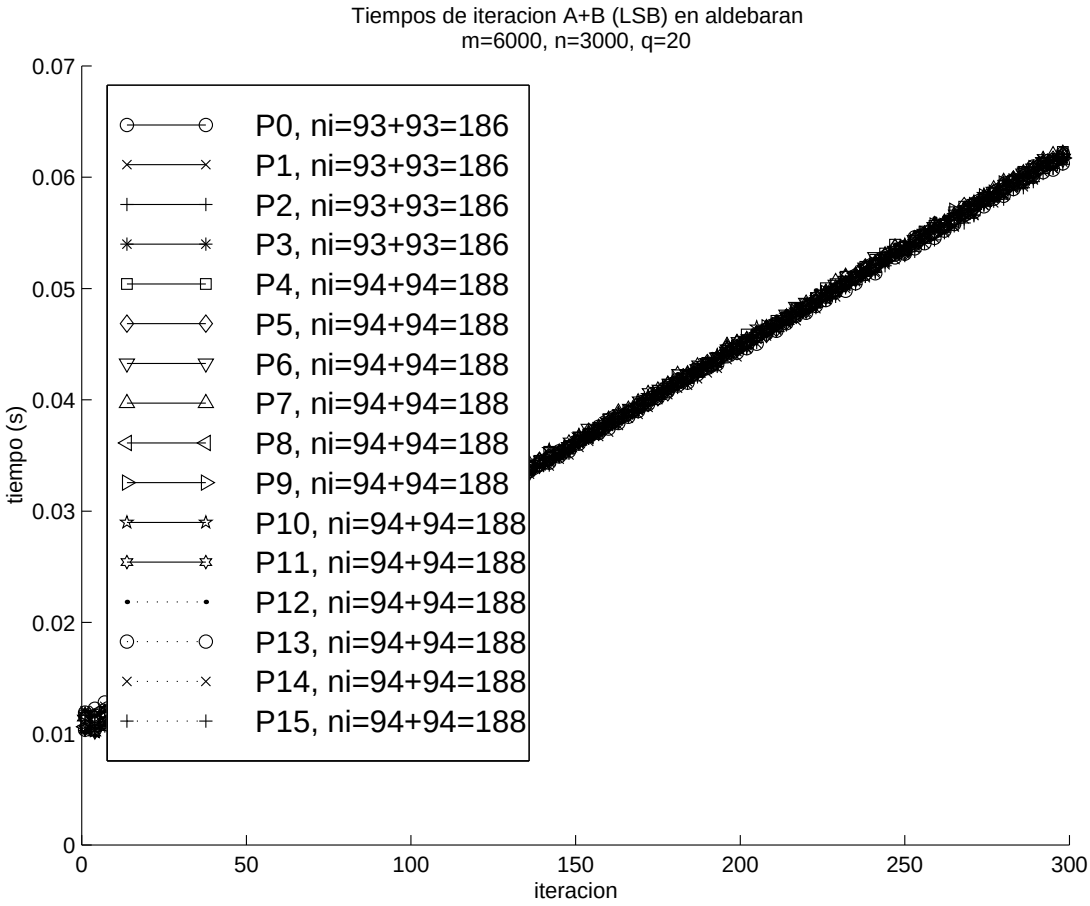

Figura 8.19: Equilibrado de carga en la línea segmentada bidireccional.

Equilibrado de la carga en línea segmentada bidireccional

En las figuras 8.19, 8.20 y 8.21 observamos los resultados para este esquema de distribución de carga, apreciando semejanzas y diferencias respecto al esquema unidireccional. En la figura 8.19 puede observarse un resultado perfecto en el que todos los procesadores tardan lo mismo en cada iteración. En la figura 8.20 se observa el desequilibrio ocasionado por el reparto no equitativo de columnas entre los procesadores, y en la 8.21, la influencia del efecto del tamaño creciente de los datos en el tiempo empleado por iteración.

En las figuras 8.22 y 8.23 se muestran los tiempos aritméticos en cada procesador observando cómo el equilibrado ha resultado notablemente superior al caso unidireccional y cómo la sobrecarga aritmética por paralelización ha sido prácticamente nula (en la figura 8.23 se observa la supuesta anomalía del algoritmo secuencial —dividido por el

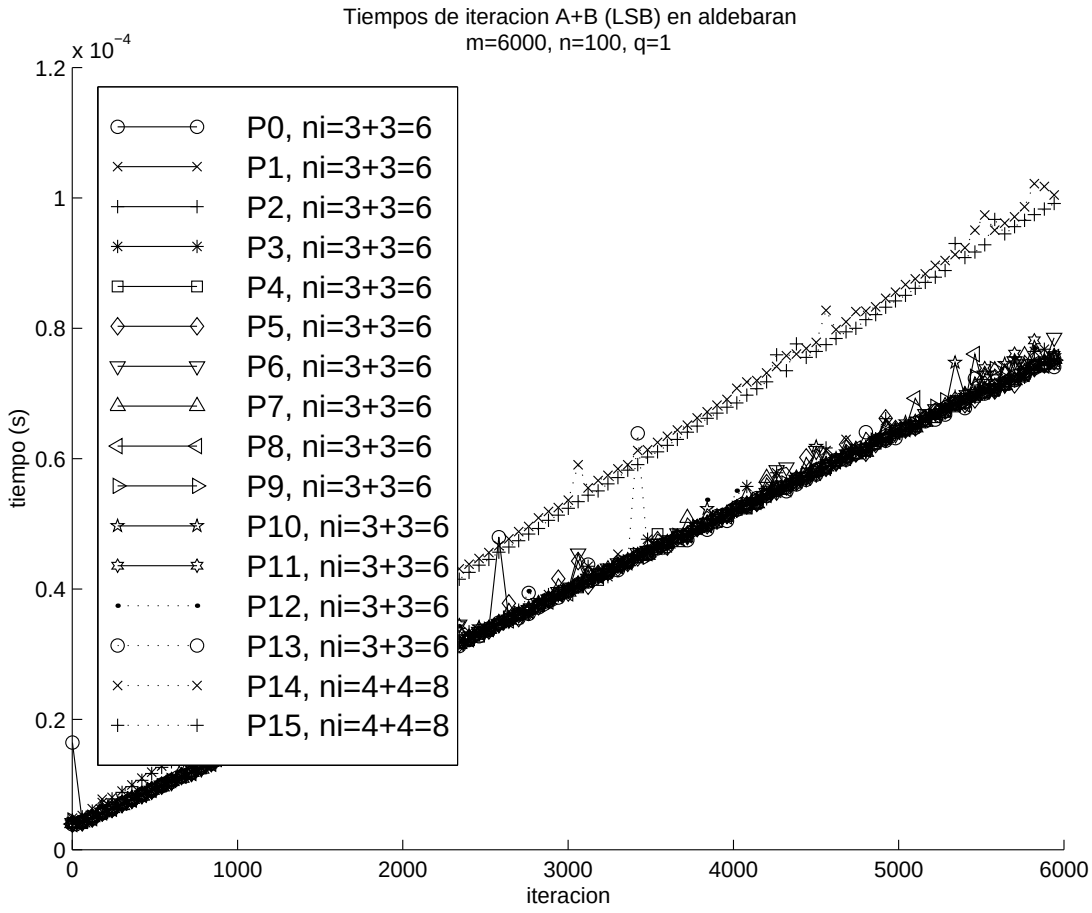

Figura 8.20: Desequilibrio por reparto no equitativo de la carga.

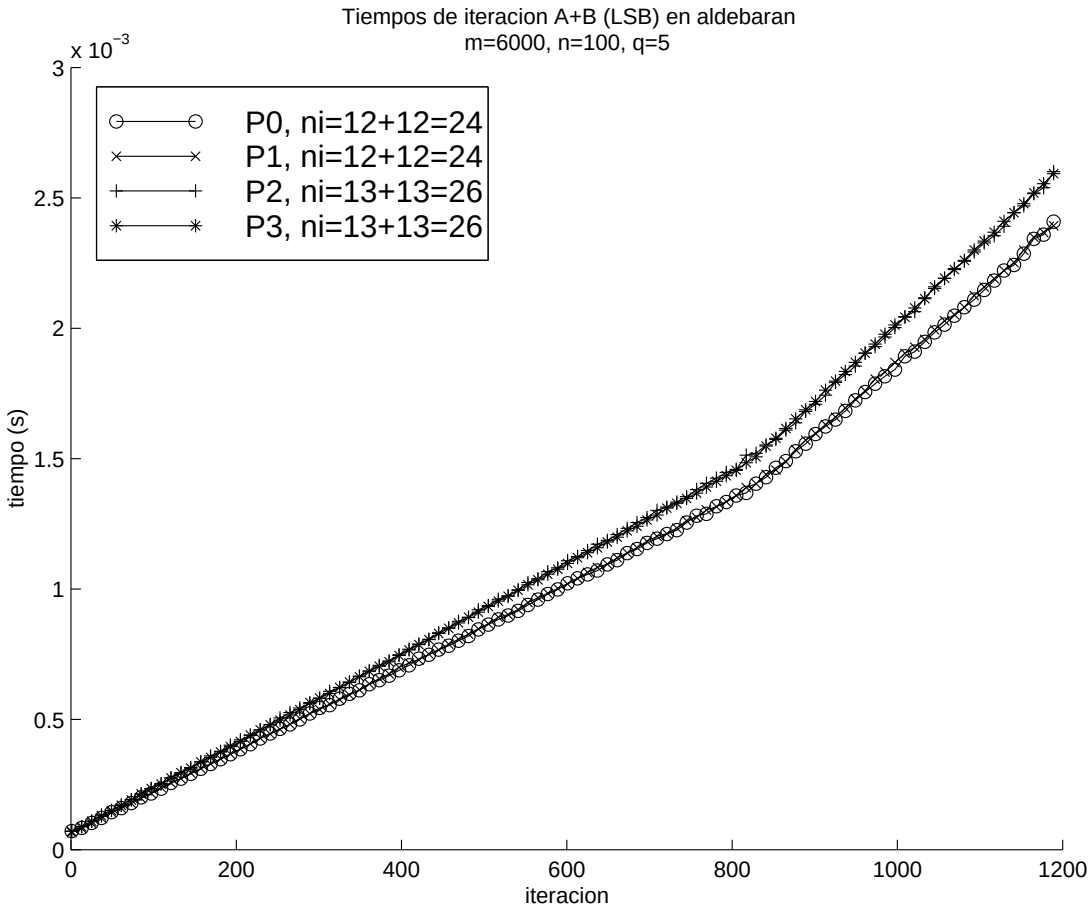

Figura 8.21: Variación de la velocidad de proceso con el tamaño creciente de los datos.

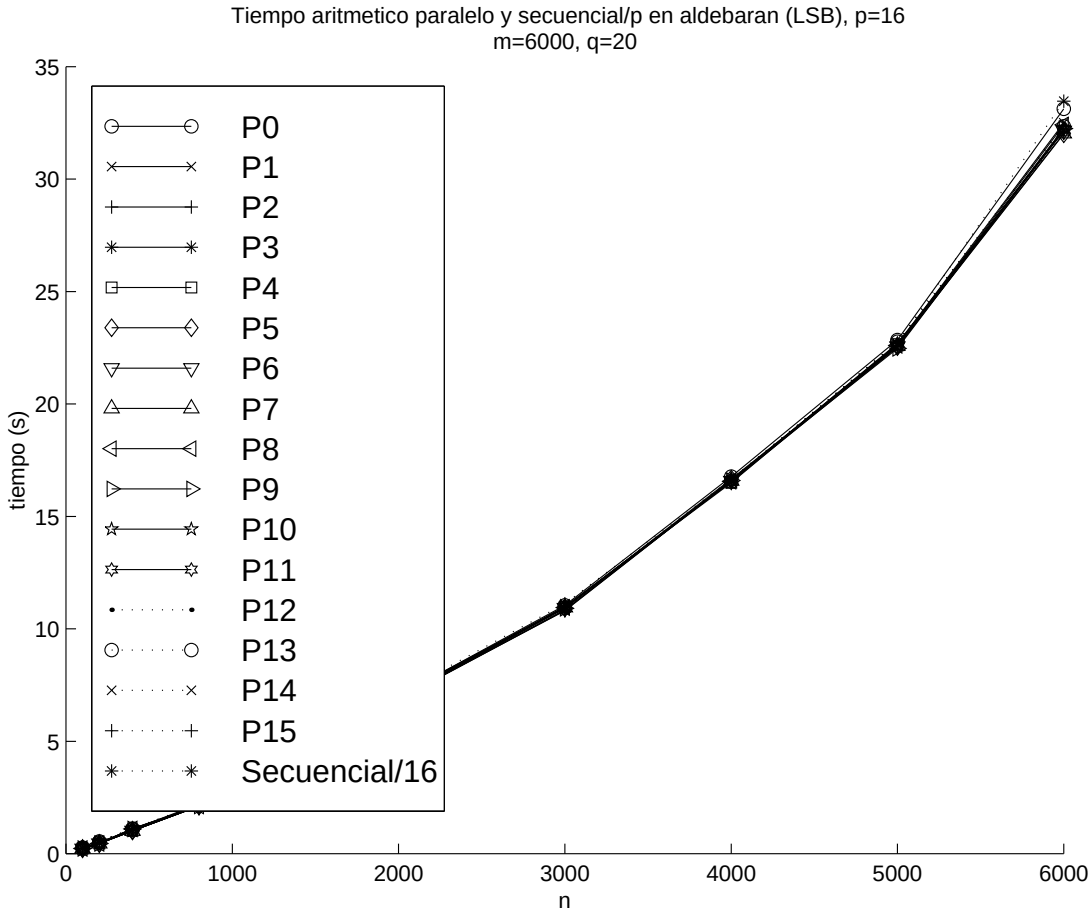

Figura 8.22: Tiempos aritméticos paralelos y secuencial dividido por el número de procesadores en la línea segmentada bidireccional.

número de procesadores— en comparación con el aritmético paralelo).

Tiempo de llenado y vaciado de la línea segmentada

Los procesadores de la línea segmentada entrarán a trabajar paulatinamente conforme vayan transcurriendo las primeras $p-1$ iteraciones del algoritmo ($2p-1$ en el caso bidireccional); de la misma forma, dejarán de trabajar en las últimas $p-1$ iteraciones ($2p-1$ en el caso bidireccional). Los tiempos de estos transitorios son los denominados tiempo de llenado y vaciado de la línea segmentada respectivamente. Esta cantidad de tiempo degrada en cierta medida la eficiencia del algoritmo paralelo. Para que su efecto sea mínimo es condición sine qua non que la cantidad de iteraciones sea notablemente superior a la cantidad de procesadores. Los tiempos de llenado y vaciado de la línea segmentada están

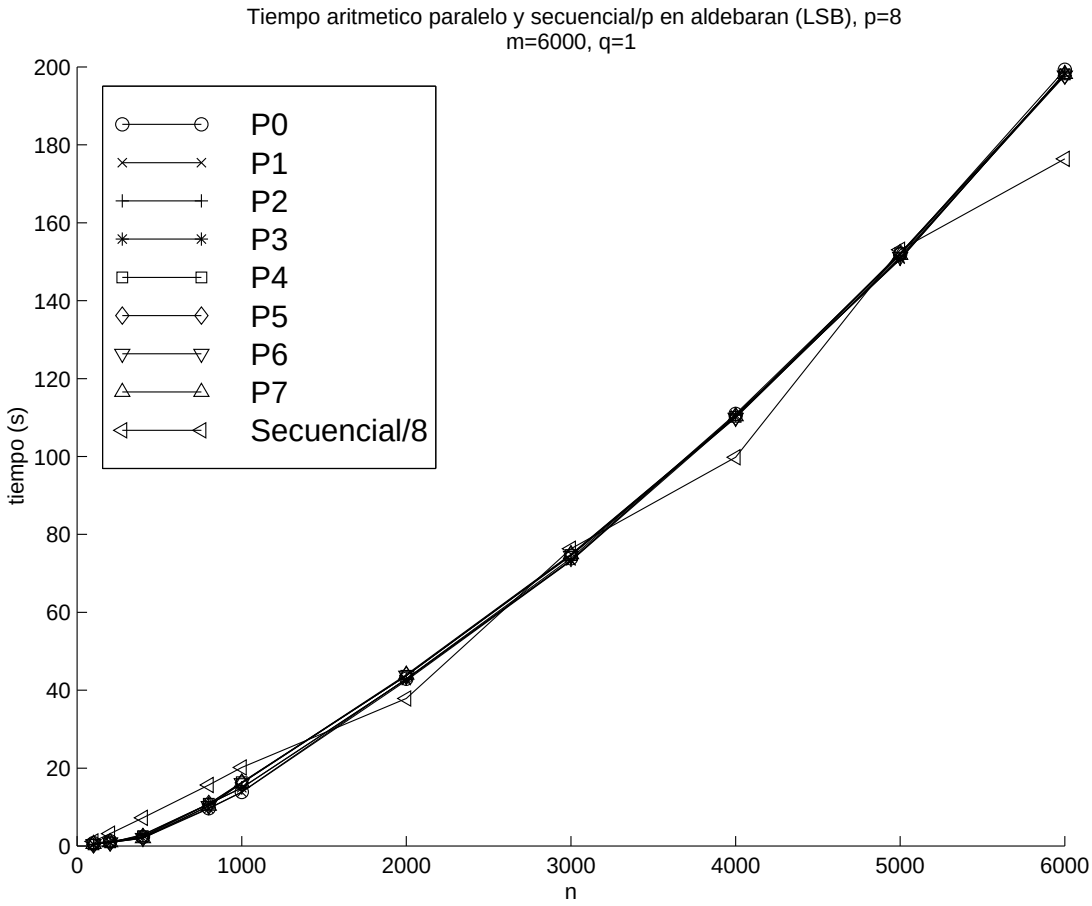

Figura 8.23: Tiempos aritméticos paralelos y secuencial dividido por el número de procesadores en la línea segmentada bidireccional (contraste con anomalía secuencial).

relacionados con los tiempos de espera inicial y final de ciertos procesadores, tal y como se comenta a continuación.

En la línea segmentada unidireccional, P_0 tiene un tiempo de espera inicial nulo y final máximo, mientras que P_{p-1} lo tiene máximo y nulo respectivamente, tal y como muestran las figuras 8.24 y 8.25. El tiempo de espera inicial de un procesador P_j es el de P_{j-1} más el tiempo de cálculo de éste más el de comunicaciones hacia P_j. Los motivos por los que observamos cada vez una menor diferencia en el tiempo de espera inicial entre procesadores consecutivos de índice creciente es que el sistema está equilibrado teóricamente sólo para la última iteración $i = m/q-1$ y este desequilibrio es el que implica esa diferencia en los incrementos, tal y como puede corroborarse observando la figura 8.14. El tiempo de espera final de un procesador P_j es el de P_{j+1} más el tiempo de cálculo de éste junto con el de comunicaciones hacia P_{j+2}. Los incrementos en el tiempo de espera final aparecen constantes entre procesadores consecutivos debido a que existe equilibrio de carga idealmente óptimo.

En la figura 8.26 se muestran los tiempos que debe esperar cada procesador en la línea segmentada bidireccional para arrancar su proceso particular. Podemos observar cómo evidentemente el procesador P_0 tiene un tiempo de espera nulo, el procesador P_1 debe esperar a que P_0 procese sus datos y se los envíe y así sucesivamente. Obviamente P_j debe esperar para arrancar, el tiempo de cálculo de P_{j-1}, el de transmisión de éste hacia P_j más el propio tiempo de espera de P_{j-1}. Los incrementos en los tiempos de espera inicial van siendo sutilmente menores conforme mayor es el índice del procesador; el motivo es el desequilibrio existente entre los procesadores por el esquema de equilibrado especular, ya que este tiempo de espera inicial hace referencia a la espera en el canal de *ida*; sólo cuando el canal de *ida* y de *vuelta* estén ocupados el sistema estará equilibrado.

En la figura 8.27 se muestran los tiempos que debe esperar cada procesador hasta que acaba el algoritmo paralelo. Tal y como ocurría en el caso anterior y dada la disposición especular de los procesadores, el procesador P_0 es el último que finaliza y por tanto no debe esperar nada, P_1 deberá esperar el tiempo de transmitir sus resultados a P_0 y el de

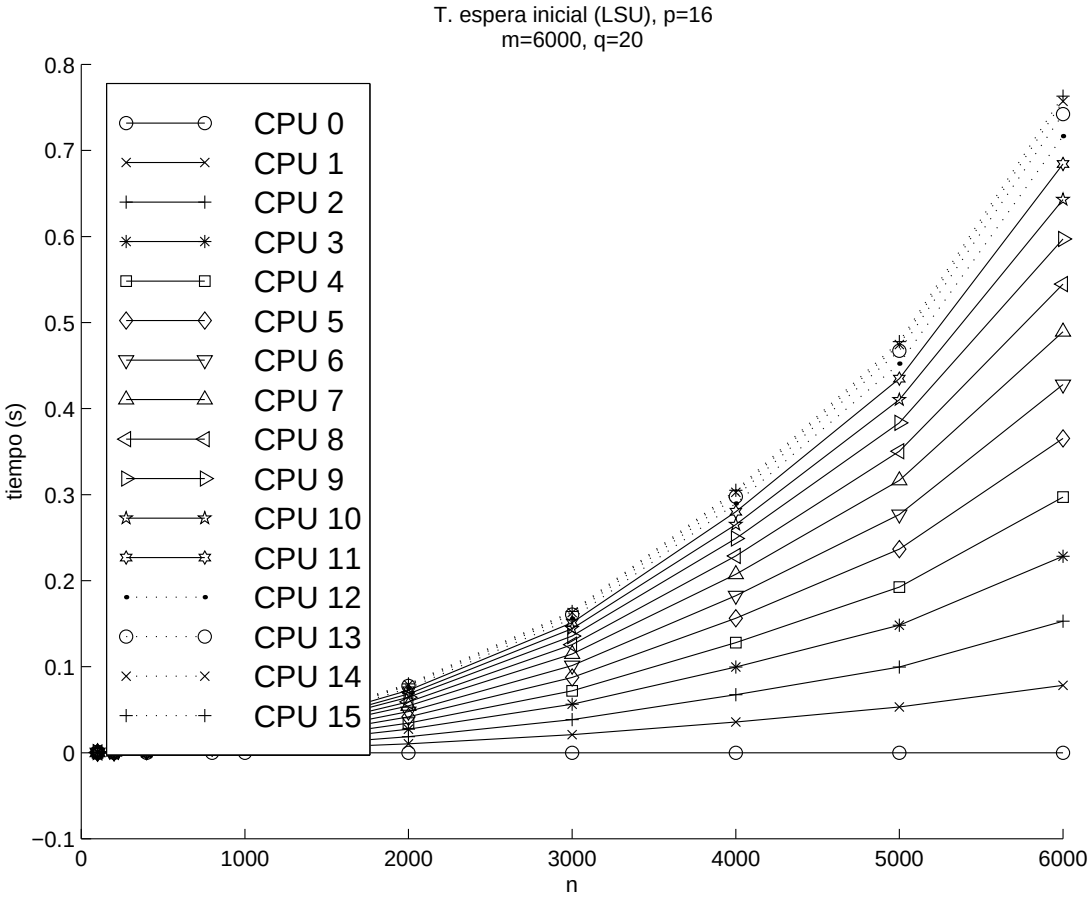

Figura 8.24: Tiempos de espera inicial en la línea segmentada unidireccional.

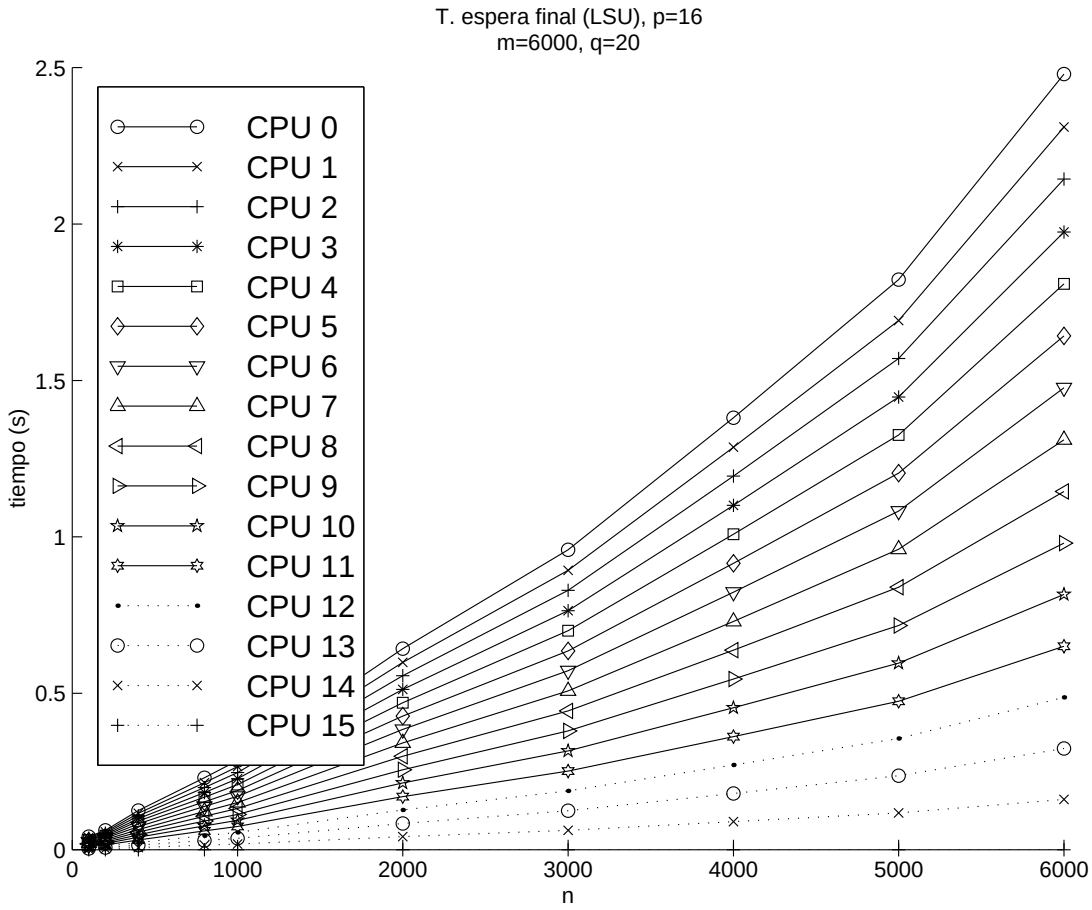

Figura 8.25: Tiempos de espera final en la línea segmentada unidireccional.

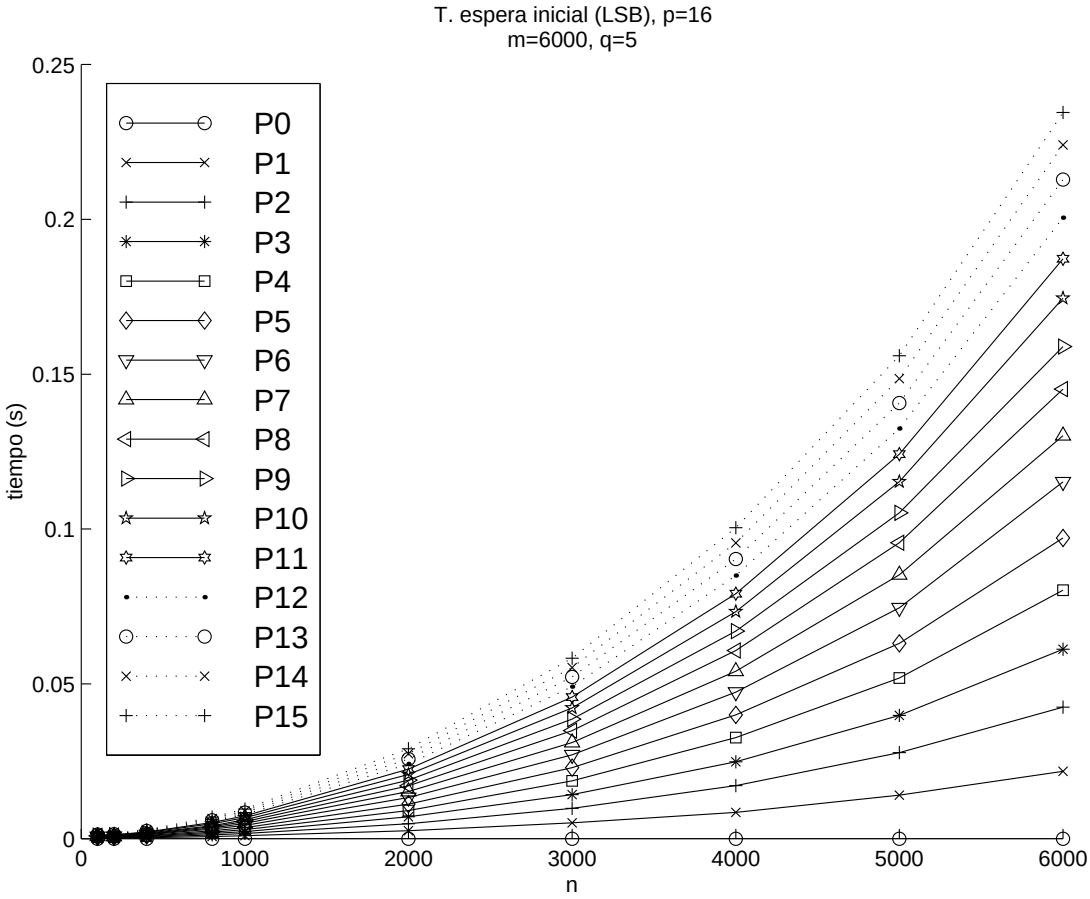

Figura 8.26: Tiempos de espera inicial en la línea segmentada bidireccional.

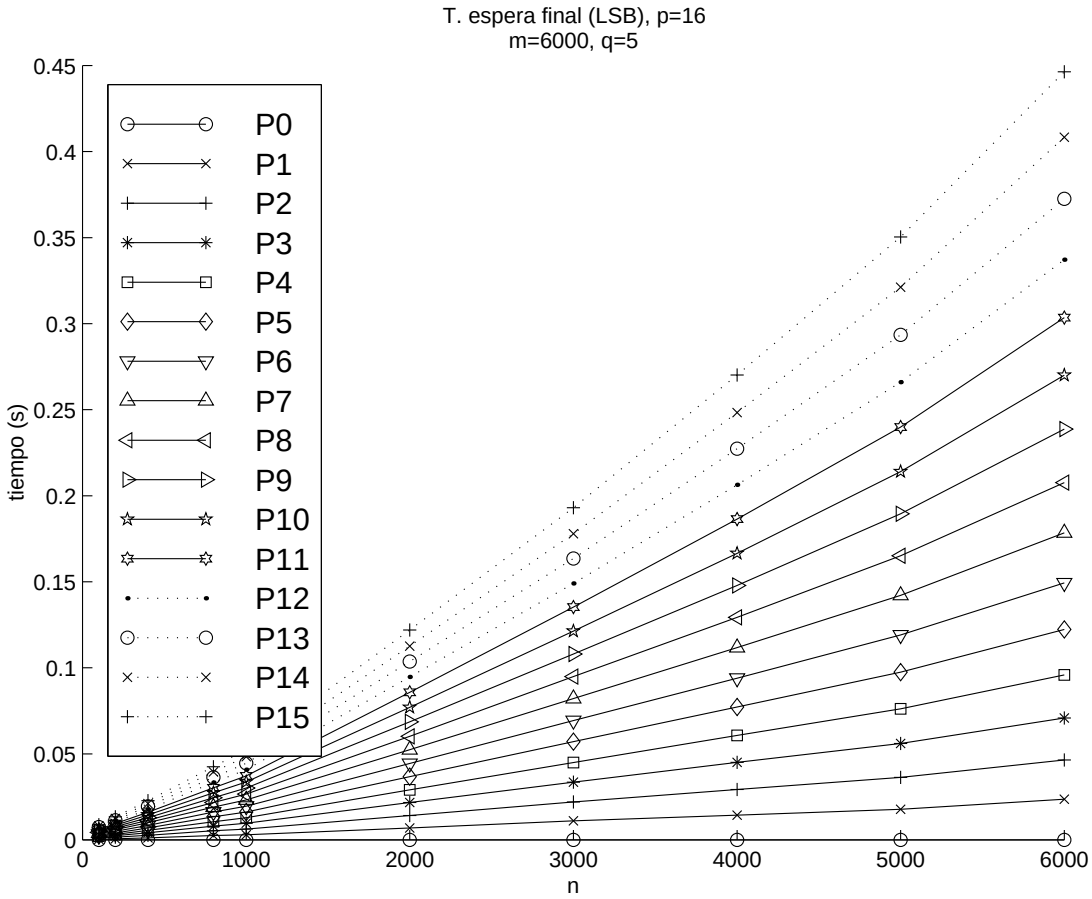

Figura 8.27: Tiempos de espera final en la línea segmentada bidireccional.

cálculo de éste, P_2 el tiempo de transmisión y de cálculo de P_1 más su tiempo de espera y así sucesivamente. Ahora no se aprecia diferencia incremental en el tiempo de espera entre procesadores consecutivos tal y como ocurría con los tiempos de espera inicial, a pesar de que el sistema está desequilibrado pues el canal de *ida* ya está vacío. El motivo es que la diferencia relativa entre la carga de los procesadores no es tan alta como en el caso de la espera inicial, ya que estamos en la iteración $i = m/q - 1$ y la submatriz $\mathbf{\Gamma}_{i+1}$ está llena.

Los tiempos son inferiores en el caso bidireccional debido a que, para las mismas condiciones, existen el doble de particiones pero con la mitad de columnas cada una de ellas, por lo que el tiempo de cálculo de una iteración es inferior, lo cual incide en el tiempo de espera.

En cualquier caso observamos que los tiempos de llenado y vaciado de la línea segmentada son despreciables frente a los tiempos de ejecución, por lo que su influencia la

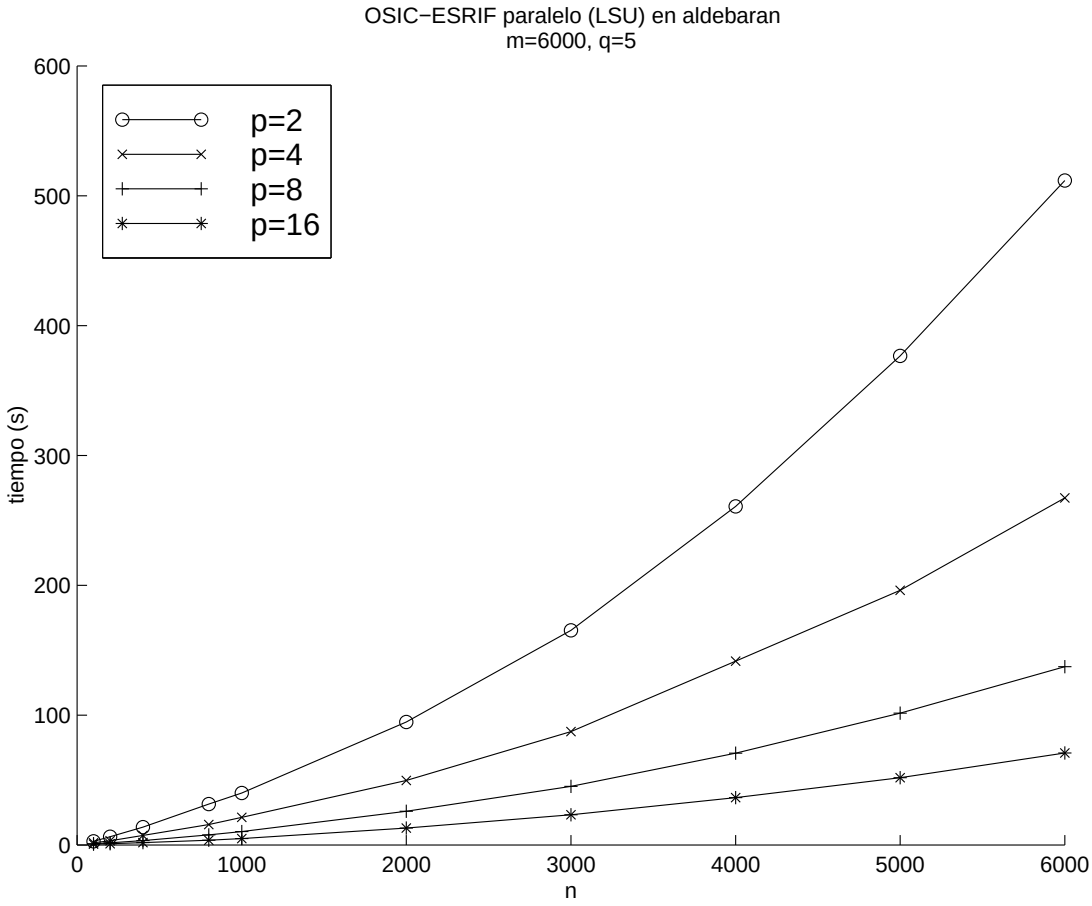

Figura 8.28: Tiempos de ejecución en paralelo en la línea segmentada unidireccional

podemos considerar despreciable.

Tiempo de ejecución en paralelo

En la figura 8.28 y 8.29 se muestran los tiempos de ejecución del algoritmo paralelo en la línea segmentada unidireccional y bidireccional respectivamente.

Eficiencia

En las figuras 8.30, 8.31, y 8.32 se muestran las eficiencias del algoritmo paralelo tanto para la línea segmentada unidireccional y bidireccional respectivamente para distintos valores del parámetro q. Podemos observar una diferencia en el comportamiento de la eficiencia: para $q = 1$ y $q = 5$, la línea segmentada bidireccional tiene una mejor eficiencia, que se invierte para $q = 20$.

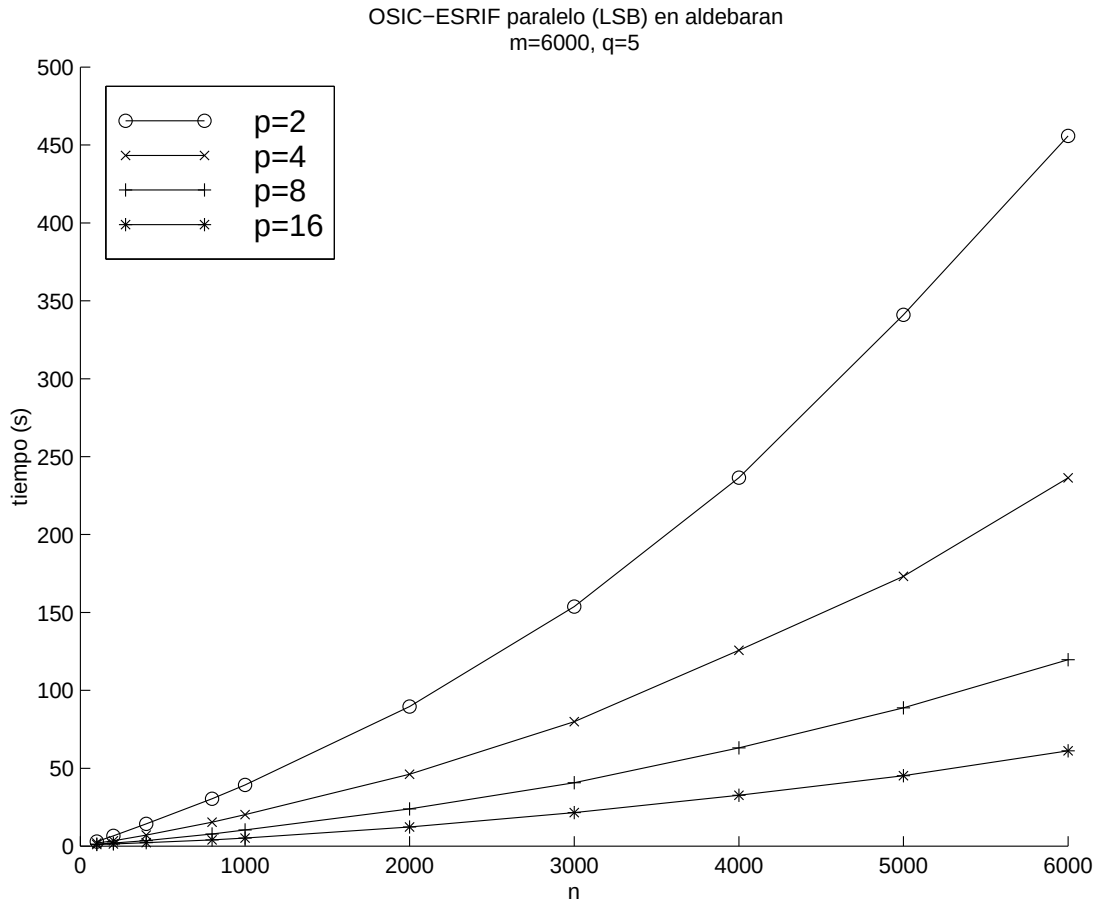

Figura 8.29: Tiempos de ejecución en paralelo en la línea segmentada bidireccional

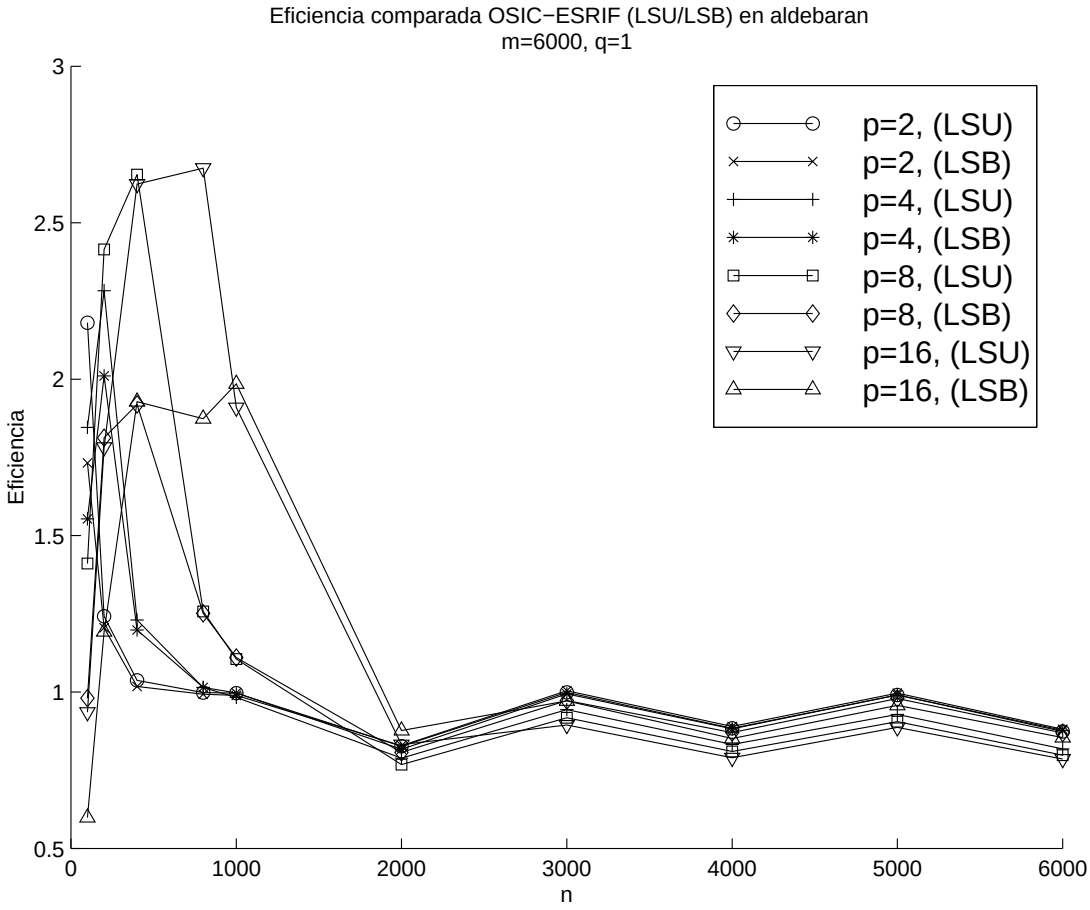

Figura 8.30: Eficiencia del algoritmo OSIC-SRIF en línea segmentada unidireccional y bidireccional con $q = 1$

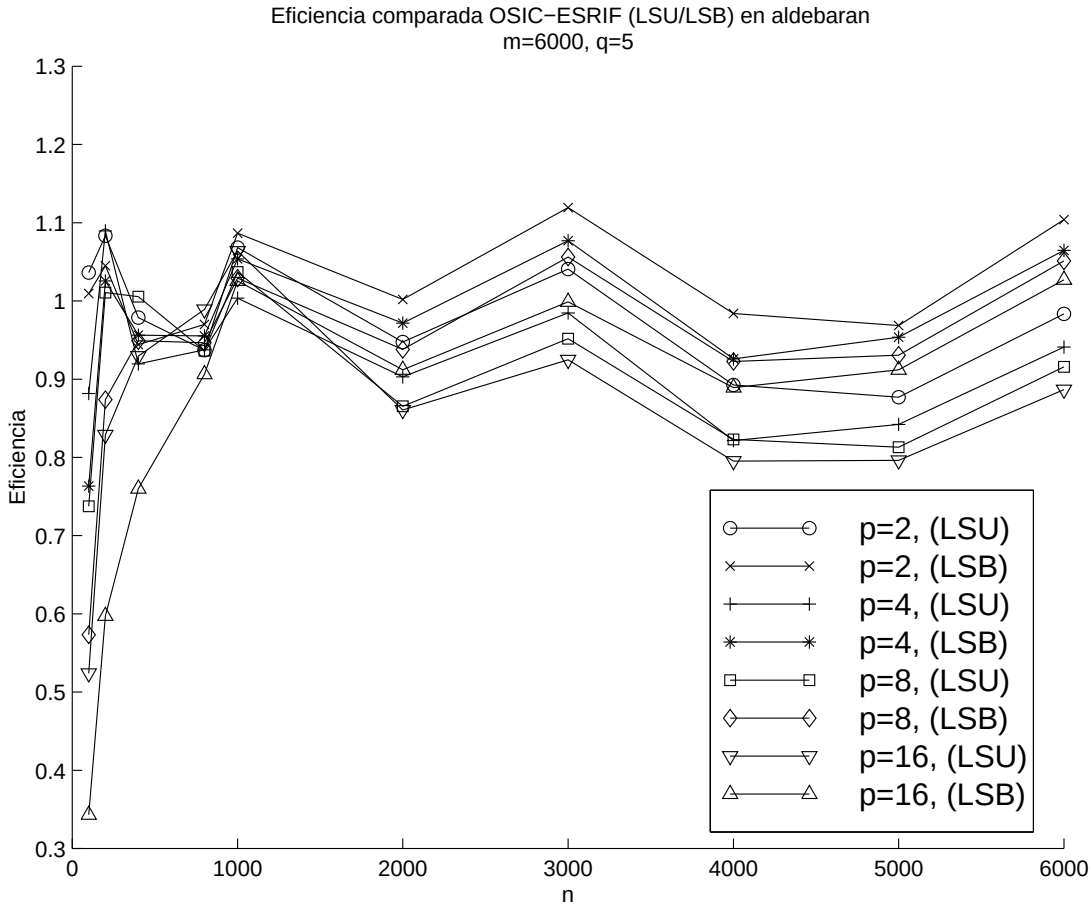

Figura 8.31: Eficiencia del algoritmo OSIC-SRIF en línea segmentada unidireccional y bidireccional con $q = 5$

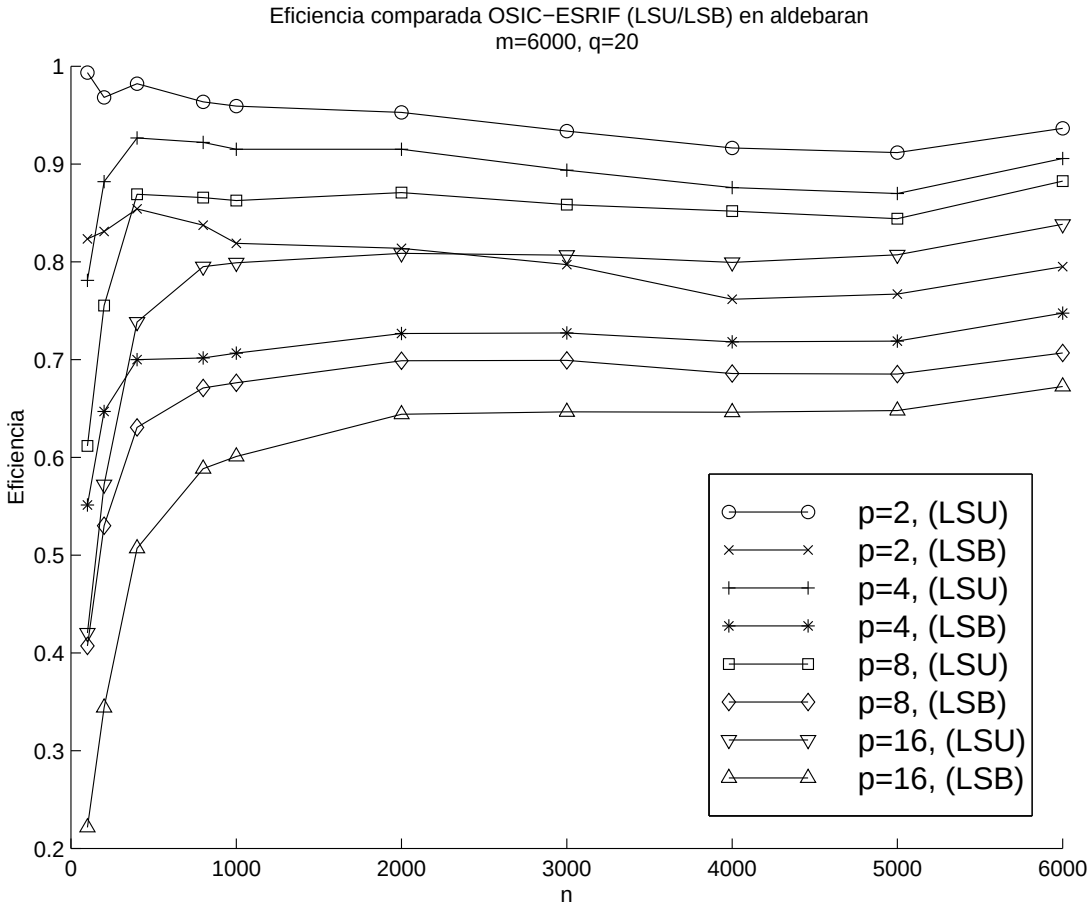

Figura 8.32: Eficiencia del algoritmo OSIC-SRIF en línea segmentada unidireccional y bidireccional con $q = 20$

Para justificar este comportamiento necesitamos conocer, a su vez, el comportamiento del tiempo de sobrecarga. Con ambos esquemas de equilibrado no existe sobrecarga aritmética por paralelización, y sí por conceptos asociados a las comunicaciones y sincronización o esperas.

En la línea segmentada unidireccional el tiempo paralelo está fuertemente influenciado por el tiempo paralelo del primer procesador, ya que es el que más carga sobrelleva, de manera que el resto de procesadores están esperando una cantidad de tiempo apreciable. El tiempo de comunicaciones del primer procesador también prevalece sobre el de los restantes procesadores, ya que es el que más datos transporta. El tiempo de sobrecarga, es decir, la diferencia entre el tiempo paralelo y el tiempo aritmético (del primer procesador), nos va a dar prácticamente el tiempo de comunicaciones del primer procesador, lo cual puede ser observado en las gráficas, teniendo un comportamiento que podemos considerar lineal con n, tal y como era de esperar. En la figura 8.33 se muestra la sobrecarga en este caso para $q = 1$ y en la figura 8.34 para $q = 5$, donde se observa una menor sobrecarga para $q = 5$, lo cual era de esperar a partir de las expresiones (8.30) ó (8.32).

En las figuras 8.36, 8.37 y 8.38 se muestran los tiempos de sobrecarga para el caso bidireccional. Ya que el equilibrado de carga ha sido razonablemente perfecto, el tiempo de sobrecarga es debido exclusivamente a conceptos relativos a comunicaciones y esperas. Las comunicaciones son mayores que en el caso unidireccional, pero siguen siendo lineales con n, por lo que no justifica el comportamiento prácticamente cuadrático del tiempo de sobrecarga para $q = 20$, por lo que el concepto asociado a este comportamiento cuadrático para este valor de q está asociado con la carga. Esto resulta paradójico al haber sido correctamente equilibrada, por lo que nuestra hipótesis más verosímil es que sea debido a que la atención de los dos flujos de información no puede ser simultánea con la implementación realizada, lo cual provoca retardos en las entregas de las informaciones al siguiente procesador en la línea segmentada bidireccional. Estos retardos dependen de la cantidad de tiempo necesaria para procesar el flujo en cuestión y tal y como puede observarse para $q = 20$, este tiempo de sobrecarga es inversamente proporcional al número de procesadores,

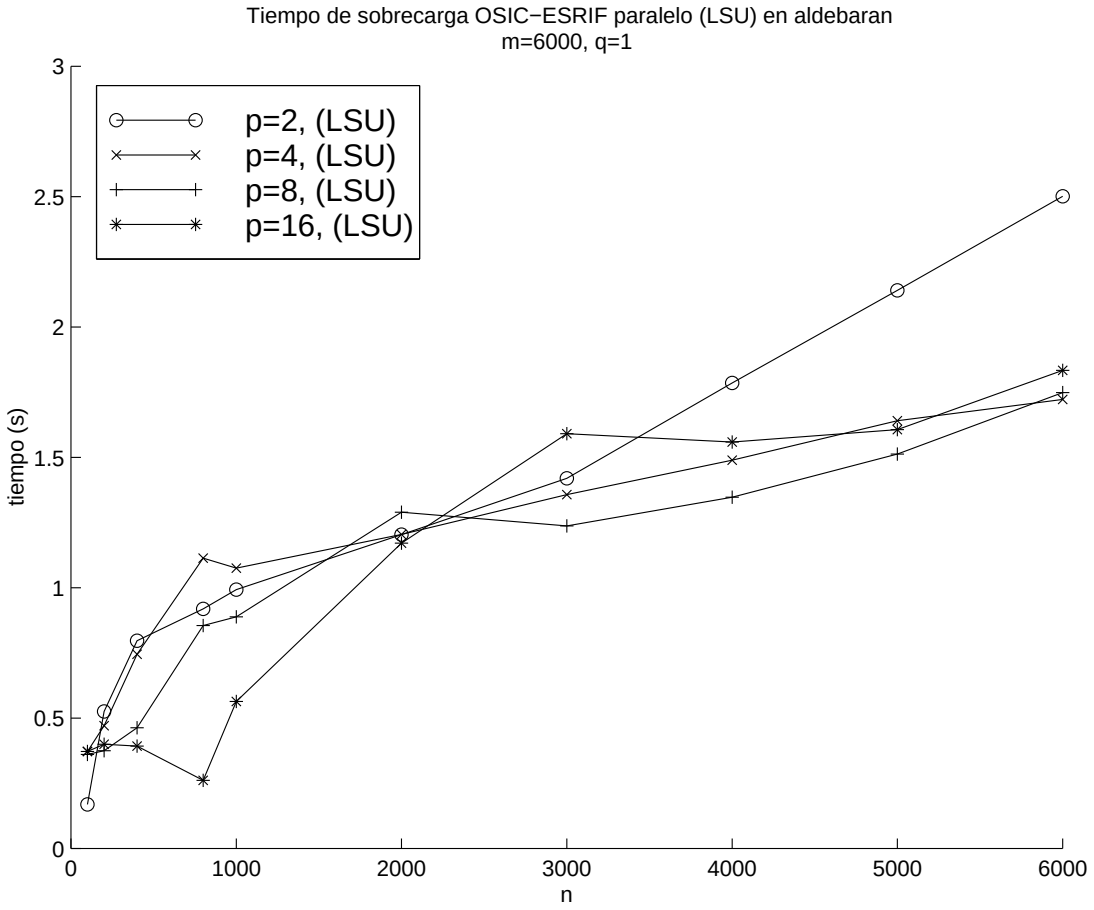

Figura 8.33: Tiempo de sobrecarga del algoritmo OSIC-SRIF en la línea segmentada unidireccional con $q = 1$

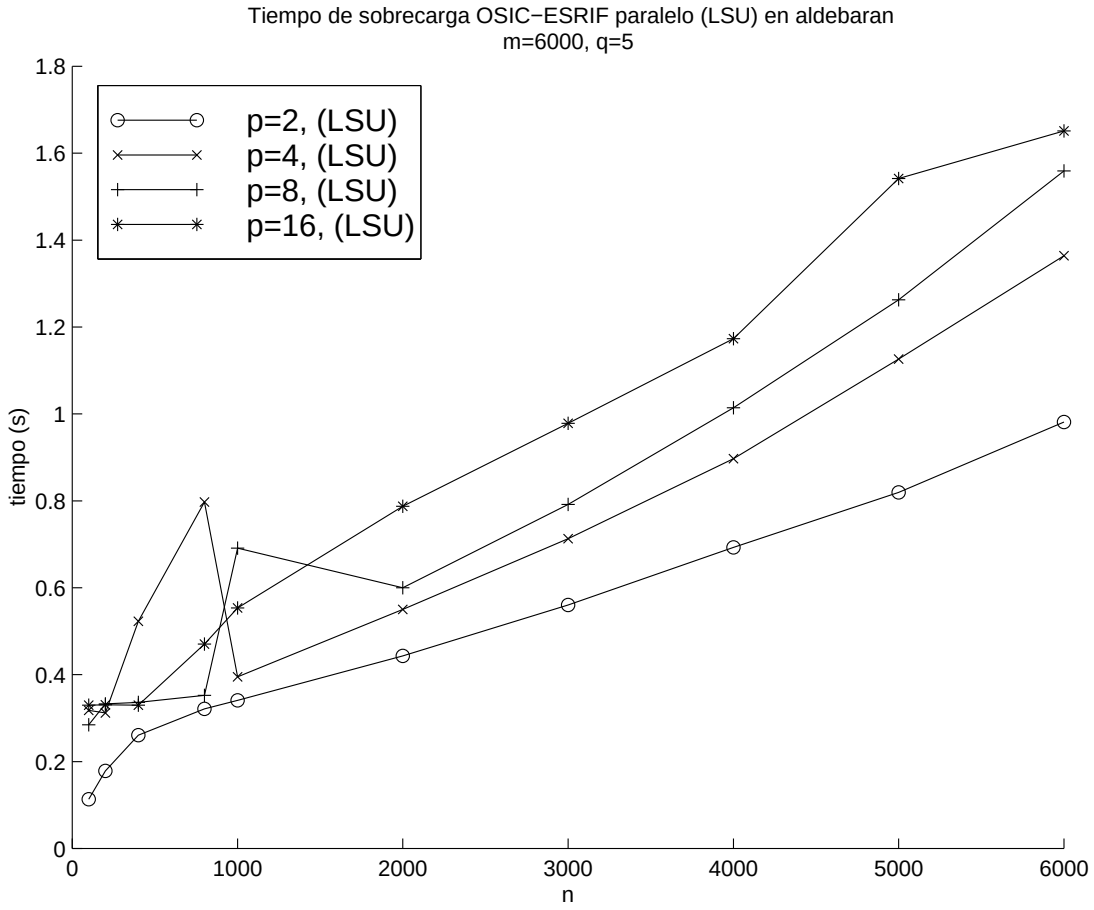

Figura 8.34: Tiempo de sobrecarga del algoritmo OSIC-SRIF en la línea segmentada unidireccional con $q = 5$

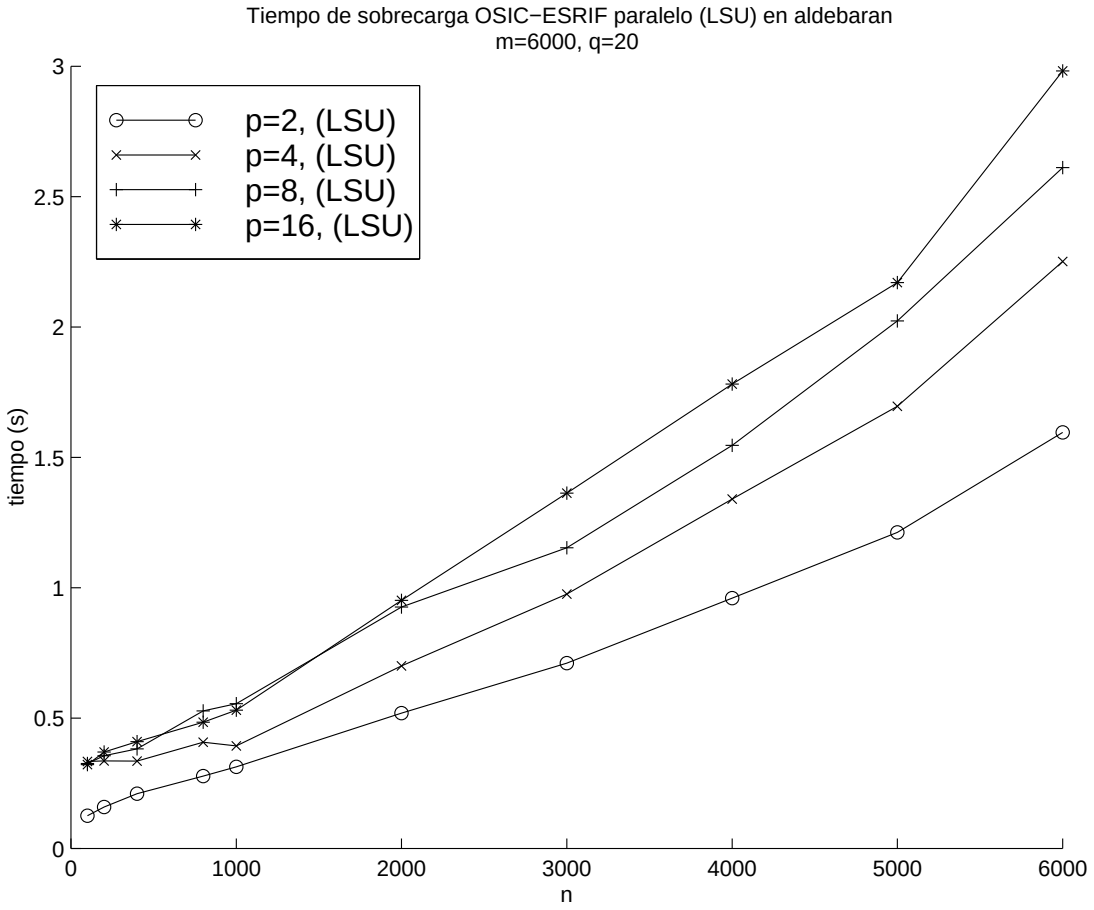

Figura 8.35: Tiempo de sobrecarga del algoritmo OSIC-SRIF en la línea segmentada unidireccional con $q = 20$

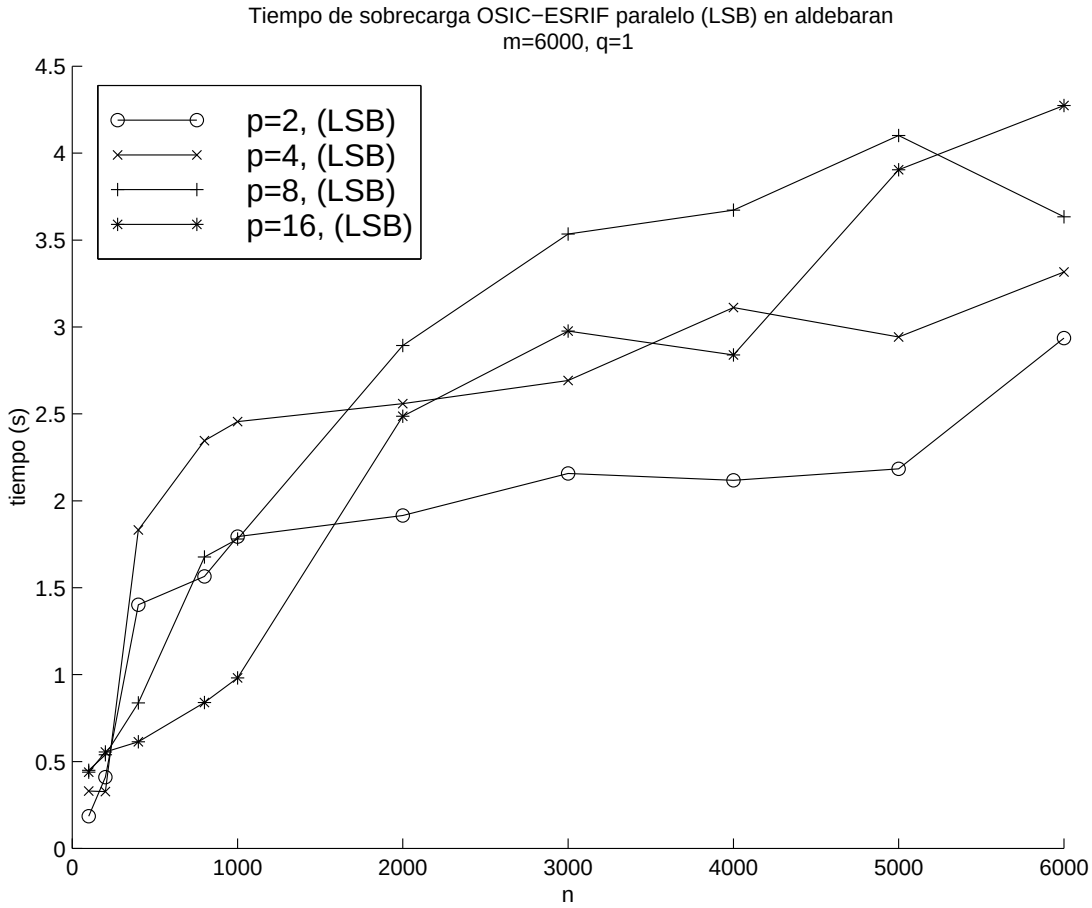

Figura 8.36: Tiempo de sobrecarga del algoritmo OSIC-SRIF en la línea segmentada bidireccional con $q = 1$

lo cual implica que es debido a cuestiones aritméticas y no a comunicaciones.

Caso heterogéneo

Hemos emulado un sistema heterogéneo para el caso de línea segmentada unidireccional utilizando la versión de Givens exclusivamente (SRIF-OSIC-G) bajo las mismas condiciones que las establecidas para el algoritmo SRIF-RLS, descritas en el subapartado 5.2.8, donde mostramos que la máxima eficiencia conseguible en dicho sistema heterogéneo era del 75 %. La figura 8.39 muestra los resultados de la eficiencia para un valor de $q = 5$. Aquí observamos cómo dicha eficiencia ronda el límite establecido teóricamente.

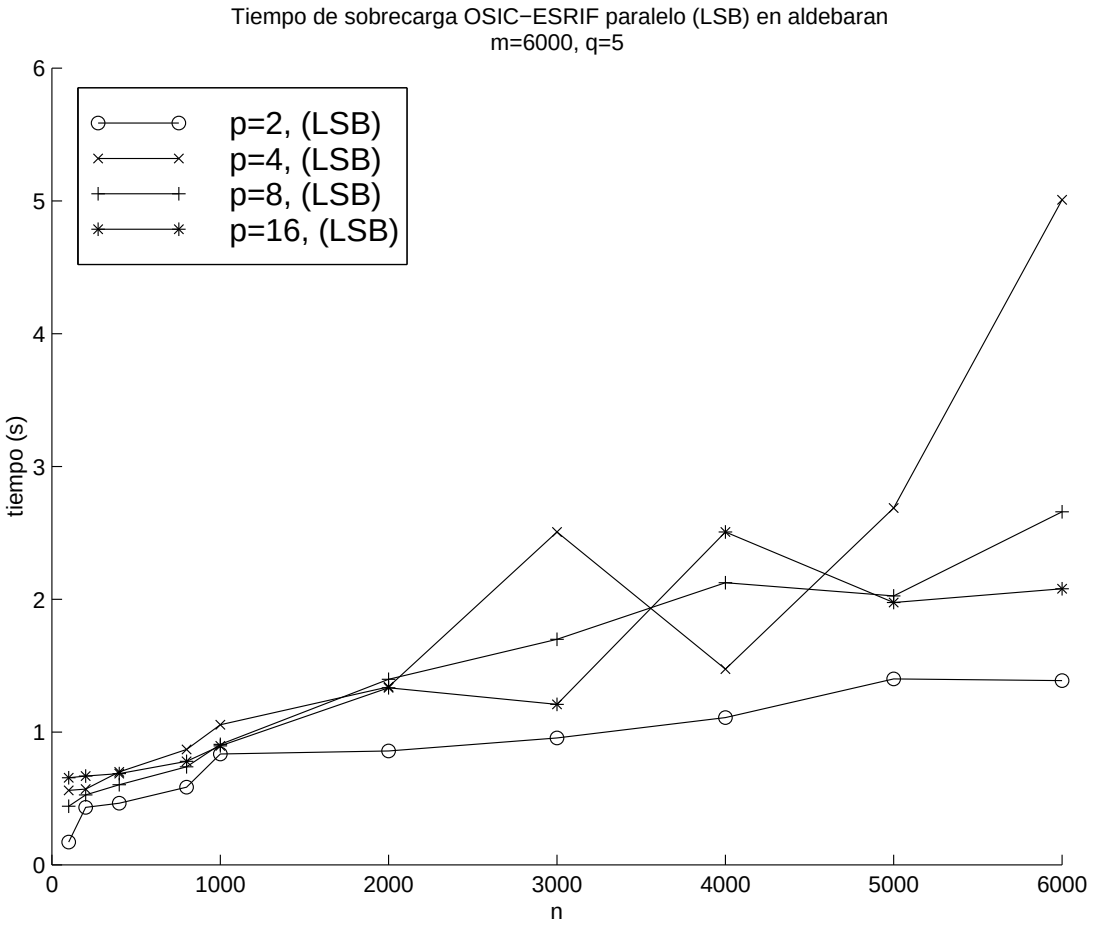

Figura 8.37: Tiempo de sobrecarga del algoritmo OSIC-SRIF en la línea segmentada bidireccional con $q = 5$

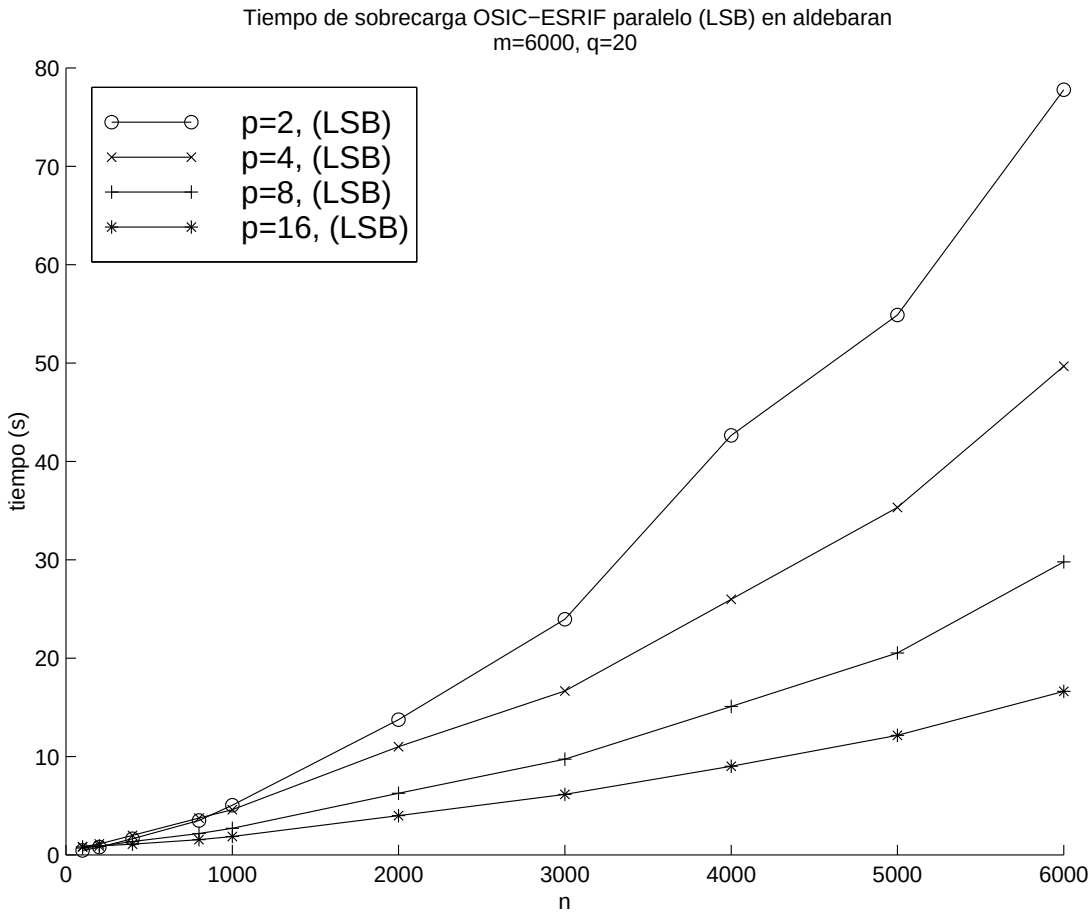

Figura 8.38: Tiempo de sobrecarga del algoritmo OSIC-SRIF en la línea segmentada bidireccional con $q = 20$

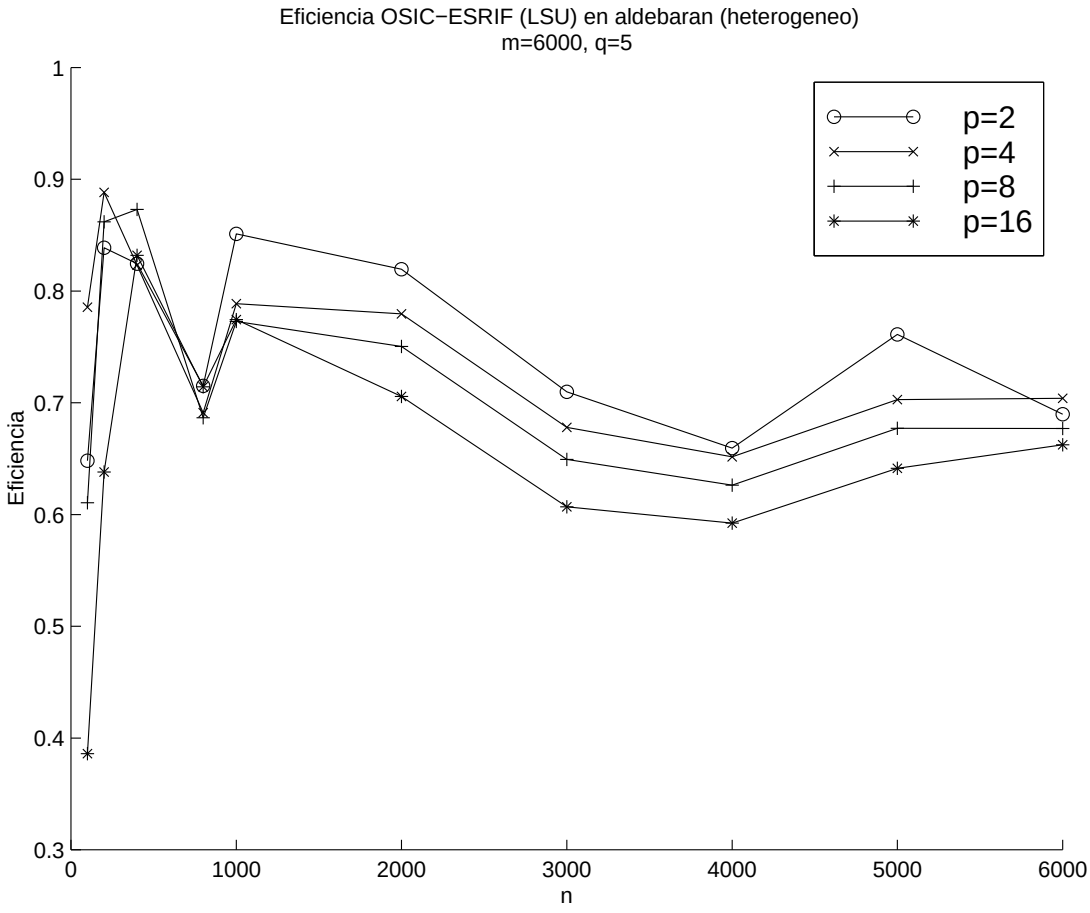

Figura 8.39: Eficiencia para el sistema heterogéneo con $q = 5$.

8.4.10. Conclusiones y mejoras

Hemos observado un excelente comportamiento del algoritmo paralelo, tanto en su versión de línea segmentada unidireccional como bidireccional. En términos generales, las prestaciones para la versión bidireccional son mejores que para el caso unidireccional debido a un notablemente mejor equilibrado de la carga.

Como posibles mejores podemos citar la reconfiguración del algoritmo de equilibrado para la línea segmentada unidireccional, la mejora de la eficiencia para valores altos de q, así como el planteamiento de un esquema adaptativo de equilibrado de carga, tanto para el caso unidireccional como bidireccional.

8.5 Comparación y conclusiones

A continuación comparamos los resultados obtenidos para los métodos SRKF-OSIC y SRIF-OSIC, [111] (en [110] se hace hincapié en la comparación con una situación heterogénea). En la figura 8.40 se muestra el ratio entre los tiempos de ejecución de la versión SRIF-OSIC-HG y SRKF-OSIC, donde podemos comprobar que el ratio depende del valor de q, tal y como cabía esperar a partir del planteamiento téorico de los costes.

La figura 8.41 se muestra el ratio entre el algoritmo OSIC-SRKF y el OSIC-SRIF-G, pero esta vez empleando exclusivamente rotaciones de Givens, observando una mejora del orden del 20 %. La mejora del método OSIC-SRIF-G sobre el OSIC-SRKF, de forma prácticamente independientemente del valor de q fue ya prevista teóricamente en el subapartado 8.4.3

Respecto a las paralelizaciones, los resultados de eficiencia de ambos algoritmos son similares (figuras 8.7 y 8.32). Si, en términos generales, el algoritmo secuencial OSIC basado en SRIF ha obtenido mejores prestaciones, sobre todo para la versión de Givens independientemente del valor del parámetro q, esta conclusión es extrapolable a las versiones paralelas.

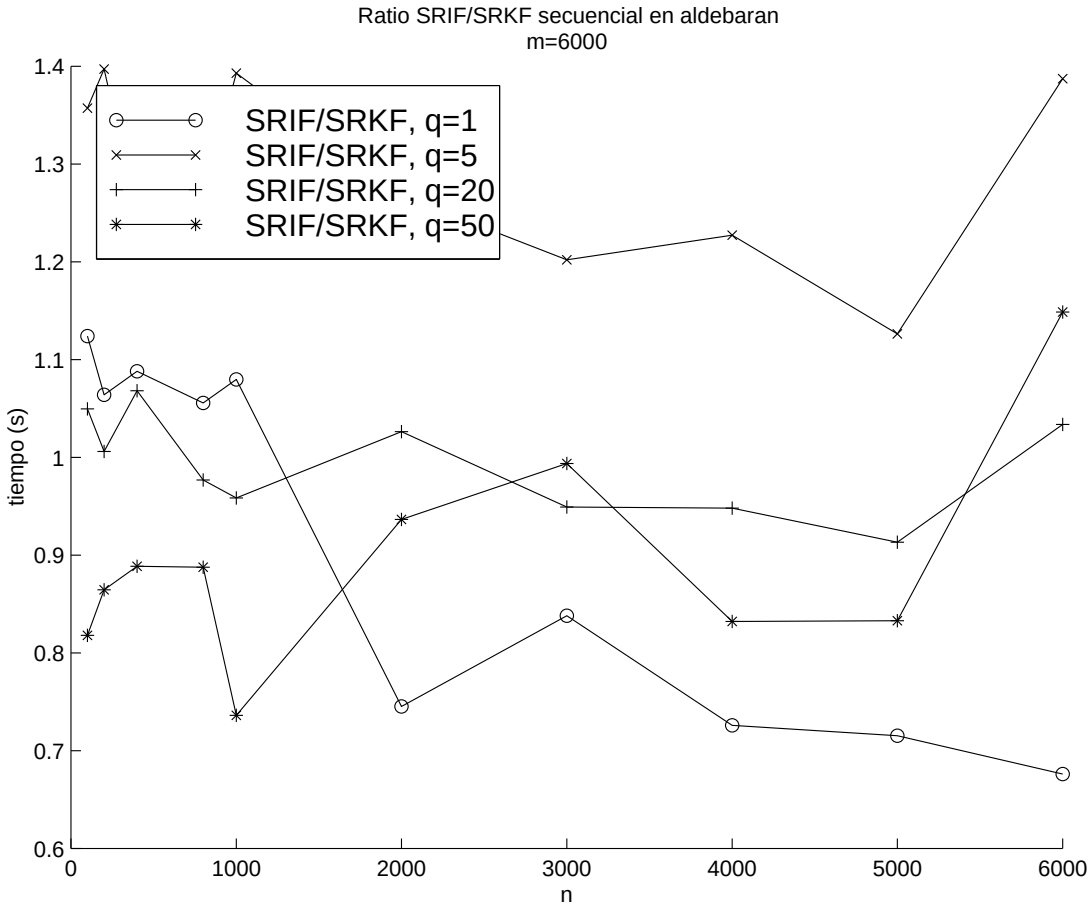

Figura 8.40: Ratio entre los tiempos de ejecución secuenciales de los algoritmos OSIC-SRIF y OSIC-SRKF

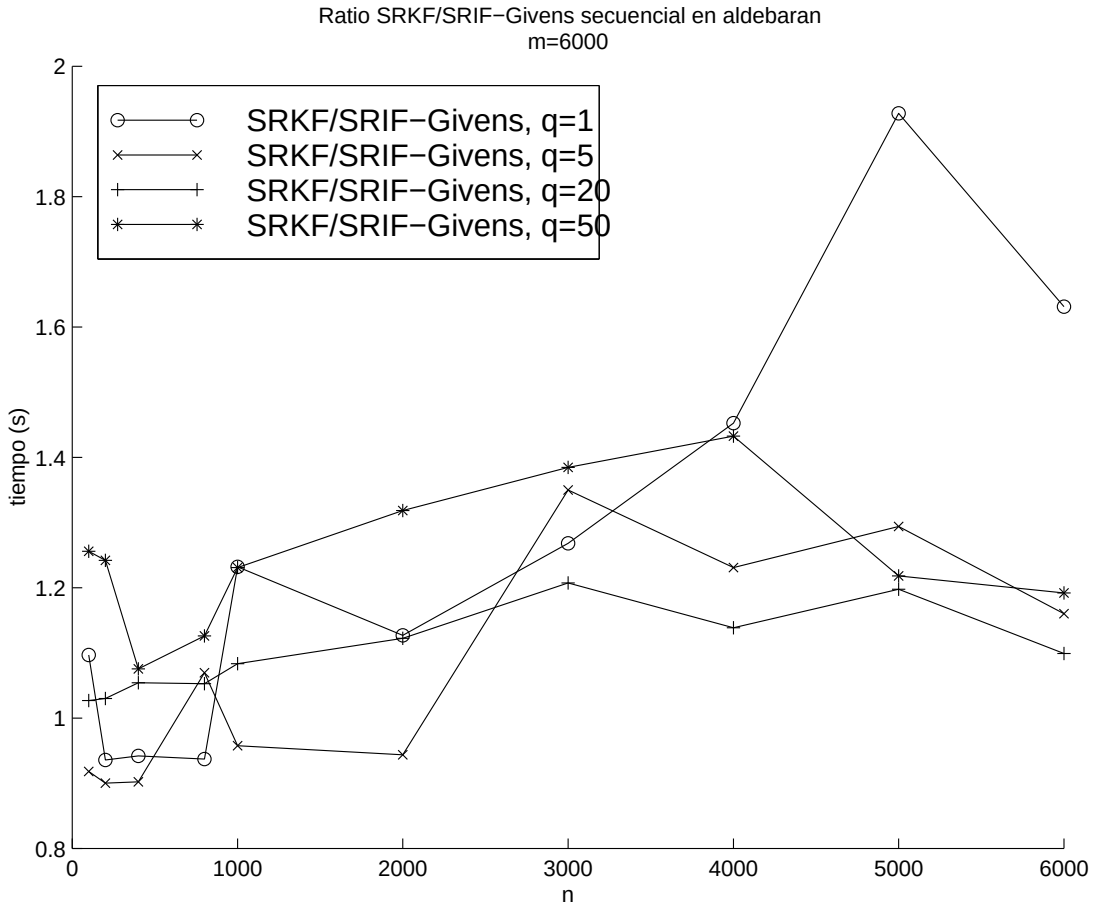

Figura 8.41: Ratio entre los tiempos de ejecución secuenciales de los algoritmos OSIC-SRKF y versión Givens de OSIC-SRIF

9 Conclusiones y líneas futuras

9.1 Conclusiones

Hemos realizado la paralelización con una alta eficiencia de tres algoritmos que resuelven el problema RLS, dos de ellos en dos versiones: línea segmentada unidireccional y línea segmentada bidireccional. En los casos en los que han resultado de interés hemos analizado el comportamiento en una situación heterogénea. El esquema basado en la línea segmentada bidireccional, en términos generales, no ha proporcionado los resultados esperados, supuestamente debido a la gestión secuencial que se realiza en los distintos flujos de información. Una posible mejora que ayudaría a paliar este problema es el emplear *hilos* para gestionar de manera concurrente estos flujos. Esta alternativa nos la planteamos como una posible línea futura de estudio para intentar aprovechar el potencial, en cuanto al equilibrado de la carga, que presenta esta opción. Tal y como era previsible, la solución RLS basada en la actualización de la factorización QR ha sido la más ventajosa en cuanto a tiempo de ejecución. No por ello, abandonamos la línea de estudio de los algoritmos basados en la estimación del estado de un sistema, ya que los algoritmos OSIC se basan en ellos y el conocimiento de sus características y detalles ayudan a perfilar mejoras para dichas versiones OSIC. Además, en una implementación de tipo VLSI el hecho de necesitar

cierta cantidad de memoria para posteriormente recuperar la solución puede ser una clara desventaja del algoritmo basado en la factorización QR respecto a los derivados del filtro de Kalman y del filtro de información.

Hemos aportado una solución novedosa para el problema OSIC con mejores resultados en tiempo de ejecución que la propuesta en la literatura, que aunque no determinista, dicho tiempo está acotado. La nueva solución ha consistido en reformular la solución del problema OSIC basada en el filtro de Kalman, en una solución basada en el filtro de información con un importante ahorro en coste computacional tal y como se ha podido contrastar en la práctica. Esta es una de las aportaciones teóricas y experimentales que consideramos de más interés de esta tesis doctoral. Asimismo se ha llevado a cabo la paralelización de tanto la versión basada en el filtro de Kalman como la basada en el de información, con buenos resultados en cuanto a la eficiencia. Igualmente que en el caso del problema RLS, hemos planteado una alternativa basada en el empleo de líneas segmentadas bidireccionales con resultados similares.

Todos los algoritmos paralelizados en todas sus versiones tienen un valor añadido potencial que consiste en la posibilidad de realizar un equilibrado adaptativo de la carga que permita reajustar en el tiempo el trabajo asignado a cada procesador, para adaptarse a condiciones variantes en el sistema de una manera relativamente sencilla, a costa de emplear cierto tiempo de ejecución, en el cálculo de nueva distribución de datos y en la actualización de la misma. La exigencia adicional para la versión en línea segmentada unidireccional es el establecimiento de un enlace bidireccional entre los procesadores contiguos en la línea segmentada para poder reasignar las columnas/filas que se hallan en la *frontera* de ambos procesadores.

El trabajo recopilado en la presente memoria de la tesis doctoral ha dado lugar a un total de 10 publicaciones, de ellas un informe técnico [94], una ponencia en un congreso nacional [96], cinco en congresos internacionales, [95, 99, 110, 111, 108] y tres artículos en revista [97, 98] y [109], sometido. Asimismo, el trabajo ha estado integrado en dos proyectos CICYT:

- TIC2000-1683-C03: "Desarrollo e implementación de algoritmos de procesado de señal y de computación de altas prestaciones para sistemas de emulación de entornos acústicos virtuales"

- TIC2003-08238-C02: "Sistemas de audio 3D robustos basados en multirresolución espectral y computación de altas prestaciones"

desde el año 2000 hasta el 2006, y plantea su continuidad en futuros proyectos del grupo de investigación.

9.2 Líneas futuras

Como líneas futuras de trabajo y estudio proponemos:

- Implementación en otras arquitecturas paralelas como clusters de PC, implementaciones VLSI (FPGA), multiprocesadores DSP, ...

- Implementación en procesadores *multicore* con cierto énfasis en el estudio de la diferencia de rendimiento entre la perspectiva de memoria compartida y distribuida.

- Análisis del comportamiento de los algoritmos con formato de datos en coma fija, remodelación de los algoritmos para matrices estructuradas (versiones de Chandrasekhar)

- Estudio de versiones OOC (*out of core*) de las versiones más significativas que intenten disminuir el tiempo de ejecución, teniendo en cuenta la posibilidad de almacenamiento de datos en memoria secundaria como consecuencia de un tamaño extremadamente grande de los mismos.

- Extensión de las ideas y conceptos desarrollados a diferentes implementaciones V-BLAST

- Estudio y comparación con métodos métodos de detección SD (*Sphere Decoding*), LLL (Lenstra, Lenstra y Lovász), etc.
- Estudio e implementación de los algoritmos en sistemas reales de comunicaciones (MIMO multiusuario, OFDM, comunicaciones ópticas, ...), audio, ...
- Interacción de los esquemas de detección con esquemas de control de errores para conseguir mejoras en las prestaciones.

Estas líneas futuras de trabajo prevén su realización y continuidad dentro de los futuros proyectos citados anteriormente.

Índice de figuras

Índice de algoritmos

Bibliografía

[1] M. Moonen and E. Deprettere. A fully pipelined RLS-based array for channel equalization. *Journal of VLSI Signal Processing*, 14:67–74, 1996.

[2] M. Moonen. Systolic algorithms for recursive total least squares estimation and mixed RLS/RTLS. *International Journal of High Speed Electronics, special issue on 'Massively Parallel Computing'*, 4(1):55–68, 1993.

[3] M. Moonen and J. Vandewalle. A systolic array for recursive least squares computations. *IEEE Transactions on Signal Processing*, 41(2):906–912, 1993.

[4] M. Moonen and J.G. McWhirter. A systolic array for recursive least squares by inverse updating. *Electronics Letters*, 29(13):1217–1218, 1993.

[5] M. Moonen. Implementing the square-root information Kalman filter on a Jacobi-type systolic array. *Journal of VLSI Signal Processing*, 42(3):1–9, 1994.

[6] I.K. Proudler, J.G. McWhirter, M. Moonen, and G. Hekstra. The formal derivation of a systolic array for recursive least squares estimation. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 43(3):247–254, March 1996.

[7] Ian N. Dunn and Gerard G.L. Meyer. QR factorization for shared memory and message passing. *Parallel Computing*, 28(11):1507–1530, 2002.

[8] I.N. Dunn and G.G.L. Meyer. Parallel compact WY QR factorization for asynchronous message passing. In *21st IEEE International Performance, Computing, and Communications Conference*, April 2002.

[9] I.N. Dunn and G.G.L. Meyer. Parameterized Householder QR factorization algorithms for the Mercury RACE multicomputer. In *Third Annual Workshop on High Performance Computing*, September 1999.

[10] I.N. Dunn and G.G.L. Meyer. Parallel fast Givens QR factorization on the HP/Convex Exemplar. In *10th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 590–595, October 1998.

[11] Jr. James J. Carrig and Gerard G. L. Meyer. Efficient Householder QR factorization for superscalar processors. *ACM Trans. Math. Softw.*, 23(3):362–378, 1997.

[12] Peter Vouras and G.G.L. Meyer. Hybrid QR factorization algorithm for high performance computing architectures. In *Sixth Annual Workshop on High Performance Embedded Computing*, September 2002.

[13] Ali H. Sayed and Thomas Kailath. A state-space approach to adaptive RLS filtering. *IEEE Signal Processing Magazine*, 11(3):18–60, July 1994.

[14] M. Lu, X. Qiao, and G. Chen. A parallel square-root algorithm for modified extended Kalman filter. *IEEE Transactions on Aerospace and Electronic Systems*, 28(1):153–163, 1992.

[15] D. W. Brown and F.M.F. Gaston. The design of parallel square-root covariance Kalman filters using algorithm engineering. *Integr. VLSI J.*, 20(1):101–119, 1995.

[16] L Ljung and T. Soderstrom. *Theory and Practice of Recursive Identification*. MIT Press, Cambridge, MA, USA, 1983.

[17] J. Jiang and Y. Zhang. A revisit to block and recursive least squares for parameter estimation. *Computers & Electrical Engineering*, 30(5):403–416, 2004.

[18] R. Fraanje, H. Sayed Ali, M. Verhaegen, and J. Doelman Niek. A fast-array Kalman filter solution to active noise control. *International Journal of Adaptive Control and Signal Processing*, 19(2–3):125–152, 2004.

[19] A. González and J.J. López. Fast transversal filters for deconvolution in multichannel sound reproduction. *IEEE Trans. on Speech and Audio Processing*, 9(4):429–440, 2001.

[20] G.J. Foschini and M.J. Gans. On limits of wireless communications in a fading environment when using multiple antennas. *Wireless Personal Communications*, 6(3):311–335, 1998.

[21] G.J. Foschini. Layered space-time architecture for wireless communications in a fading environment when using multiple antennas. *Bell Labs Technical Journal*, 1:41–59, 1996.

[22] B. Hassibi. An efficient square-root algorithm for BLAST. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages II737 – II740, 2000.

[23] M. Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. *SIGSAM Bull.*, 15(1):37–44, Feb. 1981.

[24] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Trans. Inf. Theory*, 48(8), 2002.

[25] E. Viterbo and J. Boutros. A universal lattice decoder for fading channels. *IEEE Trans. Inf. Theory*, 45(5), July 1999.

[26] M. Damen, A. Chkeif, and J.-C. Belfiore. Lattice code decoder for space-time codes. *IEEE Com. Let*, 4(5), 2000.

[27] B. Hassibi and H. Vikalo. On the expected complexity of sphere decoding. In *Asilomar Conf. Sig., Sys. and Comp.*, 2001.

[28] J. Jalden, C. Martin, and B. Ottersten. Semidefinite programming for detection in linear systems - optimality conditions and space-time decoding. In *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP-2003)*, April 2003.

[29] B. Steingrimsson, Z. Q. Luo, and K. M. Wong. Soft quasi-maximumlikelihood detection for multiple-antenna wireless channels. *IEEE Transactions on Signal Processing*, 51(11):2710–2719, Nov. 2003.

[30] N. Ibrahim H. Laamari, J.C. Belfiore. Comparison of the performance and the complexity of some ML detectors. In *IEEE VTC*, 2004.

[31] A. Wiesel, Y Eldar, and S. Shamai. Semidefinite relaxation for deteccion of 16QAM signaling in MIMO channels. *IEEE Sig. Pro. Let.*, 12(9), 2005.

[32] H. Vikalo, B. Hassibi, , and T. Kailath. Iterative decoding for MIMO channels via modified sphere decoding. *IEEE Transactions on Wireless Communications*, 3(6), November 2004.

[33] H. Vikalo and B. Hassibi. Maximum-likelihood sequence detection of multiple antenna systems over dispersive channels via sphere decoding. *EURASIP Journal on Applied Signal Processing*, 5:525–531, 2002.

[34] N. Al-Dhahir and A. H. Sayed. The finite-length multi-input multi-output MMSE-DFE. *IEEE Trans. Sig. Pro.*, 48(10), Oct. 2000.

[35] N. Al-Dhahir. FIR channel-shortening equalizers for MIMO ISI channels. *IEEE Trans. Comm.*, 49:213–218, 2001.

[36] C. Toker, S. Lambotharan, J.A. Chambers, and B. Baykal. Joint spatial and temporal channel-shortening techniques for frequency selective fading MIMO channels. *IEE Proc. Comm.*, 152(1), Feb. 2005.

[37] A. González and J.J. López. Fast transversal filters for deconvolution in multichannel sound reproduction. *IEEE Trans. On Speech and Audio Processing*, 9(4):429–440, 2001.

[38] W. Zhao and G. Giannakis. Sphere decoding algorithms with improved radius search. *IEEE Trans Comm.*, 57(7), July 2005.

[39] D.E. Culler, J.P. Singh, and A. Gupta. *Parallel Computer Architecture. A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[40] Julio Ortega, Mancia Anguita, and Alberto Prieto. *Arquitectura de Computadores*. Thomson, 2005.

[41] M.J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901—1909, December 1966.

[42] W. Händler. The impact of classification schemes on computer architecture. In *Proc. Int. Conference on Parallel Proc.*, pages 7–15, 1977.

[43] Kai Hwang. *Advanced Computer Architecture, Parallelism, Scalabiliyt, Programability*, chapter 3, pages 129–147. McGraw Hill, 1993.

[44] G.S. Almasi and A. Gottlieb. *Highly Parallel Computing*. The Benjamin/Cummings Publishing Company Inc., second edition, 1994.

[45] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26(2):62–76, February 1993.

[46] José Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10), October 1995.

[47] Parasoft Corporation, Pasadena, CA, USA. *Express Users's Guide*, 1992. Version 3.2.5.

[48] L. Bomans, Dirk Roose, and R. Hempel. The argonne/gmd macros in fortran for portable parallel programming and their implementation on the intel ipsc/2. *Parallel Computing*, 15(1–3):119–132, 1990.

[49] R. Calkin, R. Hempel, H.-C. Hoppe, and P. Wypior. Portable programming with the parmacs message-passing library. *Parallel Comput.*, 20(4):615–632, 1994.

[50] William D. Gropp and Barry Smith. Chameleon parallel programming tools users manual. Technical Report ANL-93/23, Argonne National Laboratory, March 1993.

[51] Edinburgh Parallel Computing Centre, University of Edinburgh. *CHIMP Concepts*, 1991.

[52] Edinburgh Parallel Computing Centre, University of Edinburgh. *CHIMP Version 1.0 Interface*, May 1992.

[53] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley. A user's guide to PICL : a portable instrumented communication library. Technical Report TM-11616, Oak Ridge National Laboratory, June 1991.

[54] A. Skjellum, S. Smith, C. Still, A. Leung, and M. Morari. The Zipcode message passing system. Technical report, Lawrence Livermore National Laboratory, September 1992.

[55] A. Skjellum and A. Leung. Zipcode: a portable multicomputer communication library atop the reactive kernel. In D. W. Walker and Q. F. Stout, editors, *Proceedings of the Fifth Distributed Memory Concurrent Computing Conference*, pages 767–776. IEEE Press, 1990.

[56] Anthony Skjellum, Steven G. Smith, Nathan E. Doss, Alvin P. Leung, and Manfred Morari. The design and evolution of Zipcode. *Parallel Comput.*, 20(4):565–596, 1994.

[57] R. Butler and Ewing Lusk. User's guide to the p4 programming system. Technical Report TM-ANL-92/17, Argonne National Laboratory, 1992.

[58] Ralph M. Butler and Ewing L. Lusk. Monitors, messages, and clusters: the p4 parallel programming system. *Parallel Comput.*, 20(4):547–564, 1994.

[59] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[60] J. Dongarra V.S. Sunderam, G.A. Geist and R. Manchek. The PVM concurrent computing system: Evolution, experiences, and trends. *Parallel Computing*, 20(4):531–545, April 1994.

[61] Nathan Doss, William Gropp, Ewing Lusk, and Anthony Skjellum. A model implementation of MPI. Technical report, Argonne National Laboratory, 1993.

[62] J.J. Dongarra, R. Hempel, A.J.G. Hey, and D.W. Walker. A proposal for a user-level, message passing interface in a distributed memory environment. Technical Report TM-12231, Oak Ridge National Laboratory, February 1993.

[63] Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*, June 1995.

[64] Peter Pacheco. *Parallel programming with MPI*. Morgan Kaufmann, 1996.

[65] Richard L. Graham, Galen M. Shipman, Brian W. Barrett, Ralph H. Castain, George Bosilca, and Andrew Lumsdaine. Open MPI: A high-performance, heterogeneous MPI. In *Proceedings, Fifth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, Barcelona, Spain, September 2006.

[66] E. Gabriel, G. Fagg, G. Bosilica, T. Angskun, J.J. Dongarra, J. Squyres, V. Sahay, P. Kambadur, B. Barret, A. Lumsdaine, R. Castain, D. Daniel, R. Graham, and T. Woodall. Open MPI: goals, conceps, and design of a next generation MPI implementation. In *11th European PVM/MPI Users' Group Meeting*, 2004.

[67] Parallel extensions for Fortran. Technical Report X3H5/93-SD1-Revision M, Accredited Standards Committee X3, April 1994.

[68] Rohit Chandra, Ramesh Menon, Leo Dagum, David Kohr, Dror Maydan, and Jeff McDonald. *Parallel Programming in OpenMP*. Morgan Kaufmann, 2001.

[69] Michael J. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw Hill, 2003.

[70] L. Dagum and R. Menon. OpenMP: an industri standard API for shared memory programming. *Computational Science and Engineering*, 5(1), January-March 1998.

[71] Vipin Kumar, Ananth Gram, Anshul Gupta, and George Karypis. *An Introduction to Parallel Computing: Design and Analysis of Algorithms*, chapter 4. Addison-Wesley, Harlow, England, second edition, 2003.

[72] Beverly A. Sanders, Berna Massingill, and Timothy G. Mattson. *The Algorithm Structure Design Space in Parallel Programming*. Addison Wesley, 2005.

[73] Timothy G. Mattson, Beverly A. Sanders, and Berna L. Massingill. *Patterns for Parallel Programming*. Addison Wesley Professional, 2005.

[74] C.-T. King, W.-H. Chou, and L.-M. Ni. Pipelined data parallel algorithms-concept and modeling. In *ICS '88: Proceedings of the 2nd international conference on Supercomputing*, pages 385–395, New York, NY, USA, 1988. ACM Press.

[75] Larry R. Nyhoff and Sanford C. Leestma. *Fortran 90 for Engineers & Scientists*. Prentice Hall, New Jersey, NJ, USA, 1997.

[76] Stepehen J. Chapman. *Fortran 90/95 for Scientists and Engineers*. McGraw Hill, New York, NY, USA, second edition, 2004.

[77] H.P Flatt and K. Kennedy. Performance of Parallel Processors. *Parallel Computing*, 31:1–20, 1989.

[78] Alan H. Karp and Horace P. Flatt. Measuring Parallel Processor Performance. *Communications of the ACM*, 33(5):539–543, May 1990.

[79] G.M. Amdahl. Validity of the single processor aproach to achieving large scale computing capabilities. In *AIFPS Conference Proceedings*, April 1967.

[80] John L. Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 33(5):532–533, May 1988.

[81] Xian-He Sun and Diane T. Rover. Scalability of Parallel Algorithm-Machine combinations. *IEEE Transactions on Parallel and Distributed Systems*, 5(6):599–613, June 1994.

[82] S. Sahni and V. Thanvantri. Parallel computing: Performance metrics and models. Technical report, Computer & Science Department, University of Florida, May 1995.

[83] Ignacio Martín and Francisco Tirado. Relationships between Efficiency and Execution Time of Full Multigrid Methods on Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):562–573, June 1997.

[84] Ignacio Martín, Francisco Tirado, and Luis Vázquez. Some Aspects about the Scalability of scientific applications on Parallel Architectures. *Parallel Computing*, 22(9):1169–1195, 1996.

[85] Daniel Nussbaum and Anant Agarwal. Scalability of Parallel Machines. *Communications of the ACM*, 34(3):57–61, March 1991.

[86] Anshul Gupta, Vipin Kumar, and Ahmed Sameh. Performance and Scalability of Preconditioned Conjugated Gradient Methods on Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems*, 6(5):455–469, May 1995.

[87] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[88] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. Classics in Applied Mathematics. SIAM, Philadelphia, 1995.

[89] T. Kailath, A. Sayed, and B. Hassibi. *Linear Estimation*. Prentice Hall Information and System Sciences Series, New Jersey, USA, 2000.

[90] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[91] Craig Lucas Sven Hammarling, Nick Higham. LAPACK-style codes for pivoted Cholesky and QR updating. In *PARA'06*, June 2006.

[92] Robert Schreiber and Charles van Loan. A storage-efficient wy representation for products of householder transformations. *SIAM J. Sci. Stat. Comput.*, 10(1):53–57, 1989.

[93] S. Thomas Alexander and Avinash L. Ghirnikar. A method for recursive least squares filtering based on an inverse QR decomposition. *IEEE Transactions on Signal Processing*, 41(1), January 1993.

[94] F.J. Martínez Zaldívar, A. González, and A.M. Vidal. Variantes algorítmicas del filtrado de Kalman: inicialización de los métodos y ruido en la observación. Technical Report DSIC-II/22/2002, Departamento de Sistemas Informáticos y Computación, Octubre 2001.

[95] F.J. Martínez Zaldívar and A.M. Vidal. Paralelización del algoritmo RLS (Mínimos Cuadrados Recursivos), versión raíz cuadrada (algoritmo QR inversa). In *Métodos Numéricos en Ingeniería V*, 2002.

[96] F.J. Martínez Zaldívar, A.M.Vidal, and A. González. Algoritmo paralelo para el problema RLS basado en el Filtro de Kalman, con extensión a bloques y aplicaciones en tiempo real. In *XIII Jornadas de Paralelismo, Lleida (Spain)*, Septiembre 2002.

[97] F.J. Martínez Zaldívar, A.M. Vidal Maciá, and A. González. A parallel algorithm based on a variant of the Kalman filter for solving the RLS problem. *WSEAS Transactions on Circuits and Systems*, 3(10):2143–2148, 2004.

[98] F.J. Martínez Zaldívar, A.M. Vidal Maciá, and A. González. A parallel RLS algorithm for heterogeneous systems. *WSEAS Transactions on Signal Processing*, 2(1):16–23, 2006.

[99] F.J. Martínez Zaldívar, A.M. Vidal Maciá, and A. González. An OpenMP+/MPI recursive least squares pipelined parallel algorithm based on the square root and extended version information filter for heterogenous parallel systems. In *8th WSEAS International Conference on Applied Mathematics*, 2005.

[100] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley. *ScaLAPACK user's guide*. SIAM, 1997.

[101] E.F. D'Azevedo and J.J. Dongarra. The design and implementation of the parallel out-of-core ScaLAPACK LU, QR, and Cholesky factorization routines. Technical Report LAPACK Working Note 118, CS-97-247, University of Tennesse, Knoxville, January 1997.

[102] Brian C. Gunter and Robert A. Van De Geijn. Parallel out-of-core computation and updating of the qr factorization. *ACM Trans. Math. Softw.*, 31(1):60–78, 2005.

[103] D. Wüben, R. Böhnke, V. Kühn, and K. Kammeyer. Near-maximum-likelihood detection of MIMO systems using MMSE-based lattice-reduction. In *IEEE ICC*, volume 2, pages 798–802, June 2004.

[104] E. Khan. Maximum likelihood detection of a MIMO system using second order cone programming approach. In *IEEE 5th Workshop on Signal Processing Advances in Wireless Communications*, 2004.

[105] Henuchul Lee, Byeongsi Lee, and Inkyu Lee. Iterative detection and decoding with an improved V-BLAST for MIMO-OFDM systems. *IEEE Journal on Selected Areas in Communications*, 24(3), March 2006.

[106] B. Farhang-Boroujeny, Haidong Zhu, and Zhenning Shi. Markov chain Monte Carlo algorithms for CDMA and MIMO communication systems. *IEEE Transactions on Signal Processing*, 54(5), May 2006.

[107] Yang-Seok Choi, Peter J. Voltz, and Frank A. Cassara. On channel estimation and detection for multicarrier signals in fast and selective Rayleigh fading channels. *IEEE Transactions on Communications*, 49(8), August 2001.

[108] F.J. Martínez Zaldívar, A.M. Vidal Maciá, and D. Giménez. A parallel OSIC algorithm based on the information filter for heterogeneous networks. In *HETEROPAR-07, Sixth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, Austin, Texas, Septiembre 2007.

[109] F.J. Martínez Zaldívar, A.M. Vidal Maciá, and A. González. A heterogeneous network pipelined parallel algorithm for the Recursive Least Squares problem and OSIC-BLAST. *Concurrency and Computation: Practice and Experience*. Sometido.

[110] F.J. Martínez Zaldívar, A.M. Vidal Maciá, and A. González. A heterogeneous pipelined parallel algorithm for minimum squared error estimation with ordered succesive interference cancellation. In *ParCo'07*, Jülich—Aachen, Alemania, Septiembre 2007.

[111] F.J. Martínez Zaldívar, A.M. Vidal Maciá, and P. Alonso. A pipelined parallel algorithm for OSIC decoding. In *PPAM 2007 Seventh International Conference on Parallel Processing and Applied Mathematics*, Gdansk, Polonia, Septiembre 2007.