Universidad Zaragoza
1542

Trabajo Fin de Grado

Método Paralelo para la Resolución de Ecuaciones de Ligadura para Moléculas Lineales con Ramificaciones Laterales Idénticas

Parallel Constraint Solver for Linear Molecules with Identical Side Chains

Autor

Lorién López Villellas

Directores

Jesús Alastruey Benedé

Pablo Ibáñez Marín

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2020

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe entregarse en la Secretaría de la EINA, dentro del plazo de depósito del TFG/TFM para su evaluación).

D./Dª. Lorién López Villellas ,en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza, Declaro que el presente Trabajo de Fin de (Grado/Máster) Ingeniería Informática (Título del Trabajo)

Método Paralelo para la Resolución de Ecuaciones de Ligadura para Moléculas Lineales con Ramificaciones Laterales Idénticas

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 26 de Abril de 2020

Fdo:

Método Paralelo para la Resolución de Ecuaciones de Ligadura para Moléculas Lineales con Ramificaciones Laterales Idénticas

RESUMEN

Mediante la dinámica molecular se puede simular el comportamiento de un sistema de átomos a lo largo del tiempo. Esta herramienta permite realizar experimentos sin necesidad de disponer de las sustancias reales, lo que aumenta la accesibilidad y reduce el coste de los experimentos. Entre otras muchas aplicaciones, la dinámica molecular se usa para el diseño de nuevos fármacos o el análisis de materiales. A la hora de realizar experimentos, se acostumbra a restringir algunos de los grados de libertad internos del sistema a estudiar para aumentar el paso temporal de la simulación. Una de las restricciones más comúnmente aplicadas es la imposición de ligaduras (restricción en la longitud del enlace de una pareja de átomos). Los métodos más usados para imponer ligaduras convergen muy lentamente y están basados en aproximaciones que afectan a su estabilidad numérica. Esta forma de implementar imposiciones de ligadura puede ser reemplazada por cálculos analíticos, lo que aumenta la eficiencia y la precisión de los experimentos.

En este proyecto se presenta una implementación paralela de ILVES, un nuevo algoritmo para imponer ligaduras que converge cuadráticamente. Se ha abordado la resolución de ecuaciones de ligadura para moléculas compuestas por cadenas de átomos lineales con ramificaciones lineales idénticas, y se ha conseguido extender la implementación para moléculas compuestas por cadenas de átomos reales con ramificaciones reales idénticas. La versión paralela de ILVES implementada es capaz de explotar el paralelismo vectorial (unidades funcionales capaces de operar con múltiples datos) y el paralelismo multinúcleo (procesadores que integran varios núcleos de ejecución, *cores*) existentes en los procesadores comerciales actuales, por lo que mejora el rendimiento de otros métodos de imposición de ligaduras. Además, resuelve eficientemente las ecuaciones de ligadura hasta el límite de precisión máquina.

Índice

Capítulo 1

Introducción

1.1. Motivación

Mediante la dinámica molecular se puede simular el comportamiento de un sistema de átomos a lo largo del tiempo. Esta herramienta permite realizar experimentos sin necesidad de disponer de las sustancias reales, lo que aumenta la accesibilidad y reduce el coste de los experimentos. La dinámica molecular hace posible el estudio de diferentes propiedades fisicoquímicas, lo que permite, entre otras muchas aplicaciones, el análisis de materiales o el diseño de nuevos fármacos. Por ejemplo, las simulaciones de dinámica se utilizan actualmente en la lucha contra la COVID-19 [1].

Para realizar experimentos de dinámica molecular más eficientes se acostumbra a restringir algunos de los grados de libertad internos del sistema a estudiar. La opción más común es imponer restricciones, denominadas ligaduras, en la longitud de los enlaces atómicos o en los ángulos entre enlaces. Esta técnica permite aumentar el paso temporal de la simulación, y con ello lograr mayores tiempos simulados, lo que incrementa el poder predictivo del experimento. Algunos de los métodos más utilizados, como SHAKE [2] y LINCS [3], hacen uso de aproximaciones y métodos iterativos para fijar la longitud del enlace de una pareja de átomos. Esta forma de implementar imposiciones de ligadura puede ser reemplazada por cálculos analíticos, lo que aumenta la eficiencia y la precisión de los experimentos.

1.2. Objetivos y Tareas

El objetivo de este trabajo es realizar una implementación paralela de un nuevo algoritmo para resolver ecuaciones de imposición de ligaduras, ILVES. Se parte de un trabajo anterior, en el cual se implementa este mismo algoritmo para cadenas de átomos lineales sin ramificaciones [4]. Aunque el objetivo inicial del presente trabajo era

implementar el algoritmo para moléculas compuestas por cadenas de átomos **lineales** con ramificaciones **lineales** idénticas, se ha conseguido extender el algoritmo para soportar moléculas compuestas por cadenas de átomos **reales** con ramificaciones **reales** idénticas. En concreto, el algoritmo permite resolver las ecuaciones de ligadura para una molécula con ramificaciones de lisina, un aminoácido de tamaño medio. Esta estructura molecular es muy próxima a la que se encuentra en la naturaleza, siendo necesario, únicamente, permitir cualquier tipo de ramificación real, así como permitir ramificaciones de distinto tipo en la misma molécula. Una vez completado este paso en futuros trabajos, se pretende integrar esta implementación en GROMACS [5], una de las aplicaciones de dinámica molecular más usadas.

Para resolver el sistema de ecuaciones se hace uso del método de Newton y de eliminación Gaussiana, además, para permitir la vectorización y paralelización multinúcleo de la resolución se ha utilizado el método del complemento de Schur.

Antes de realizar la implementación del algoritmo se ha realizado una labor de documentación acerca de la dinámica molecular, además del estudio a fondo de los proyectos anteriormente realizados sobre ILVES [4], [6], [7]. En segundo lugar se ha desarrollado el algoritmo a utilizar para la resolución del problema. Tras esto se ha realizado la implementación paralela del algoritmo. Finalmente se han evaluado las prestaciones conseguidas, comparando los resultados con algoritmos del estado del arte, como SHAKE y LINCS, así como con anteriores implementaciones de ILVES. En la Figura 1.1 se puede ver la distribución del tiempo dedicado al proyecto.

ID	TAREA	DURACIÓN	SEPTIEMBRE	OCTUBRE	NOVIEMBRE	DICIEMBRE	ENERO	FEBRERO	MARZO	ABRIL	MAYO	JUNIO
1	Estudio de documentación											
1.1	Estudio de documentación general sobre dinámica molecular	13 horas										
1.2	Estudio de documentación de anteriores proyectos	19 horas										
1.3	Estudio de código de anteriores proyectos	59 horas										
2	Estudio de implementación de ILVES											
2.1	Estudio de implementación de ILVES	34 horas										
3	Implementación de ILVES											
3.1	ILVES para cadenas reales con ramificaciones lineales	90 horas										
3.2	ILVES para cadenas reales con ramificaciones de lisina	26 horas										
3.3	Módulos complementarios a ILVES	39 horas										
4	Experimentos											
4.1	Scripts de ejecución automática de experimentos	13 horas										
4.2	Ejecución de experimentos y generación de resultados	6 horas										
5	Documentación											
5.1	Documentación ajena a la memoria	19 horas										
5.2	Memoria	80 horas										
6	Reuniones											
6.1	Reuniones	43 horas										

Figura 1.1: Diagrama de Gantt del proyecto.

1.3. Descripción del Contenido

Este documento contiene cinco capítulos, incluyendo esta introducción. En el capítulo 2 se explican los conceptos de dinámica molecular necesarios para entender el resto de la memoria. Se explica el proceso de imposición de ligaduras para mejorar el rendimiento de las simulaciones de dinámica molecular, así como los algoritmos más utilizados con

este objetivo, SHAKE y LINCS. También se presenta ILVES, un nuevo algoritmo para imponer ligaduras del cual se quiere hacer una implementación paralela, objetivo de este trabajo.

En el capítulo 3 se explica como se ha realizado la implementación paralela de ILVES y se hace un breve análisis de rendimiento de la misma.

En el capítulo 4 se presenta la metodología que se ha usado para ejecutar experimentos, se describen los experimentos realizados y se analizan los resultados obtenidos.

En el capítulo 5 se extraen las conclusiones del proyecto.

Se han añadido tres anexos con información adicional que complementan algunos conceptos explicados a lo largo de la memoria. El primer anexo explica el paralelismo vectorial. El segundo anexo expone el fenómeno de *false sharing* y cómo puede afectar negativamente al rendimiento de un programa. El último anexo contiene información acerca del código de la implementación paralela de ILVES.

Capítulo 2

Fundamentos de dinámica molecular

El objetivo de este capítulo es hacer una introducción al campo de la dinámica molecular. En la primera sección se explican los pasos de una simulación de dinámica molecular. A continuación se describe el proceso de imposición de ligaduras para acelerar las simulaciones. Por último se explica el procedimiento mediante el cual los algoritmos SHAKE, LINCS e ILVES aplican la imposición de ligaduras.

2.1. Simulaciones de dinámica molecular

El objetivo de una simulación de dinámica molecular es describir la evolución de un sistema molecular a lo largo del tiempo, teniendo en cuenta todas las fuerzas que actúan sobre cada átomo en cada instante temporal. Para conocer la posición de cada átomo del sistema al final de cada paso temporal se resuelve la ecuación clásica del movimiento según la segunda ley de Newton:

$$F_i(t) = m_i \cdot a_i(t) = m_i \frac{\partial v_i(t)}{\partial t} = m_i \frac{\partial^2 x_i(t)}{\partial t^2} \tag{2.1}$$

siendo $F_i(t)$ la fuerza que actúa sobre el átomo i en el instante t, m_i la masa del átomo i en el instante t, $a_i(t)$ la aceleración del átomo i en el instante t, $v_i(t)$ la velocidad del átomo i en el instante t y x_i el vector de posición (x_x, x_y, x_z) del átomo i en el instante t.

Para calcular F_i en el instante t se usa la siguiente ecuación:

$$F_i(t) = -\frac{\partial V(x_1(t), ..., x_n(t))}{\partial x_i(t)} \tag{2.2}$$

donde $\partial V(x_1(t), ..., x_n(t))$ representa la energía potencial por la interacción de los N átomos en el instante t como una función de sus posiciones y $x_i(t)$ la posición del átomo i en el instante t.

Para un sistema molecular con N átomos no existe solución algebraica. Por otro lado, la solución numérica es directa:

$$\begin{aligned} x_i(t + \triangle t) &= x_i(t) + \triangle t \cdot v_i(t) \\ v_i(t + \triangle t) &= v_i(t) + \frac{\triangle t}{m_i} F_i(t) \end{aligned} \tag{2.3}$$

Resolviendo el anterior sistema se puede conocer la velocidad y posición de cada átomo en el instante $t + \triangle t$ cuando dichos valores son conocidos en el instante t. $\triangle t$ es conocido como el paso temporal (*time step*).

Para aumentar la precisión obtenida al resolver el anterior sistema, se suelen usar algoritmos numéricos (*numerical integrators*), uno de los más utilizados es el algoritmo del salto de la rana *(leap-frog algorithm*), que transforma el anterior sistema de ecuaciones en el siguiente:

$$\begin{aligned} x_i(t + \triangle t) &= x_i(t) + \triangle t \cdot v_i(t + \frac{\triangle t}{2}) \\ v_i(t + \frac{\triangle t}{2}) &= v_i(t - \frac{\triangle t}{2}) + \frac{\triangle t}{m_i} F_i(t) \end{aligned} \tag{2.4}$$

2.2. Imposición de ligaduras

El alcance de una simulación de dinámica molecular está limitado por el paso temporal elegido (*time step*, $\triangle t$). La distancia entre dos átomos, es decir, la longitud de su enlace, no es fija, esto es debido a que los átomos vibran ligeramente. El periodo de estas vibraciones es del orden de femtosegundos (fs, 10^{-15} s), por lo que el paso temporal elegido deberá ser menor que dicho periodo, normalmente cinco veces menor. Muchos de los procesos fisicoquímicos a estudiar tienen una duración del orden de segundos, por lo que el paso temporal impuesto por las vibraciones de cada átomo individual es excesivamente limitante. Por ejemplo, simular un proceso de un segundo con pasos temporales de 1 fs requeriría 10^{15} iteraciones. Para eliminar esta limitación se pueden incorporar restricciones de ligadura, es decir, congelar la longitud de la ligadura entre cada pareja de átomos del sistema. Esto permite aumentar el paso temporal sin afectar al cálculo de las magnitudes de interés, a esta técnica se le conoce como imposición de

ligaduras. La ligadura de un enlace k entre dos átomos a_k y b_k se define como:

$$\sigma_k = ||x_{a_k} - x_{b_k}||_2^2 - (d_{a_k,b_k})^2 = 0 \tag{2.5}$$

donde x_{a_k} y x_{b_k} son los vectores de posición de los átomos a_k y b_k y d_{a_k,b_k} es la longitud del enlace atómico formado por los átomos a_k y b_k.

La imposición de ligaduras implica que nuevas fuerzas aparezcan en el sistema, las fuerzas de ligadura, esto provoca cambios en la ecuación del movimiento según la segunda ley de Newton:

$$F_i(t) - \sum_{k=1}^{M} \lambda_k \frac{\partial \sigma_k}{\partial r_i}(t) = m_i \cdot a_i(t) \tag{2.6}$$

donde M es el número de ligaduras del sistema, m_i es la masa del átomo i, $a_i(t)$ es la aceleración del átomo i en el instante t, $F_i(t)$ es la fuerza ejercida sobre el átomo i en el instante t, $-\sum_{k=1}^{M} \lambda_k \frac{\partial \sigma_k}{\partial r_i}(t)$ es la fuerza de ligadura sobre el átomo i en el instante t, λ_k es el multiplicador de Lagrange [8] asociado a la ligadura k y σ_k es la ligadura k.

La existencia de M ligaduras convierte el sistema de $3N$ ecuaciones con $3N$ incógnitas en un sistema con $3N + M$ ecuaciones y $3N + M$ incógnitas. El proceso de resolución del sistema se hace en dos fases, primero se calculan las posiciones de los átomos sin tener en cuenta la imposición de ligaduras y posteriormente se calculan las fuerzas de ligadura para corregir la posición de cada átomo.

Al añadir las fuerzas de ligadura al sistema, las vibraciones de átomos, anteriormente descritas, ya no serán simuladas, por lo que el paso temporal ya no estará limitado por la frecuencia de dichas vibraciones.

2.3. SHAKE

SHAKE es un algoritmo iterativo que usa el método de Gauss-Seidel para resolver ecuaciones de ligadura. El método resulta en un sistema con M ecuaciones y M incógnitas, siendo M el número de ligaduras del sistema molecular.

Para cada ligadura k, desde 1 hasta M, se debe corregir el conjunto de coordenadas obtenidas tras calcular la posición de sus átomos sin aplicar fuerzas de ligadura. Para calcular el multiplicador de Lagrange de cada ligadura y poder aplicar este paso se requiere: la posición resultante de cada átomo $i \in k$ tras aplicarle todas las fuerzas, menos las de ligadura; $x_i(t + \triangle t)$, y la posición final de cada átomo $i \in k$ del paso temporal

anterior; $x_i(t)$. Estas posiciones se usan como referencia para aplicar la corrección, ya que, al corregir la posición de un átomo, este se mueve paralelamente al enlace del paso temporal anterior, como se puede ver en la Figura 2.1.

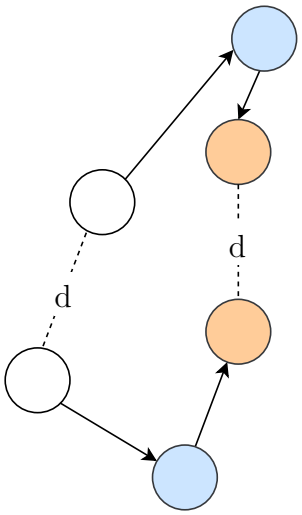

Figura 2.1: Corrección de posiciones en SHAKE. Los círculos blancos representan las posiciones de los átomos en el instante temporal t. Los círculos azules representan las posiciones de los átomos en el instante $t + \triangle t$, antes de ejecutar SHAKE. Los círculos naranjas representan las posiciones de los átomos en el instante $t + \triangle t$, tras ejecutar SHAKE. Nótese que las flechas que unen los átomos azules a los naranjas son paralelas a la ligadura (d).

Al resolver la ecuación asociada a una ligadura k y aplicar la corrección al conjunto de átomos que la componen, se asegura que la posición de dichos átomos satisface la restricción de dicha ligadura k, sin embargo, se deben volver a mover átomos cuyas posiciones se habían corregido anteriormente, rompiendo parcialmente la satisfacción de las restricciones de las ligaduras anteriores a k. Esto implica que, para reducir el error en la posición de todos los átomos del sistema, el método deba ejecutarse varias veces (método iterativo). La ejecución de cada una de estas repeticiones tiene coste lineal ($O(n)$).

2.4. LINCS

LINCS es otro algoritmo para resolver ecuaciones de ligadura muy usado en la actualidad, por ejemplo en GROMACS.

Al igual que SHAKE, LINCS utiliza las posiciones finales de los átomos en el instante temporal anterior; $x_i(t)$, y la posición resultante de cada átomo tras aplicarle todas las fuerzas, menos las de ligadura; $x_i(t + \triangle t)$. El algoritmo consta de dos fases. En la primera fase, cada átomo de cada pareja de átomos unidos por un enlace, se mueve en

paralelo al enlace del paso temporal anterior, hasta que alcanza la altura de su posición en dicho paso temporal. En la segunda fase, cada átomo se sigue moviendo en paralelo al enlace del paso temporal $\triangle t$ hasta que se satisface la restricción de ligadura. Estos pasos se pueden ver en la Figura 2.2.

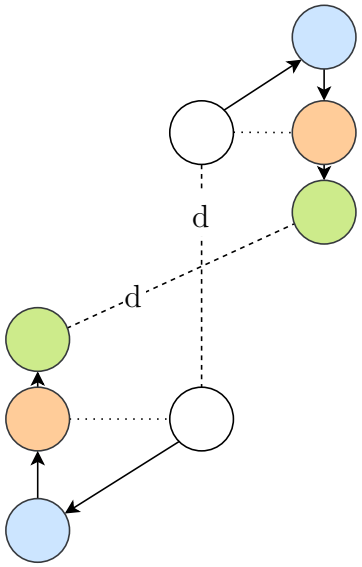

Figura 2.2: Corrección de posiciones en LINCS. Los círculos blancos representan las posiciones de los átomos en el instante temporal t. Los círculos azules representan las posiciones de los átomos en el instante $t + \triangle t$, antes de ejecutar LINCS. Los círculos naranjas representan las posiciones de los átomos en el instante $t + \triangle t$, tras ejecutar el primer paso de LINCS. Los círculos verdes representan las posiciones de los átomos en el instante $t + \triangle t$, tras ejecutar el segundo paso de LINCS.

LINCS no utiliza un método iterativo para obtener los multiplicadores de Lagrange, estos se obtienen haciendo una serie de expansiones para aproximar la inversa de la matriz Jacobiana J:

$$(I - J)^{-1} \approx I + J + J^2 + J^3 + J^4 + ... \tag{2.7}$$

El error en la posición de los átomos dependerá de la cantidad de potencias que se calculen. LINCS es entre tres y cuatro veces más rápido que SHAKE, pero el método solo funciona en moléculas con bajo grado de conectividad, por lo que no resulta un reemplazo completo a SHAKE.

2.5. ILVES

SHAKE puede tener problemas de convergencia e inestabilidad, además es un algoritmo no paralelizable, y LINCS no es adecuado para todo tipo de moléculas. El

Dr. Pablo García Risueño propone ILVES como algoritmo alternativo, que resuelve los problemas de convergencia e inestabilidad de SHAKE, es paralelizable y además es aplicable a cualquier tipo de molécula.

Este proyecto es parte de una línea de investigación que pretende integrar ILVES en GROMACS. Existen dos versiones de ILVES: ILVES-S, basada en SHAKE e ILVES-L, basada en LINCS. En este proyecto se implementa ILVES-S, que resuelve el mismo sistema de ecuaciones que SHAKE pero, en lugar de aplicar las correcciones a los átomos tras la resolución de cada ecuación, primero se resuelve por completo el sistema de ecuaciones y después se aplican todas las correcciones a la vez. Al igual que SHAKE, se trata de un método iterativo que se debe ejecutar varias veces para aumentar la precisión de la solución.

Ya se han hecho varias implementaciones exitosas de ILVES. La primera fue realizada por María Astón Serrano Gracia en su Proyecto de fin de carrera [6]. En dicho trabajo se hizo una implementación escalar y secuencial de ILVES, que sirvió para estimar las prestaciones, precisión y convergencia del algoritmo. En un trabajo posterior se realizó una implementación paralela de ILVES para un conjunto grande de moléculas pequeñas e idénticas (disolventes), en dicho trabajo se paraleliza la resolución del sistema de ecuaciones de cada molécula por separado [7]. La implementación precursora a la presentada en este proyecto fue realizada por Rubén Langarita Benítez [4]. En su Trabajo de fin de grado se realizó una implementación paralela de ILVES a nivel de molécula, que usaba tanto paralelismo vectorial como multinúcleo, para el caso específico de cadenas lineales sin ramificaciones. En este trabajo se presenta una implementación paralela a nivel de molécula de ILVES para cadenas reales con ramificaciones de lisina, que también usa paralelismo vectorial y multinúcleo. En futuros trabajos se deberá añadir la posibilidad de resolver ecuaciones de ligadura de moléculas con ramificaciones distintas, y por último, integrar la implementación en GROMACS.

Capítulo 3

Implementación paralela de ILVES

El lenguaje de programación usado para la implementación ha sido *C*, ya que se partía de código escrito en fases anteriores del proyecto [4], [7], además es el lenguaje usado normalmente en este tipo de herramientas. La implementación de ILVES explota dos tipos de paralelismo: paralelismo vectorial, mediante auto-vectorización del compilador y paralelismo multinúcleo, mediante la biblioteca OpenMP [9].

El proyecto se centra en la resolución de ecuaciones de ligadura, por lo que los pasos previos a la corrección de posiciones para satisfacer restricciones de ligadura son simulados dando posiciones semi-aleatorias a los átomos del sistema molecular.

Se comenzó abordando el caso de cadenas lineales con ramificaciones lineales idénticas (Figura 3.1), y además de eso se consiguió aumentar el alcance de la implementación, primero para cadenas reales con ramificaciones lineales idénticas (Figura 3.2) y por último para cadenas reales con ramificaciones reales idénticas, específicamente cadenas laterales de lisina (Figura 3.3). La explicación de la implementación se centrará en las cadenas reales, ya que para futuros proyectos resultan de mayor interés que las lineales.

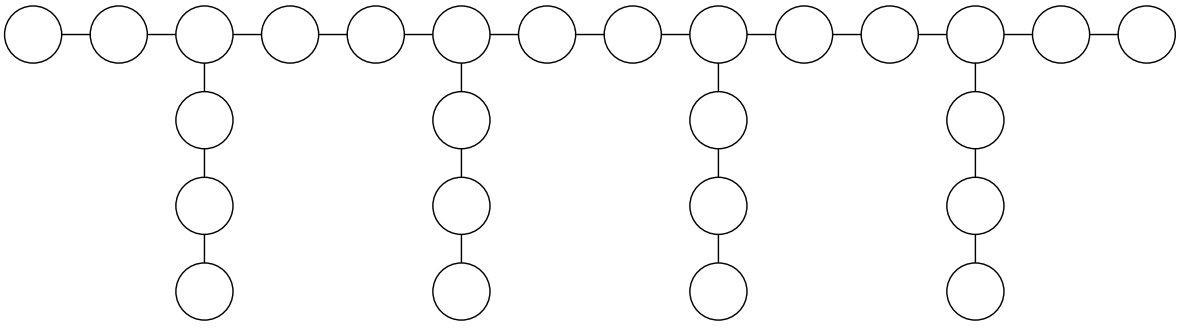

Figura 3.1: Ejemplo de cadena lineal con ramificaciones lineales idénticas de tres átomos.

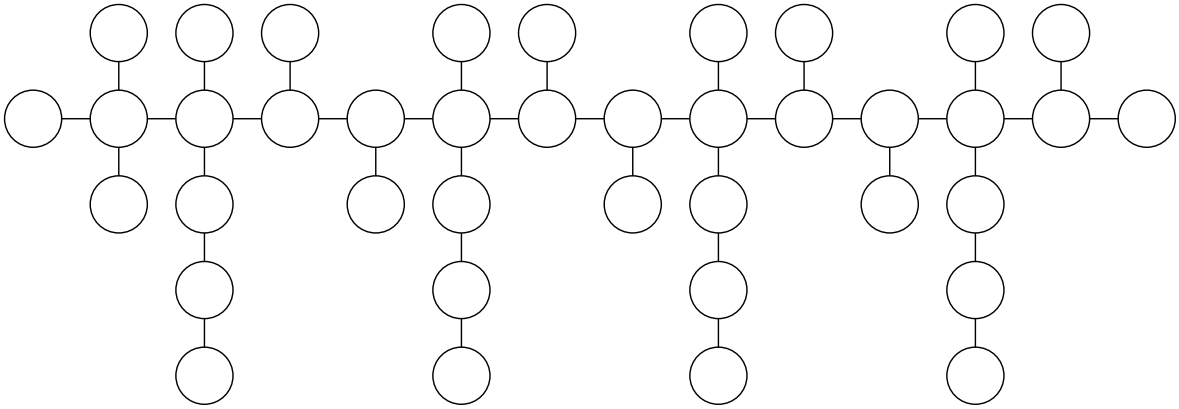

Figura 3.2: Ejemplo de cadena real con ramificaciones lineales idénticas de tres átomos.

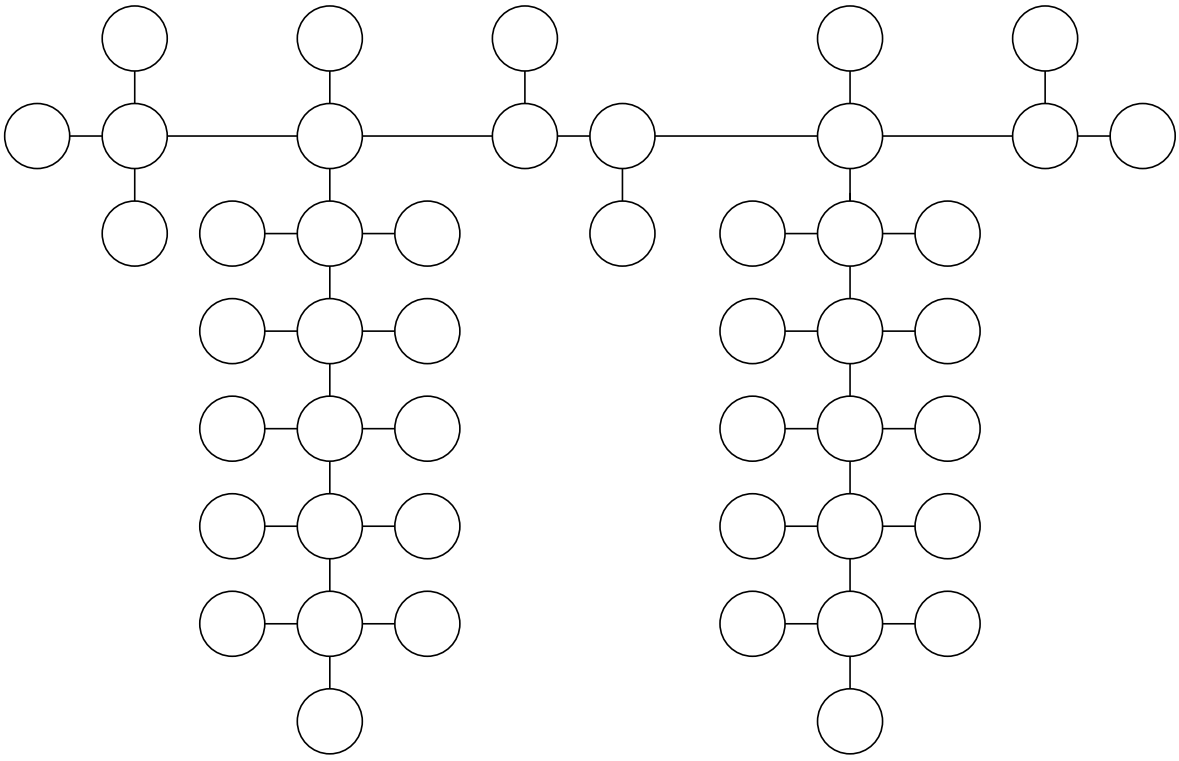

Figura 3.3: Ejemplo de cadena real con ramificaciones idénticas de lisina.

3.1. Generación del sistema de ecuaciones

Para una molécula compuesta por N átomos ILVES resulta en un sistema lineal con M ecuaciones y M incógnitas, siendo M el número de enlaces. Las incógnitas del sistema corresponden a los multiplicadores de Lagrange. Este sistema de ecuaciones se representa como una matriz de coeficientes cuadrada, simétrica y casi vacía, donde un elemento no nulo indica que los enlaces i y j tienen algún átomo en común. Cada elemento de la matriz de coeficientes se calcula con la siguiente ecuación [7]:

$$A = Dg(x) \cdot W^{-1} \cdot Dg(x)^T \tag{3.1}$$

donde A es la matriz de coeficientes inicial del sistema de ecuaciones a resolver, x es el vector de posición (x_x, x_y, x_z) de cada átomo, $Dg(x)$ es el Jacobiano $Dg(x) \in \mathbb{R}^{M \times 3N}$ y W es una matriz de masas diagonal de dimensión $3N$ que contiene las masas de los átomos del sistema molecular.

La matriz de términos independientes se calcula de la siguiente manera [7]:

$$g : \mathbb{R}^{3N} \to \mathbb{R}^M, g(x) = (g_1(x), g_2(x), ..., g_M(x))^T \tag{3.2}$$

donde x es el vector de posición de cada átomo y $g_k(x) = \frac{1}{2}||x_{a_k} - x_{b_k}||_2^2 - (d_{a_k,b_k})^2$, siendo d_{a_k,b_k} la longitud del enlace atómico formado por los átomos a_k y b_k.

El objetivo es hallar la matriz identidad usando eliminación Gaussiana. La numeración de los enlaces de la molécula resulta crucial para obtener patrones en la matriz de coeficientes del sistema que faciliten su resolución. Desde el punto de vista de la implementación, la numeración es importante para que el sistema de ecuaciones sea fácilmente particionable, de forma que su resolución se pueda paralelizar. El patrón de numeración ha sido estudiado y escogido meticulosamente para conseguir el mayor rendimiento posible. En las Figuras 3.4 y 3.5 se puede ver cómo se deben numerar los enlaces para una cadena real con ramificaciones lineales y una cadena real con ramificaciones de lisina respectivamente.

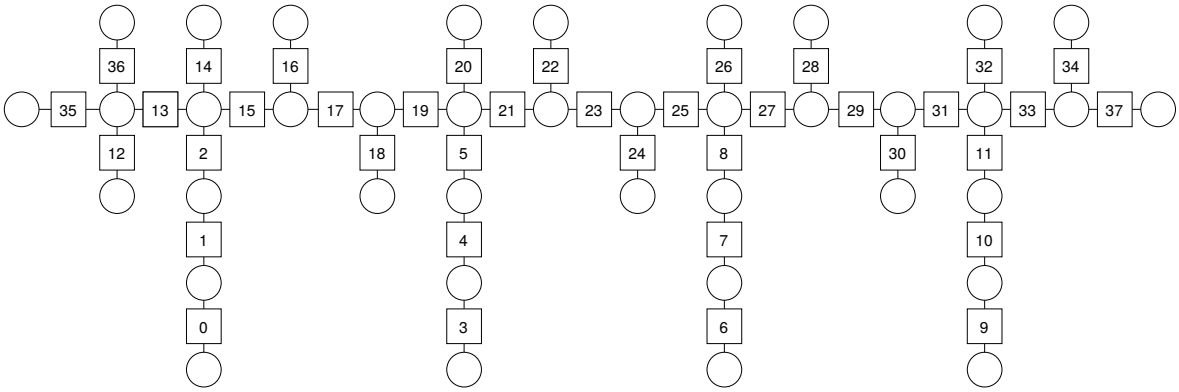

Figura 3.4: Numeración de enlaces aplicado a una cadena real con ramificaciones lineales idénticas de tres átomos.

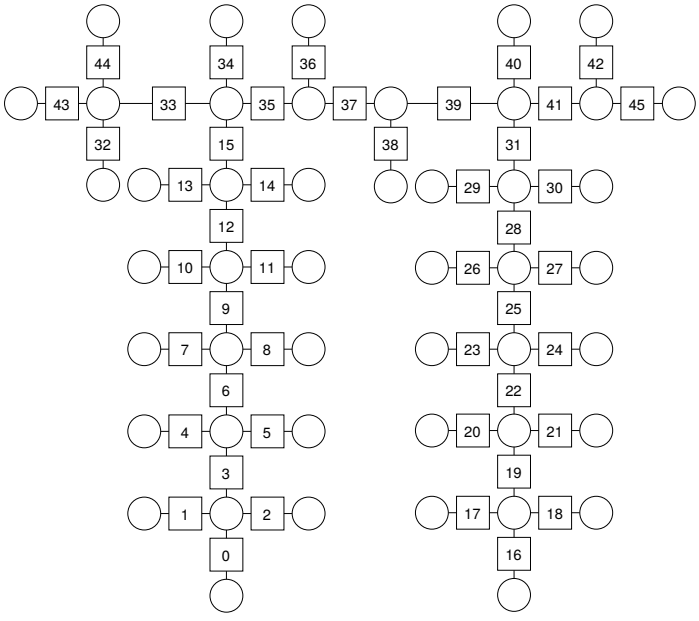

Figura 3.5: Numeración de enlaces aplicado a una cadena real con ramificaciones idénticas de lisina.

Los dos tipos de molécula anteriormente presentados tienen una estructura que sigue los patrones de las Figuras 3.6 y 3.7, además toda molécula tiene en sus extremos tres átomos extra, cuyos enlaces deberán ser los últimos en numerarse. Para numerar una molécula se debe seguir el orden alfabético presentado en las anteriores figuras —siempre de izquierda a derecha—. Se debe comenzar por los enlaces de las ramificaciones (rojo), seguido por los enlaces pertenecientes a la cadena principal (*backbone*, de color azul) y por último los enlaces especiales.

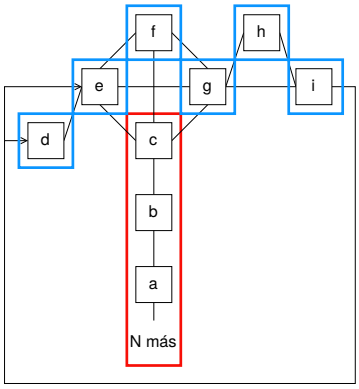

Figura 3.6: Patrón estructural en cadenas reales con ramificaciones lineales idénticas. Cada cuadrado representa un enlace, dos enlaces están unidos si tienen un átomo en común. En rojo los enlaces que pertenecen a ramificaciones de la molécula, en azul los que pertenecen a la cadena principal de la molécula.

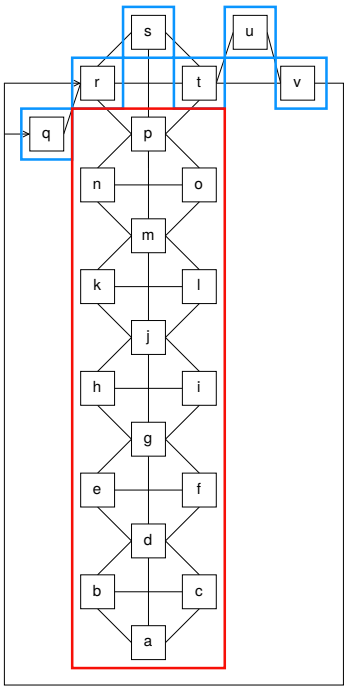

Figura 3.7: Patrón estructural en cadenas reales con ramificaciones idénticas de lisina. Se sigue la misma notación que en la Figura 3.6.

Una vez numerados los enlaces de una molécula se puede obtener el grafo de enlaces

de la misma. En dicho grafo un enlace está unido a otro si ambos tienen un átomo en común. Los grafos de enlaces de los dos ejemplos presentados en esta sección se pueden ver en las figuras 3.8 y 3.9.

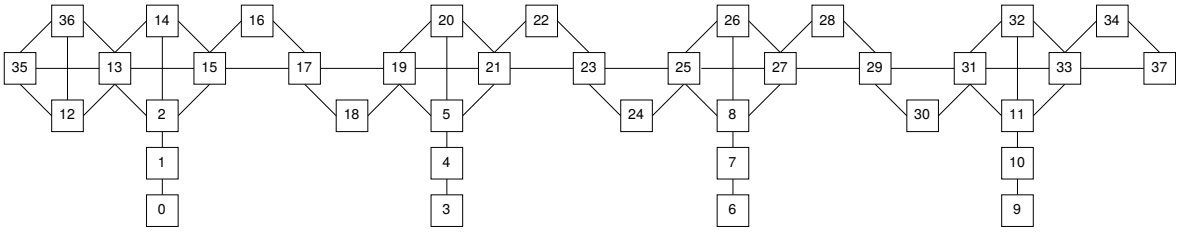

Figura 3.8: Grafo de enlaces de una cadena real con ramificaciones lineales idénticas de tres átomos.

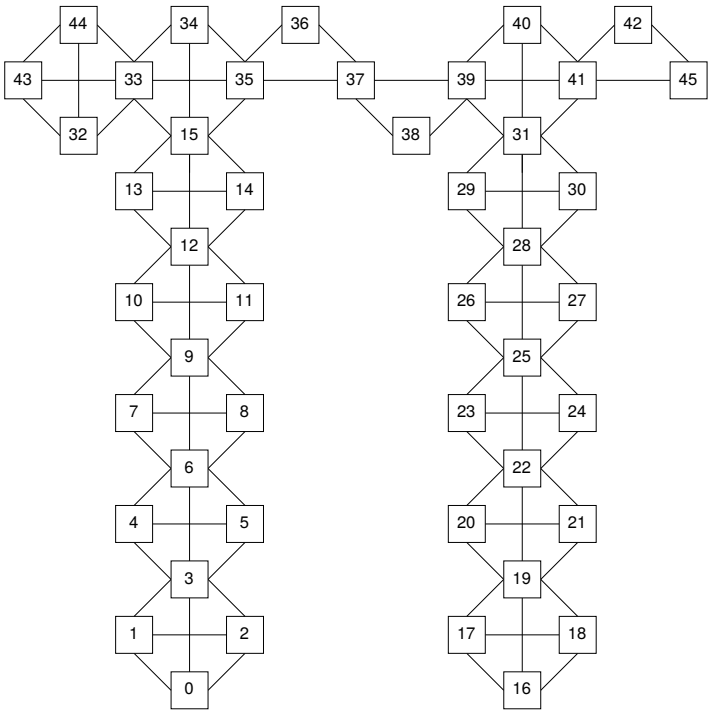

Figura 3.9: Grafo de enlaces de una cadena real con ramificaciones idénticas de lisina.

Aplicando las ecuaciones anteriormente presentadas a los ejemplos, se obtienen los sistemas de ecuaciones de las Figuras 3.10 y 3.11. La matriz que representa el sistema de ecuaciones se pueden dividir en varios conjuntos de submatrices:

1. Submatrices ramificación-a-ramificación (cuadrantes Ri-Ri en las figuras).
2. Submatrices ramificación-a-cadena principal (cuadrantes Ri-BBi en las figuras).
3. Submatrices ramificación-a-términos independientes (cuadrantes Ri-TI en las figuras).
4. Submatrices cadena principal-a-ramificación (cuadrantes BBi-Ri en las figuras).
5. Submatrices cadena principal-a-cadena principal (cuadrantes BBi-BBi en las figuras). Nótese que dos submatrices consecutivas de este tipo tienen algunas

columnas en común. Esta dependencia entre submatrices complica la resolución paralela del sistema de ecuaciones.

6. Submatrices cadena principal-a-especial izquierda (cuadrantes BBi-SL en las figuras).
7. Submatrices cadena principal-a-especial derecha (cuadrantes BBi-SR en las figuras).
8. Submatrices cadena principal-a-especial independientes (cuadrantes BBi-TI en las figuras).
9. Submatrices especial izquierda-a-cadena principal (cuadrantes SL-BBi en las figuras).
10. Submatrices especial izquierda-a-especial izquierda (cuadrantes SL-SL en las figuras).
11. Submatrices especial izquierda-a-términos independientes (cuadrantes SL-TI en las figuras).
12. Submatrices especial derecha-a-cadena principal (cuadrantes SR-BBi en las figuras).
13. Submatrices especial derecha-a-especial derecha (cuadrantes SR-SR en las figuras).
14. Submatrices especial derecha-a-términos independientes (cuadrantes SR-TI en las figuras).

Esta notación se usará en los siguientes apartados para explicar los pasos necesarios para resolver el sistema. Además, como se puede apreciar, todas las submatrices pertenecientes a un conjunto de submatrices son idénticas (exceptuando la primera y última submatriz del conjunto cadena principal-a-cadena principal). Estos patrones permiten paralelizar el algoritmo.

Los dos tipos de paralelismo, vectorial y multinúcleo, siempre se aplicarán a distintas submatrices. El paralelismo multinúcleo se consigue asignando varias submatrices a distintos núcleos, mientras que el paralelismo vectorial se consigue resolviendo varias submatrices al mismo tiempo usando vectorización en cada núcleo. La asignación de submatrices a núcleos es estática y se mantiene durante toda la resolución del sistema de ecuaciones.

La creación del sistema de ecuaciones (matriz de coeficientes y de términos independientes) es paralelizable pero no se puede aplicar vectorización. Cada núcleo poblará las entradas del sistema de ecuaciones que se le asignen. En la Figura 3.12 se puede ver la división de un sistema de ecuaciones de una cadena lineal con ramificaciones lineales para dos núcleos y dos elementos de longitud vectorial.

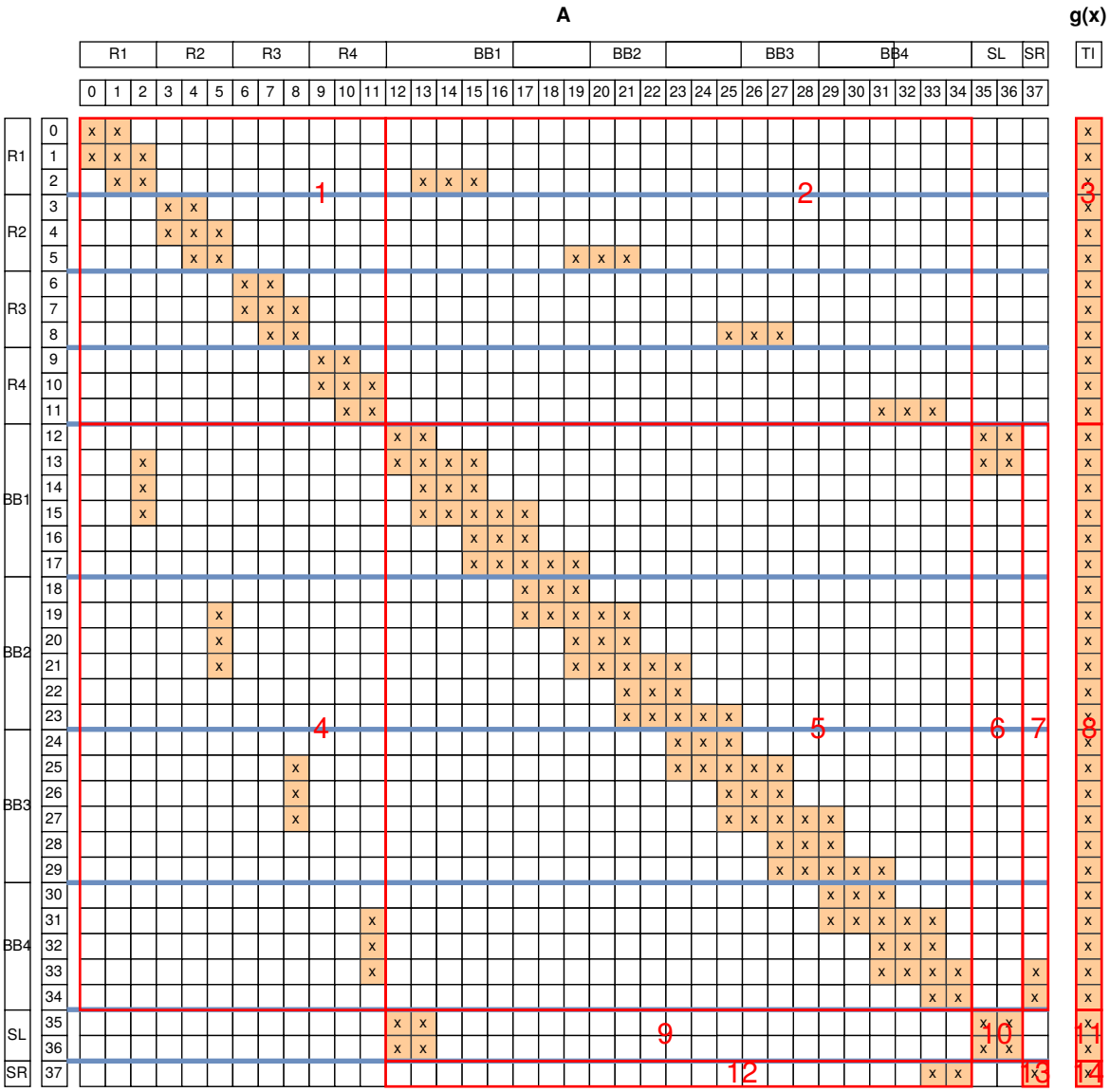

Figura 3.10: Sistema de ecuaciones de una cadena real con cuatro ramificaciones lineales idénticas de tres átomos. Cada tipo de submatriz se ha numerado siguiendo el listado de tipos de submatrices anteriormente presentado.

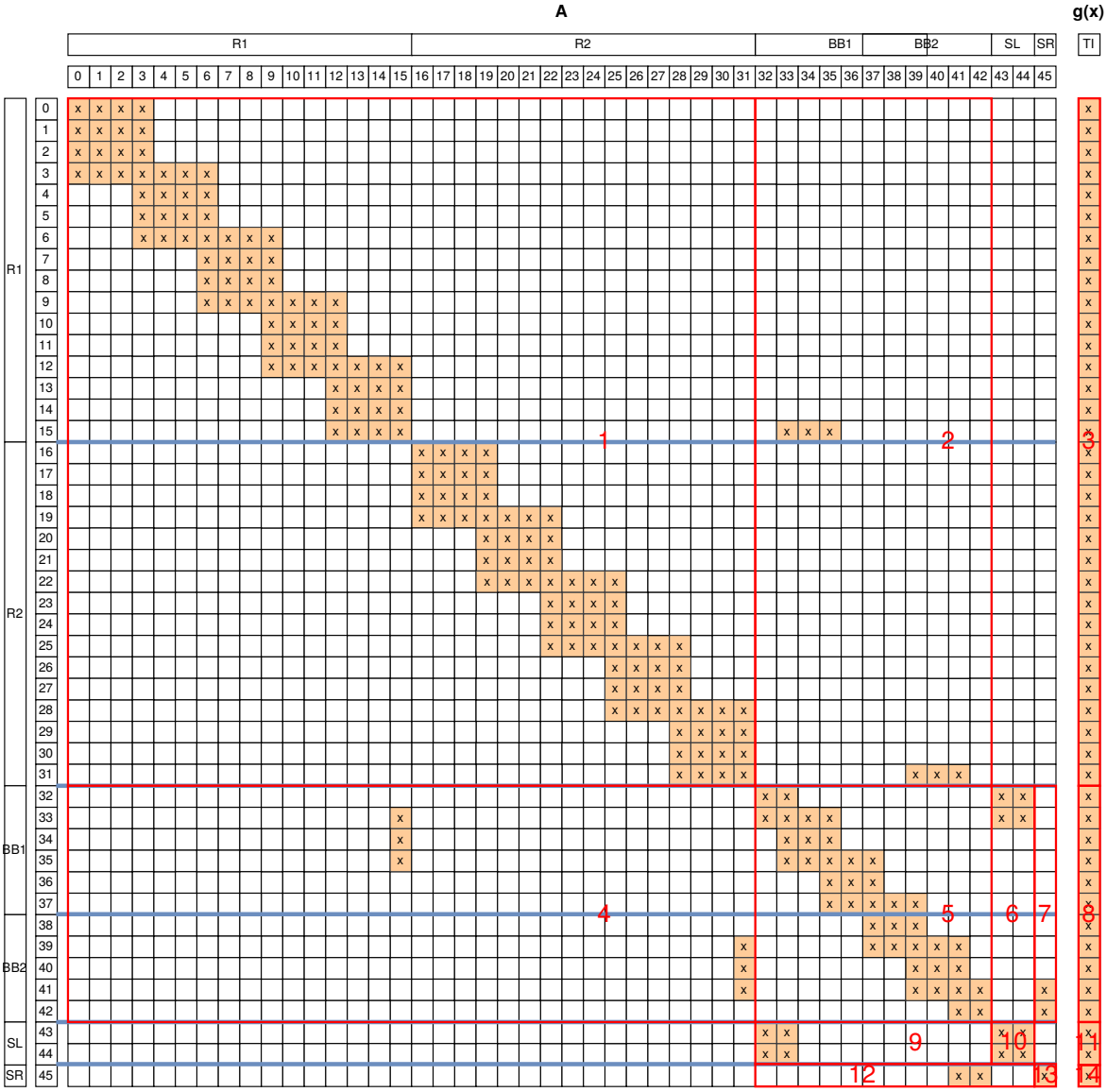

Figura 3.11: Sistema de ecuaciones de una cadena real con dos ramificaciones idénticas de lisina. Cada tipo de submatriz se ha numerado siguiendo el listado de tipos de submatrices anteriormente presentado.

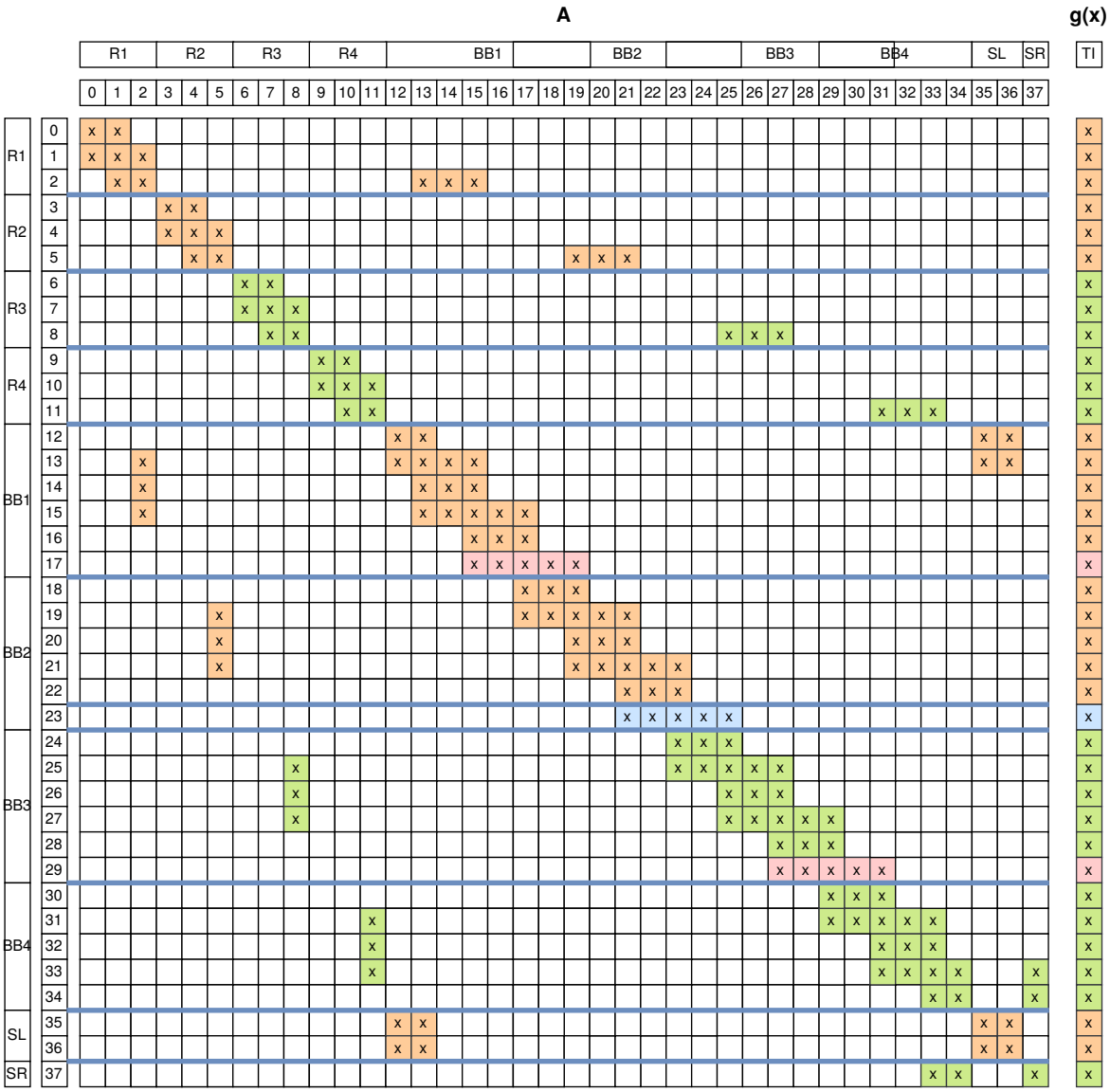

Figura 3.12: Sistema de ecuaciones de una cadena real con cuatro ramificaciones lineales idénticas de tres átomos dividida entre dos núcleos, uno identificado por el color naranja y otro por el color verde, cada uno de ellos con una longitud vectorial de dos elementos. Los colores azul y rosa identifican las filas Schur (explicadas más adelante).

3.2. Método del Complemento de Schur

El sistema de ecuaciones generado se debe resolver de forma paralela mediante eliminación Gaussiana. Este procedimiento paralelo plantea un problema a la hora de resolver las submatrices cadena principal-a-cadena principal, ya que existen dependencias entre ellas. Para solucionar esto se utiliza el método del complemento de Schur. Su aplicación trata a submatrices que son dependientes como si fueran independientes, separándolas por una fila compartida que se tratará de distinta forma que al resto, la fila Schur. Un ejemplo de esta división se puede ver aplicada a una matriz tri-diagonal en la Figura 3.13.

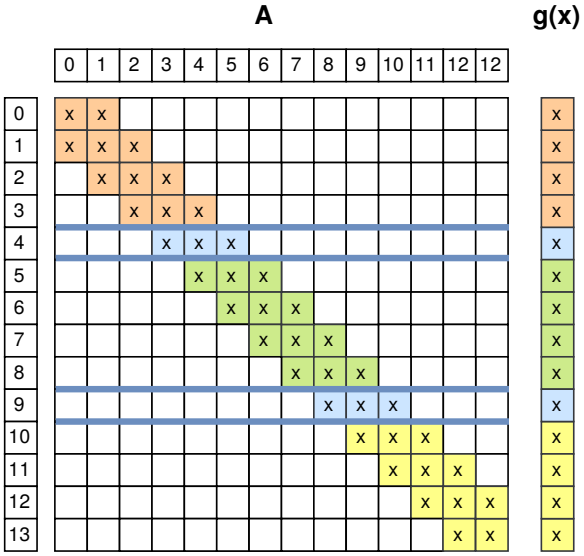

Figura 3.13: Sistema de ecuaciones (matriz de coeficientes y de términos independientes) dividido en tres (naranja, verde y amarillo). Las dos filas azules representan las filas Schur.

El objetivo es hallar la matriz identidad, para ello se comienza eliminando al mismo tiempo los elementos que están debajo de la diagonal. En la Figura 3.14 se parte del estado inicial, anteriormente presentado, y se muestra el resultado de aplicar operaciones elementales al mismo tiempo a cada submatriz (naranja, verde y amarillo) hasta obtener 'ceros' por debajo de la diagonal y 'unos' en esta. Este paso hace que aparezca una nueva columna de elementos no nulos (*fill-ins*) a la izquierda de la diagonal.

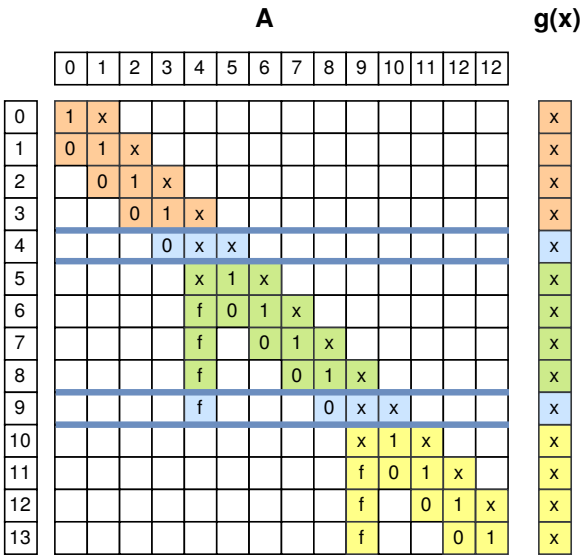

Figura 3.14: Sistema de ecuaciones resultante tras aplicar operaciones elementales para hacer 'ceros' debajo de la diagonal y 'unos' en esta, partiendo del sistema de ecuaciones de la Figura 3.13. Los elementos designados con una f representan elementos que anteriormente eran nulos (*fill-ins*).

Aplicando operaciones elementales se eliminan los elementos que están encima de la diagonal, haciendo aparecer, de la misma manera que antes, una columna de nuevos elementos (*fill-ins*), a la derecha de la diagonal (Figura 3.15). Este paso se puede estar ejecutando al mismo tiempo que el anterior se ejecuta en otro núcleo, por lo que es necesario que los accesos a las filas Schur se hagan en exclusión mutua.

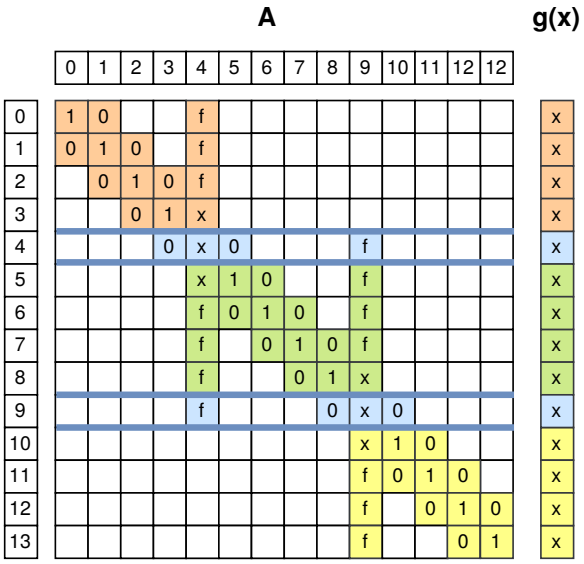

Figura 3.15: Sistema de ecuaciones resultante tras aplicar operaciones elementales para hacer 'ceros' encima de la diagonal a la matriz de coeficientes de la Figura 3.14.

Una vez aplicados los anteriores pasos se puede pasar a resolver la matriz que forman el conjunto de las filas Schur (matriz Schur). Este paso se puede resolver

secuencialmente, con cualquier método de resolución de sistemas de ecuaciones o aplicando, recursivamente, la técnica del complemento de Schur de nuevo. (Figura 3.16).

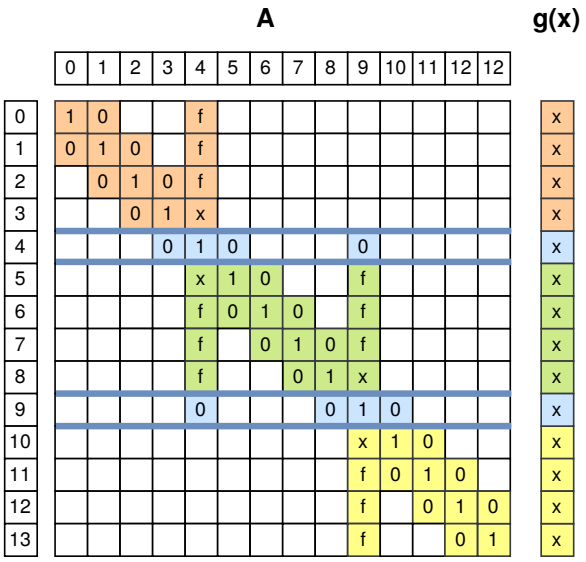

Figura 3.16: Sistema de ecuaciones resultante tras resolver la matriz formada por el conjunto de las filas Schur de la Figura 3.15

Por último se pueden eliminar al mismo tiempo los *fill-ins* que han aparecido en anteriores pasos aplicando operaciones elementales (Figura 3.17).

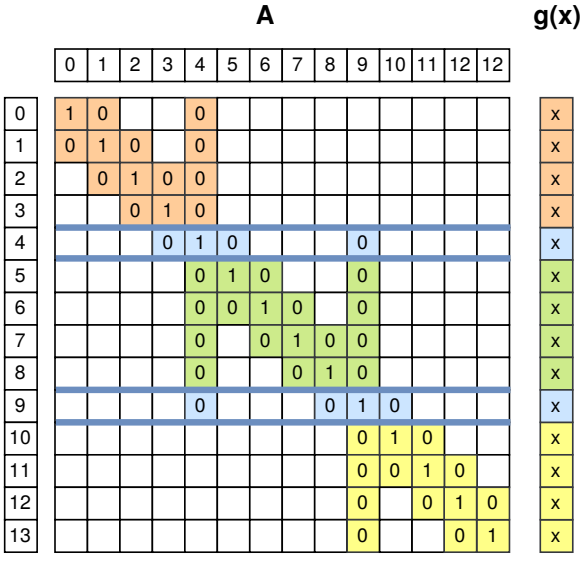

Figura 3.17: Sistema de ecuaciones resultante tras eliminar los *fill-ins* usando operaciones elementales partiendo del sistema de ecuaciones de la Figura 3.16.

El uso de esta técnica en la implementación de ILVES permite aplicar paralelismo, tanto vectorial como multinúcleo, a la resolución de las submatrices cadena principal-a-cadena principal. El número de filas Schur usado en la implementación de ILVES depende de dos factores: la longitud vectorial ($VLEN$) y el número de núcleos ($NTHREADS$).

Además se distinguen entre dos tipos de filas Schur: las filas Schur privadas (propias de cada núcleo), que permiten el paralelismo vectorial y de las que se requieren $VLEN - 1$ filas por núcleo; y las filas Schur compartidas (accedidas por todos los núcleos en exclusión mutua), que permiten el paralelismo multinúcleo y de las que se requieren $NTHREADS - 1$ filas en total.

3.3. Resolución del sistema de ecuaciones

El orden en el que se eliminan los elementos de la matriz de coeficientes aplicando eliminación Gaussiana es de vital importancia. Usar un orden distinto al presentado en esta sección haría que aparecieran elementos no nulos adicionales durante la resolución del sistema de ecuaciones, lo que conllevaría una pérdida de rendimiento. Por conveniencia, se usará el sistema de ecuaciones de la Figura 3.12, perteneciente a una cadena con ramificaciones lineales idénticas, para explicar todos los pasos de ILVES. Los pasos que se deben aplicar a un sistema de ecuaciones de una cadena real con ramificaciones idénticas de lisina son exactamente los mismos, cambiando únicamente la estructura de las submatrices ramificación-a-ramificación. Salvo los pasos pertenecientes a la resolución de la matriz Schur, todos los pasos presentados se resuelven explotando tanto paralelismo vectorial como multinúcleo. Las filas Schur privadas, accedidas por un solo núcleo, se identifican con el color rosa y las filas Schur compartidas, accedidas en exclusión mutua por varios núcleos, con el color azul. Para los ejemplos se supondrá que se cuenta con dos núcleos y dos elementos de longitud vectorial.

Los dos primeros pasos (Figura 3.18) consisten en simplificar las submatrices especial izquierda-a-especial izquierda y especial derecha-a-especial derecha hasta obtener la matriz identidad y, usando dichas submatrices, hacer 'ceros' en las entradas de las submatrices cadena principal-a-especial izquierda y cadena principal-a-especial derecha. De esta forma, todas las submatrices cadena principal-a-{especial izquierda, especial derecha} son exactamente iguales, evitando dividir la resolución de estas en distintos casos. Nótese que al aplicar estos dos pasos, como efecto colateral, se modifican algunas entradas de otras submatrices. Esto ocurrirá a lo largo de todo el procedimiento.

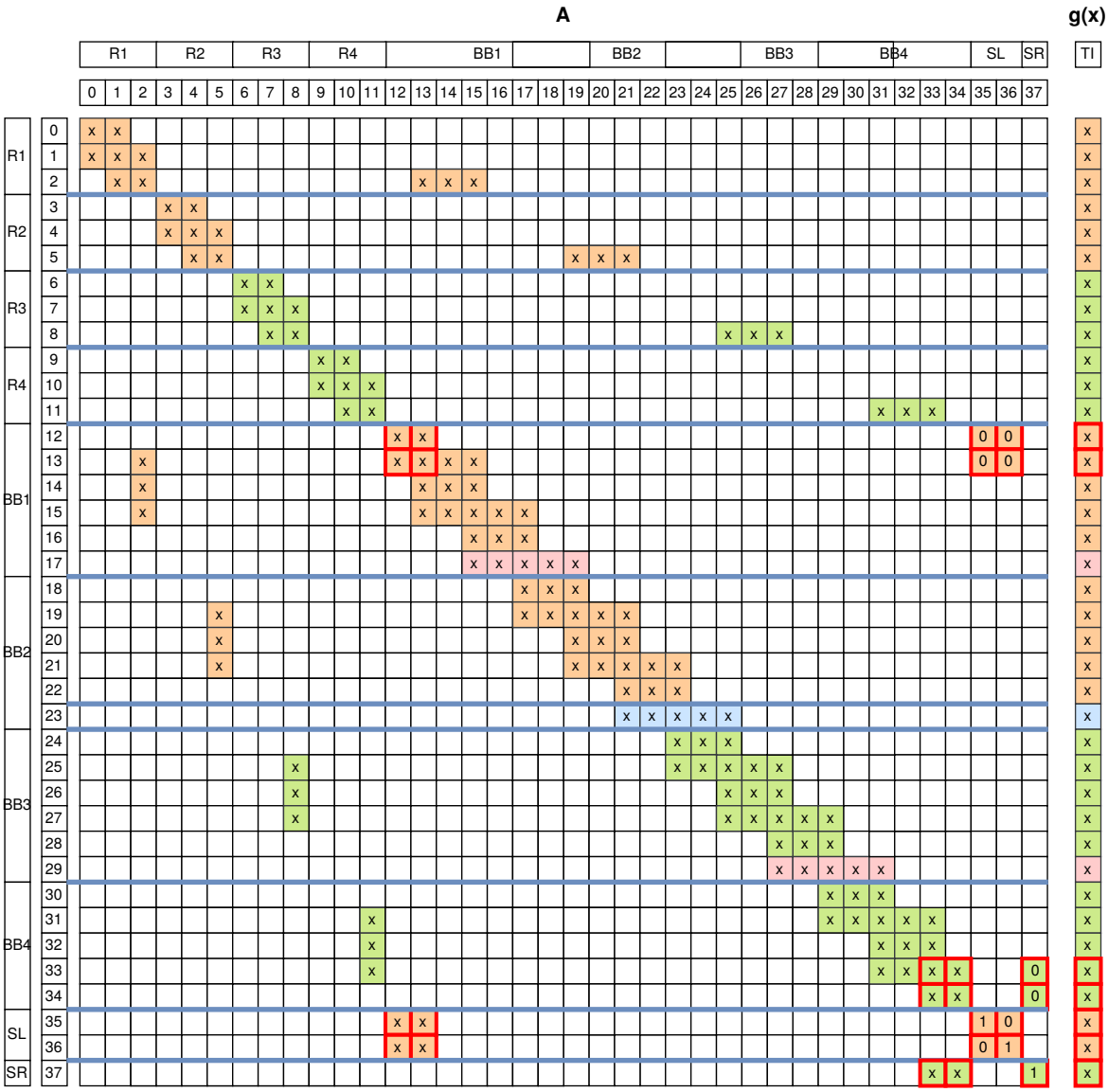

Figura 3.18: Pasos 1 y 2 de ILVES. Obtención de la matriz identidad en las submatrices especial izquierda-a-especial izquierda y especial derecha-a-especial derecha y eliminación de las entradas de las submatrices cadena principal-a-{especial izquierda, especial derecha} (partiendo de la Figura 3.12). Las entradas modificadas en estos pasos se rodean en rojo.

En el siguiente paso (Figura 3.19) se limpian los elementos sub-diagonales de las submatrices ramificación-a-ramificación, además de aplicar transformaciones elementales a la diagonal de dichas submatrices para obtener 'unos' en todas sus entradas.

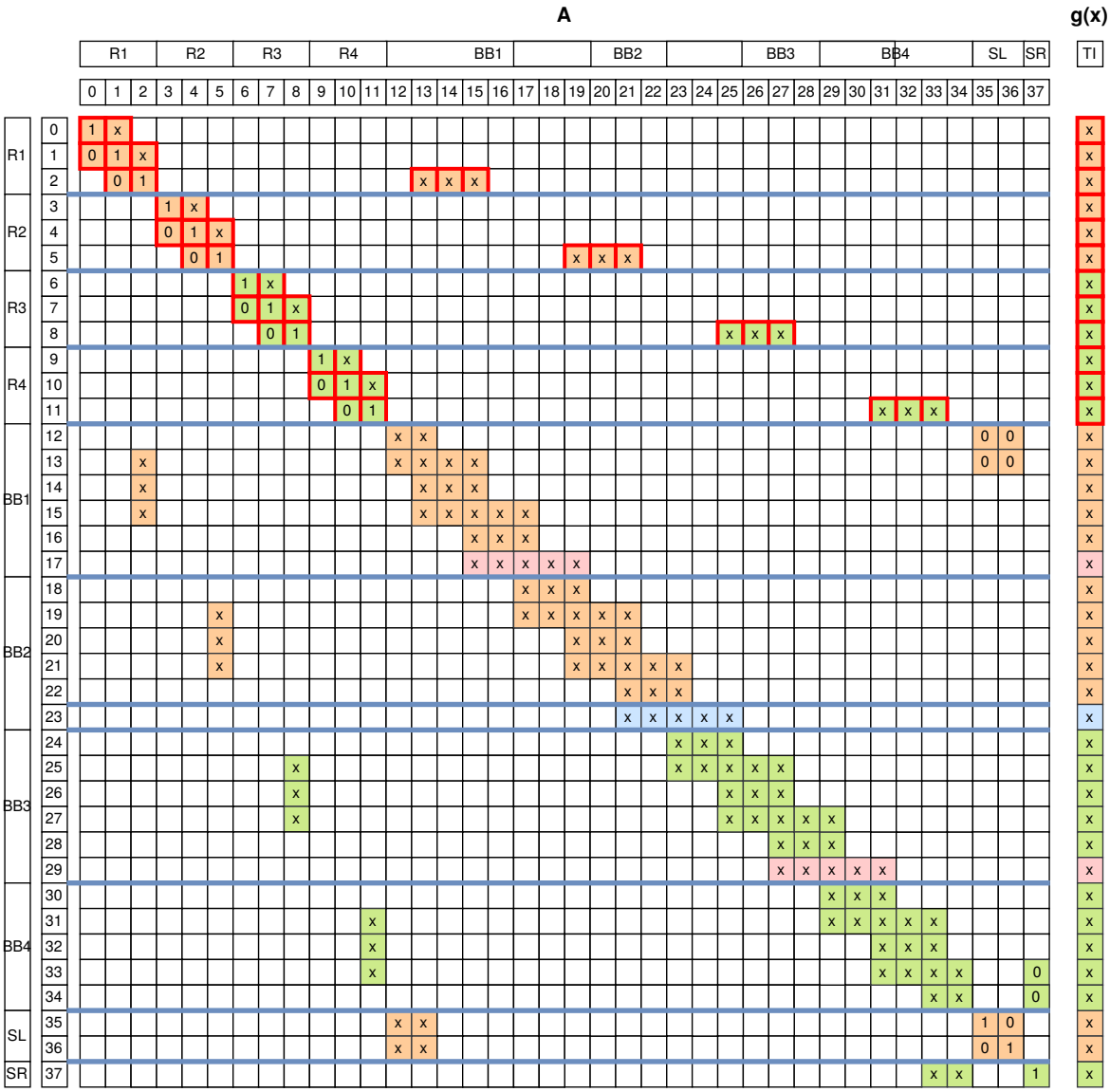

Figura 3.19: Paso 3 de ILVES. Eliminación de los elementos de la subdiagonal de las submatrices ramificación-a-ramificación y transformaciones elementales para obtener 'unos' en la diagonal de dichas submatrices (partiendo de la Figura 3.18).

Lo siguiente es eliminar todas las entradas de las submatrices cadena principal-a-ramificación usando la última fila de cada submatriz ramificación-a-ramificación (Figura 3.20). Este paso evita que aparezcan *fill-ins* adicionales a la hora de resolver los siguientes pasos.

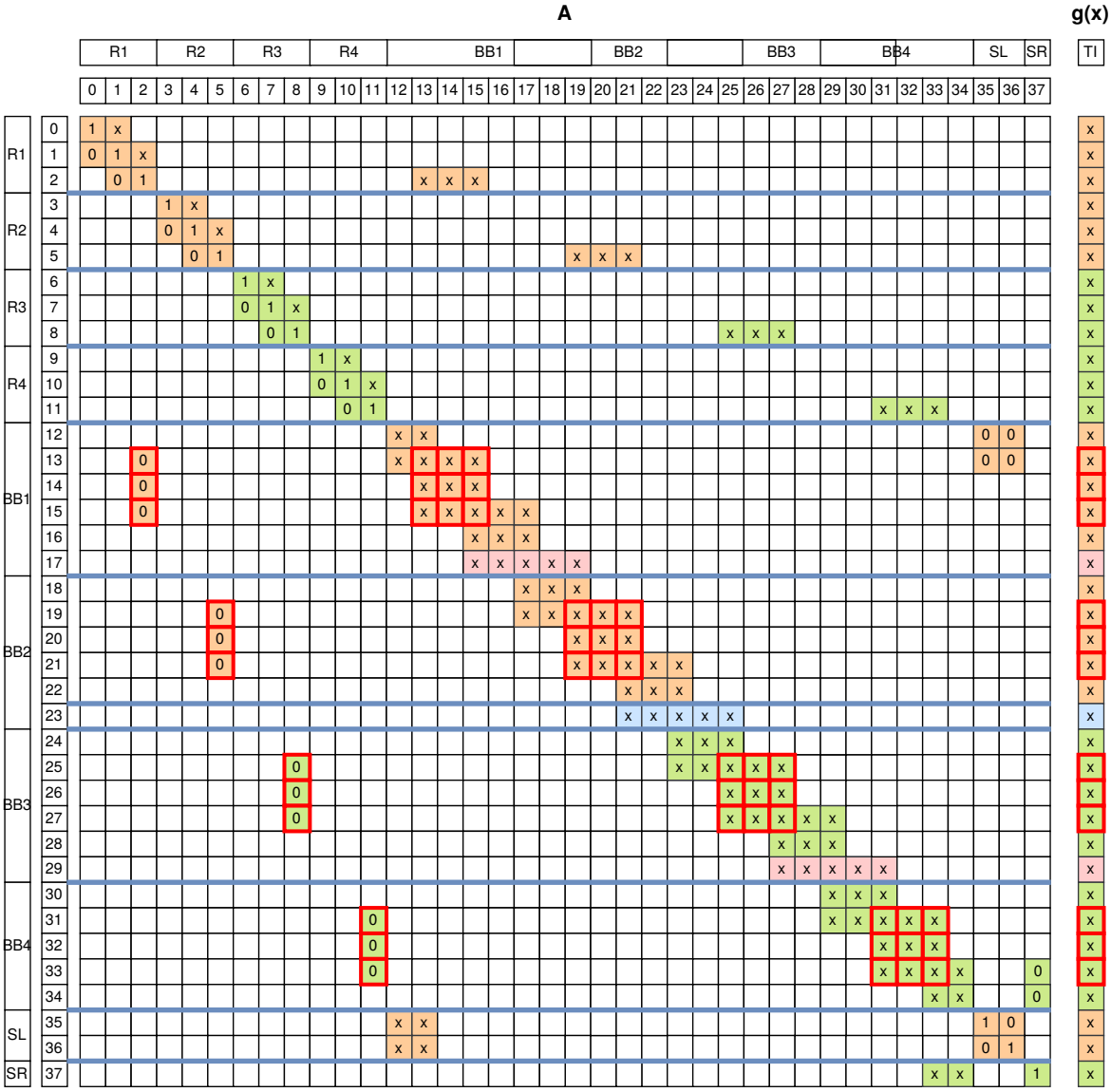

Figura 3.20: Paso 4 de ILVES. Eliminación de todos los elementos de las submatrices cadena principal-a-ramificación (partiendo de la Figura 3.19).

Los siguientes seis pasos resuelven las submatrices cadena principal-a-cadena principal haciendo uso del método del complemento de Schur anteriormente explicado. Primero se eliminan los elementos de la sub-diagonal y se hacen 'unos' en la diagonal de cada submatriz (Figura 3.21). A continuación se eliminan las entradas de la super-diagonal (Figura 3.22). Estos dos pasos hacen aparecer elementos no nulos adicionales (*fill-ins*) encima y debajo de la diagonal de cada submatriz. En tercer lugar se resuelve la matriz formada por el conjunto de filas Schur, tanto privadas como compartidas (Figura 3.23). La resolución de la matriz Schur la realiza un solo núcleo, sin aplicar vectorización. Por último se eliminan todos los *fill-ins* usando las filas Schur (Figura 3.24).

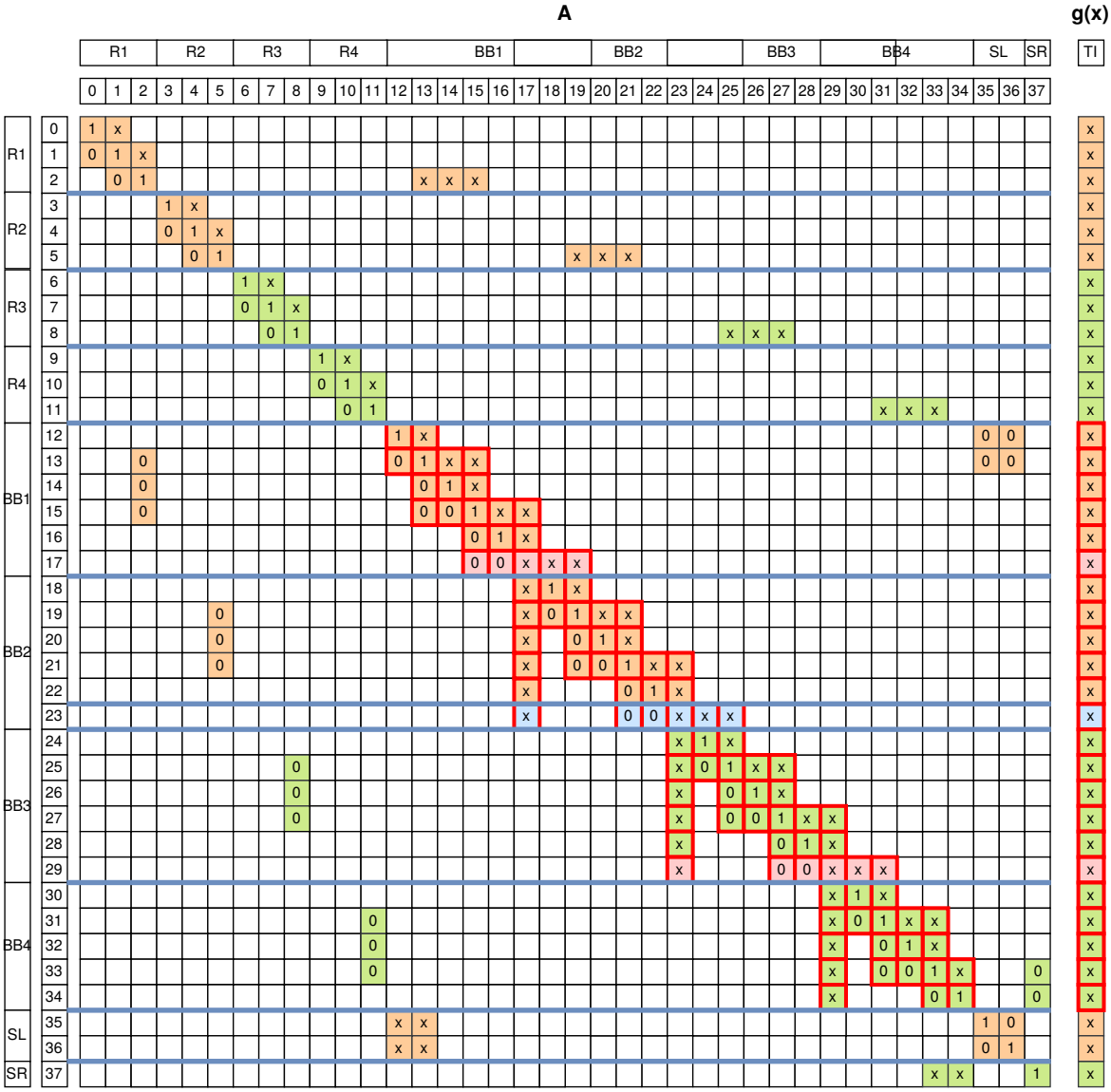

Figura 3.21: Paso 5 de ILVES. Eliminación de los elementos de la subdiagonal de las submatrices cadena principal-a-cadena principal y transformaciones elementales para obtener 'unos' en la diagonal de dichas submatrices (partiendo de la Figura 3.20).

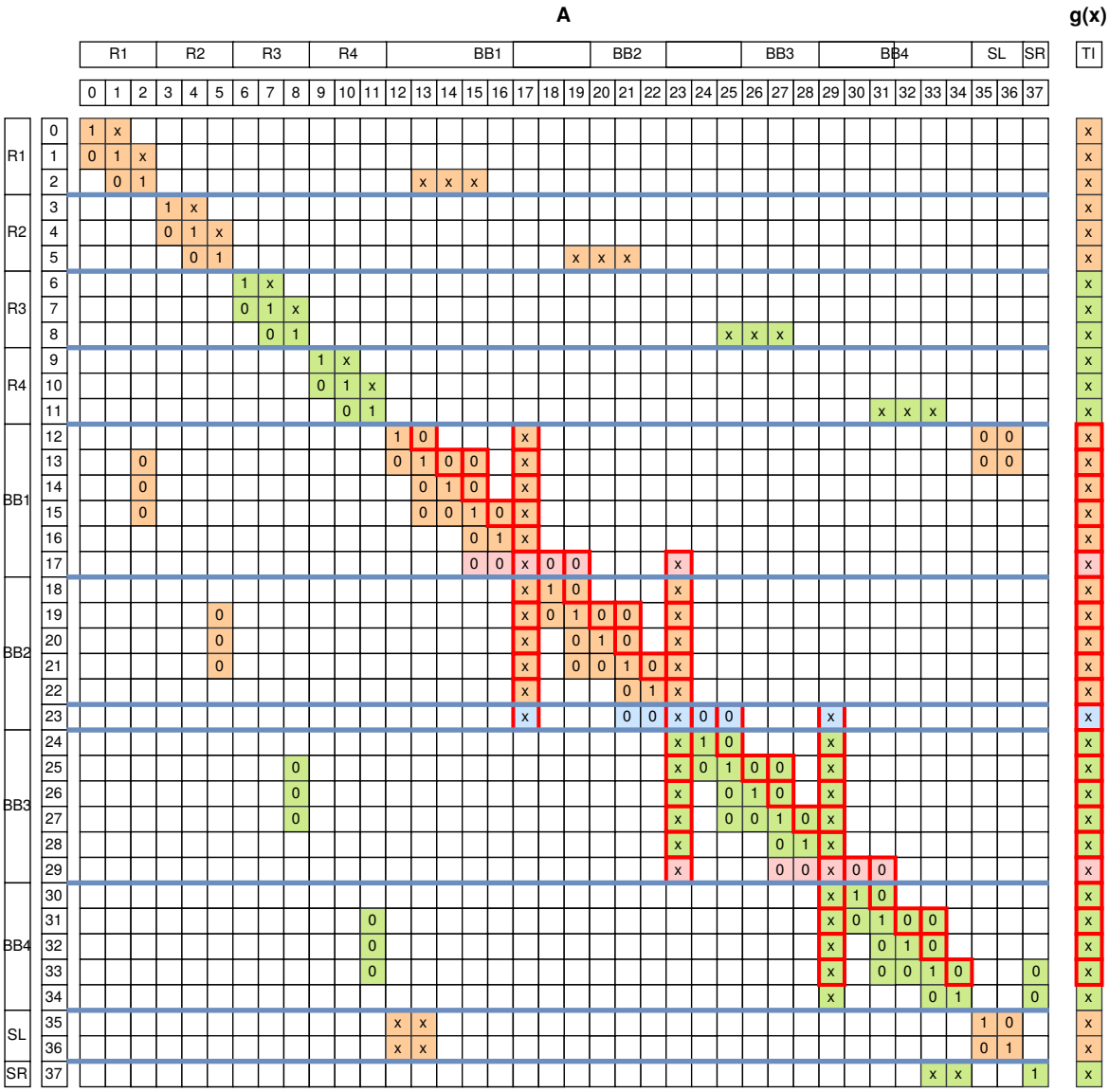

Figura 3.22: Paso 6 de ILVES. Eliminación de los elementos de la super-diagonal de las submatrices cadena principal-a-cadena principal (partiendo de la Figura 3.21).

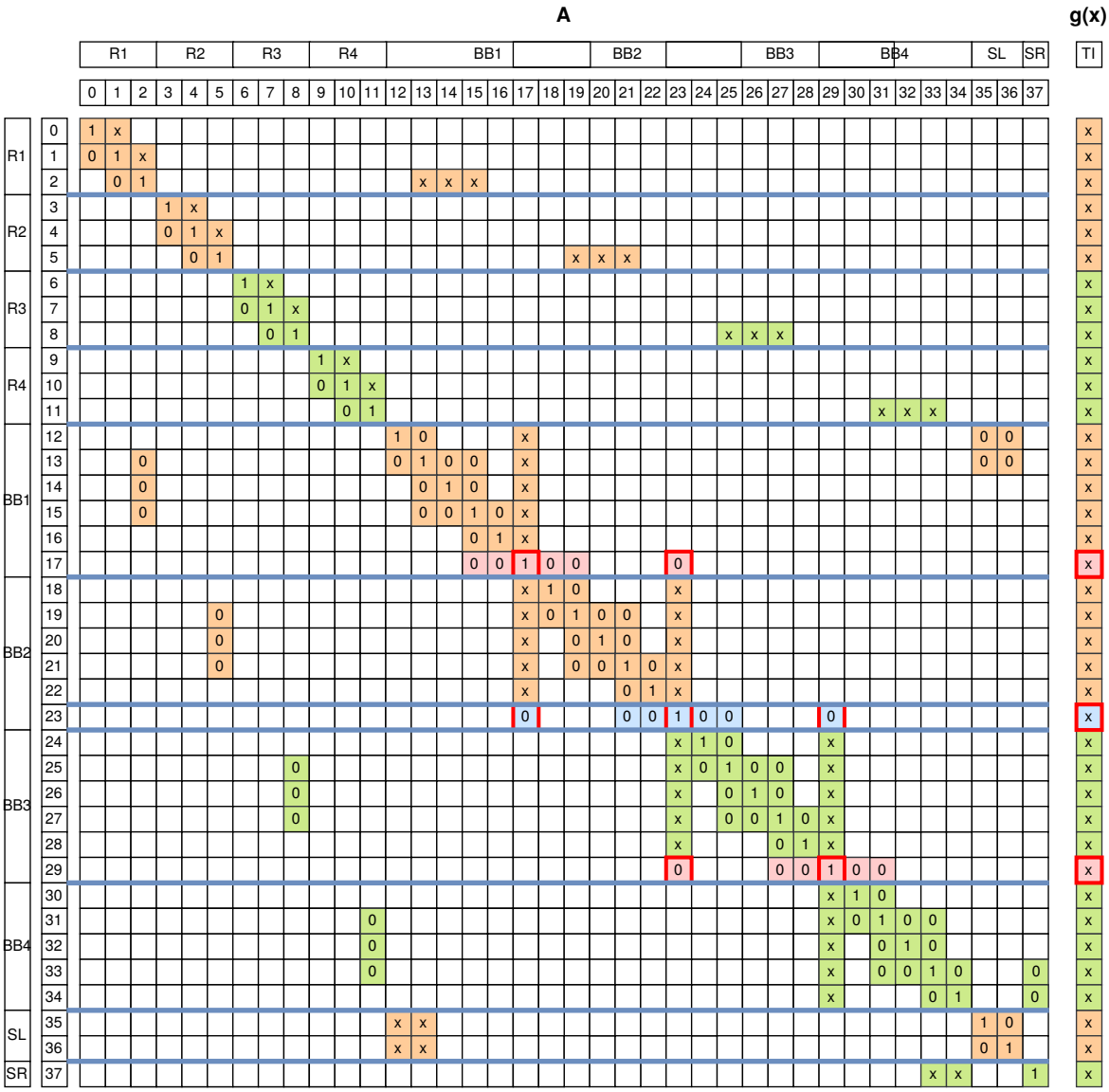

Figura 3.23: Pasos 7 y 8 de ILVES. Obtención de la matriz identidad en la matriz formada por el conjunto de filas Schur (partiendo de la Figura 3.22).

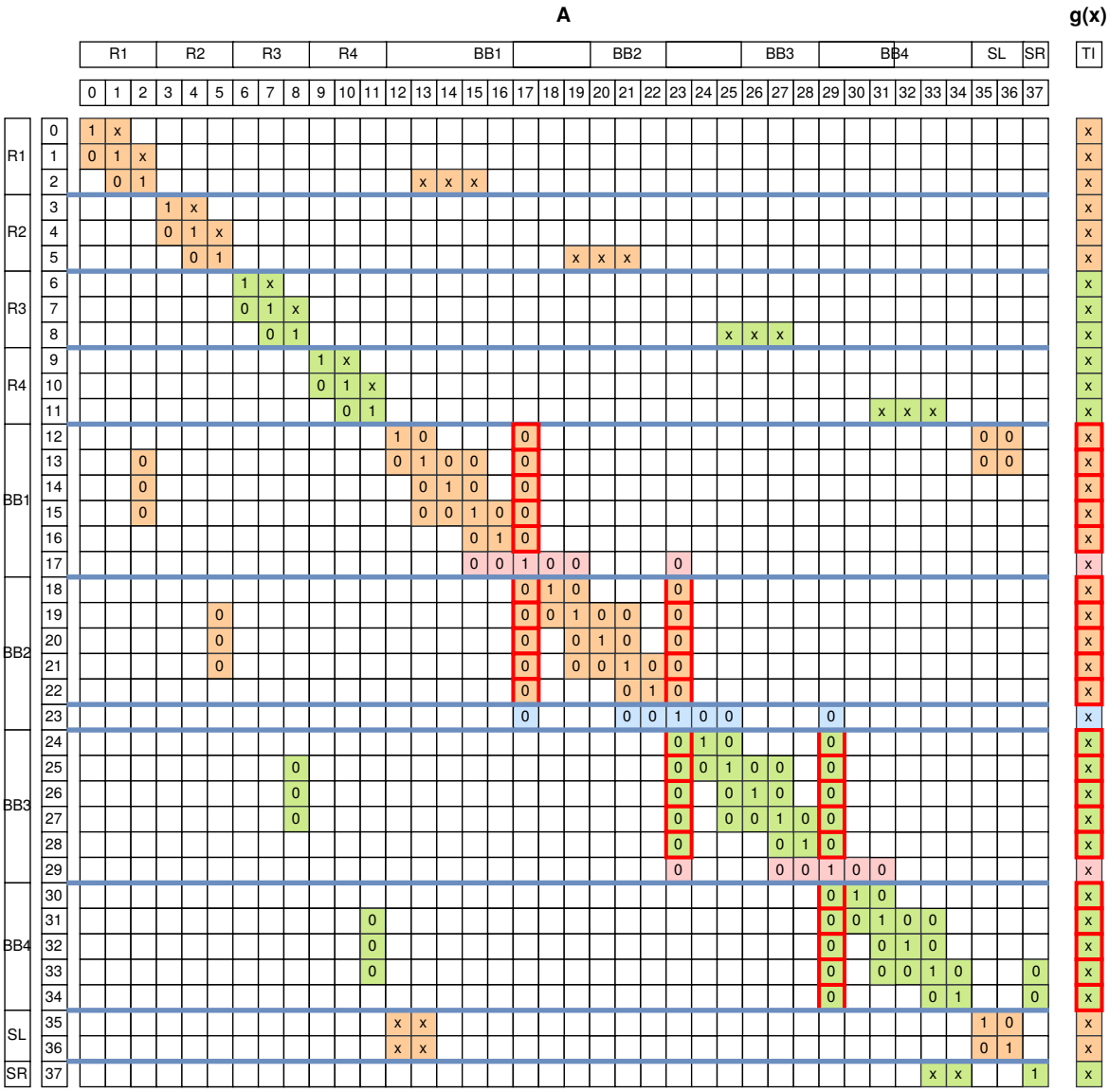

Figura 3.24: Pasos 9 y 10 de ILVES. Eliminación de los *fill-ins* (partiendo de la Figura 3.23).

Una vez resueltas las submatrices cadena principal-a-cadena principal se pueden eliminar las entradas de las submatrices ramificación-a-cadena principal (Figura 3.25).

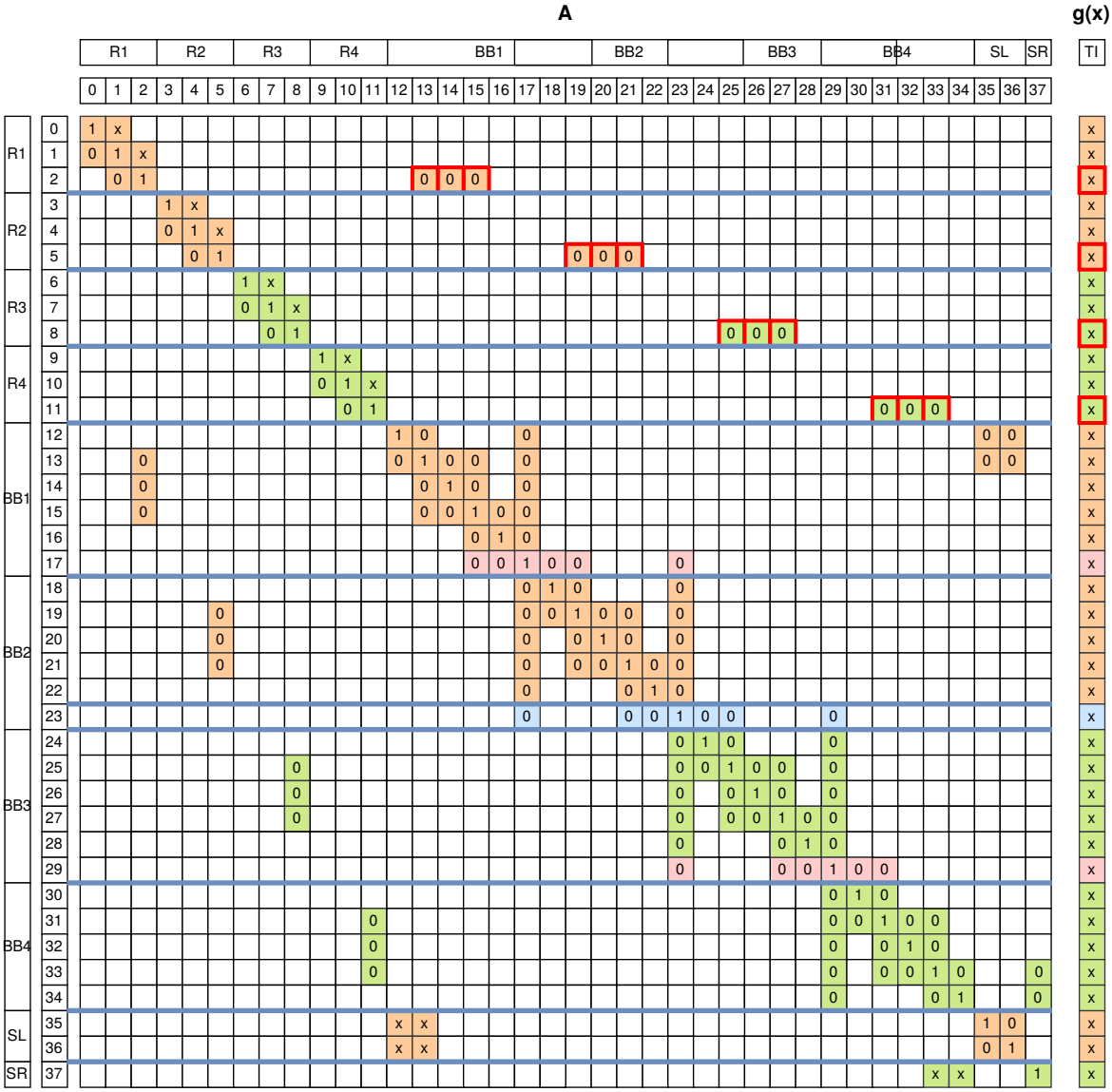

Figura 3.25: Paso 11 de ILVES. Eliminación de los elementos de las submatrices ramificación-a-cadena principal (partiendo de la Figura 3.24).

Lo siguiente es limpiar los elementos de la super-diagonal de las submatrices ramificación-a-ramificación (Figura 3.26).

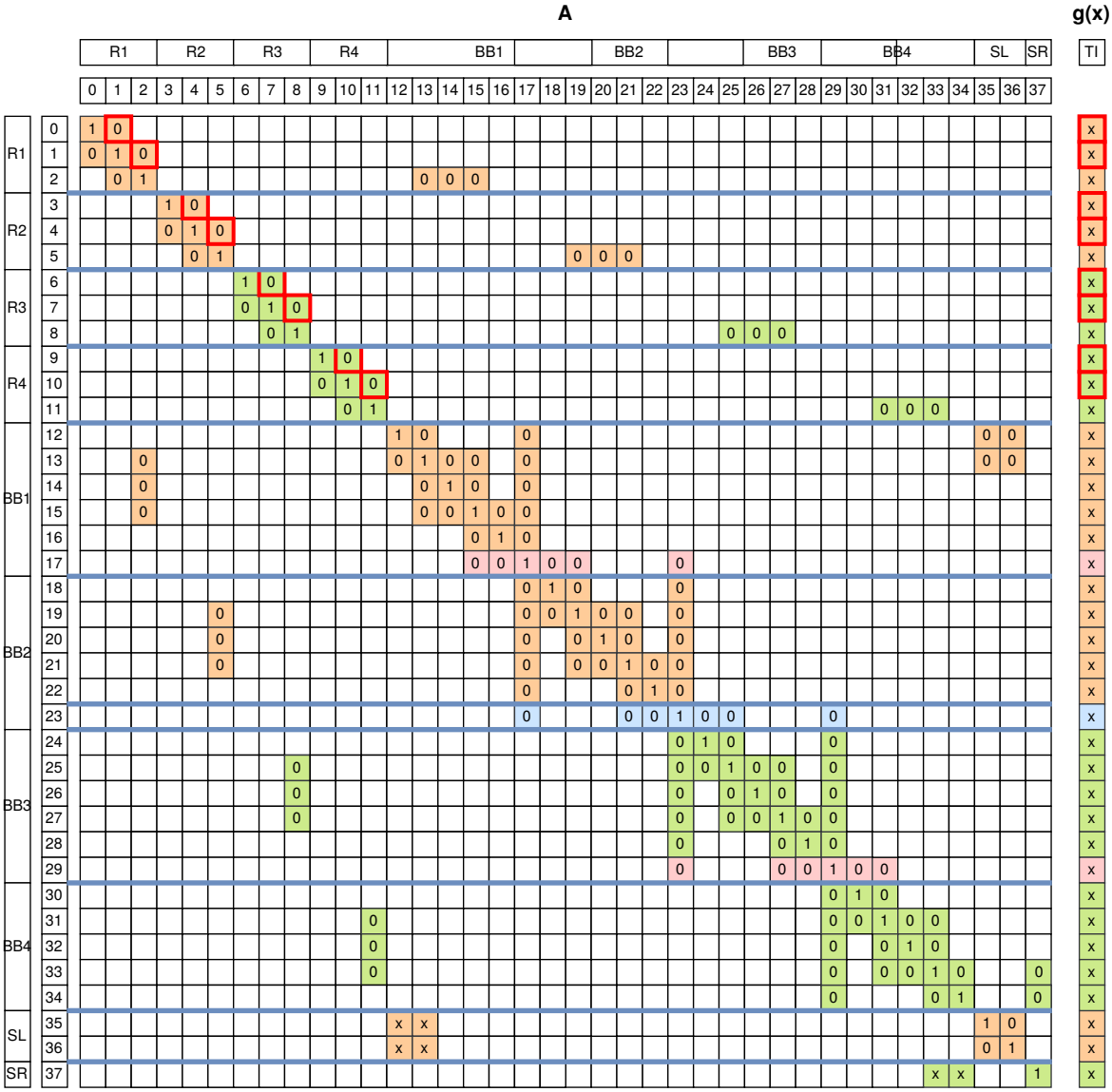

Figura 3.26: Paso 12 de ILVES. Eliminación de los elementos super-diagonales de las submatrices ramificación-a-ramificación (partiendo de la Figura 3.25).

Los últimos dos pasos consisten en eliminar las entradas restantes de las submatrices {especial izquierda, especial derecha}-a-cadena principal (Figura 3.27).

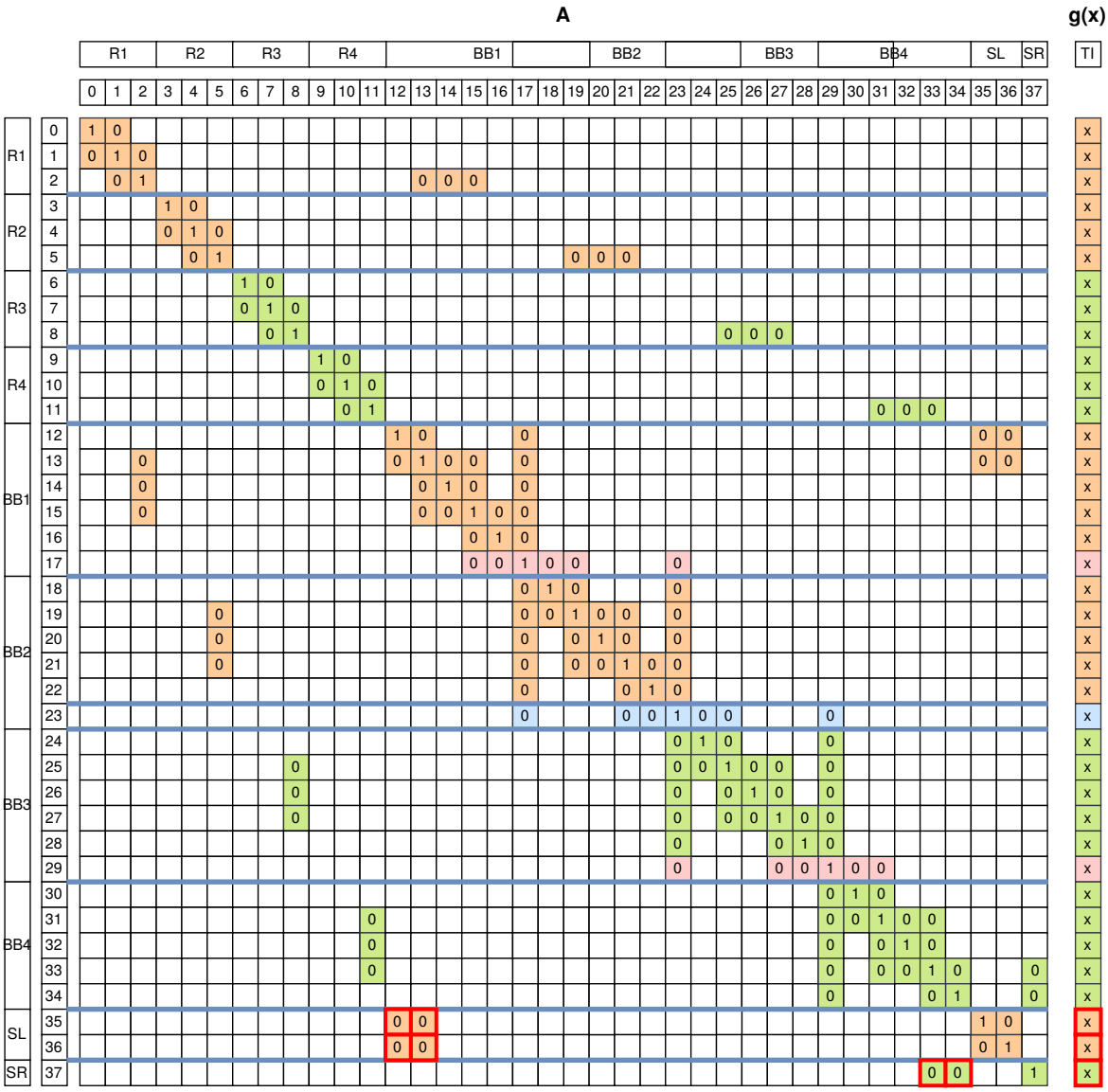

Figura 3.27: Pasos 13 y 14 de ILVES. Eliminación de los elementos restantes de las submatrices especial izquierda-a-cadena principal y especial derecha-a-cadena principal (partiendo de la Figura 3.26).

3.4. Corrección de posiciones

Una vez resuelto el sistema de ecuaciones se deben aplicar las correcciones a las posiciones de los átomos. La actualización de la posición de cada átomo se calcula con la siguiente ecuación [7]:

$$x = x - Dg(x)^T \cdot g(x) \tag{3.3}$$

donde x es el vector de posición (x_x, x_y, x_z) de cada átomo, $Dg(x)$ es el Jacobiano $Dg(x) \in \mathbb{R}^{M \times 3N}$ y $g(x)$ es la matriz de términos independientes del sistema, la cual contiene el resultado del sistema de ecuaciones.

Una vez aplicada la corrección de posiciones se puede calcular el error relativo de un enlace k mediante la siguiente ecuación [7]:

$$e_k = \frac{1}{2} \frac{||x_{a_k} - x_{b_k}||_2^2 - (d_{a_k,b_k})^2}{(d_{a_k,b_k})^2} \tag{3.4}$$

donde d_{a_k,b_k} es la longitud del enlace atómico formado por los átomos a_k y b_k y x es el vector de posición (x_x, x_y, x_z) de cada átomo del enlace k. El error de un paso temporal es el máximo de los errores relativos de todas las ligaduras. La tolerancia se define como el máximo error permitido. Para reducir el error máximo se deberá volver a ejecutar el algoritmo iterativamente, hasta que el error máximo sea menor a la tolerancia.

Este paso no es vectorizable, pero sí puede usar paralelismo multinúcleo, manteniendo la división estática por submatrices realizada a la hora de construir la matriz.

3.5. Organización de datos

Para que un bucle sea vectorizable los datos con los que opera cada una de sus instrucciones deben estar contiguos en memoria (Anexo A). La mayoría de procesadores soportan instrucciones vectoriales que operan con datos no contiguos en memoria, pero son más lentas que si accedieran a datos contiguos. Por esta razón la estructura de datos que almacena el sistema de ecuaciones no es trivial. A la hora de construir la matriz se requiere hacer una reordenación de datos para conseguir que las instrucciones vectoriales siempre accedan a datos contiguos en memoria. Esta ordenación es dependiente de la longitud vectorial.

La matriz de coeficientes del sistema de ecuaciones está compuesta mayormente por entradas nulas, es por esto que dicha matriz se almacena de forma dispersa, lo que permite ahorrar memoria y aumentar el rendimiento.

Se utilizan varias estructuras de datos para almacenar el sistema de ecuaciones. Cada núcleo tiene un vector para cada una de las submatrices del sistema. Además se dispone de $NTHREADS - 1$ vectores adiciones para almacenar los datos de las filas Schur compartidas, cada uno de estos vectores está protegido por un *mutex*. Para prevenir el fenómeno de *false sharing* se añade un relleno adicional al final de cada

vector (*padding*), cuyo tamaño es dependiente de la arquitectura sobre la que se compila ILVES. En la Figura 3.28 se puede ver la forma en la que se almacenan las entradas del sistema de ecuaciones que se ha estado usando para explicar los pasos de ILVES.

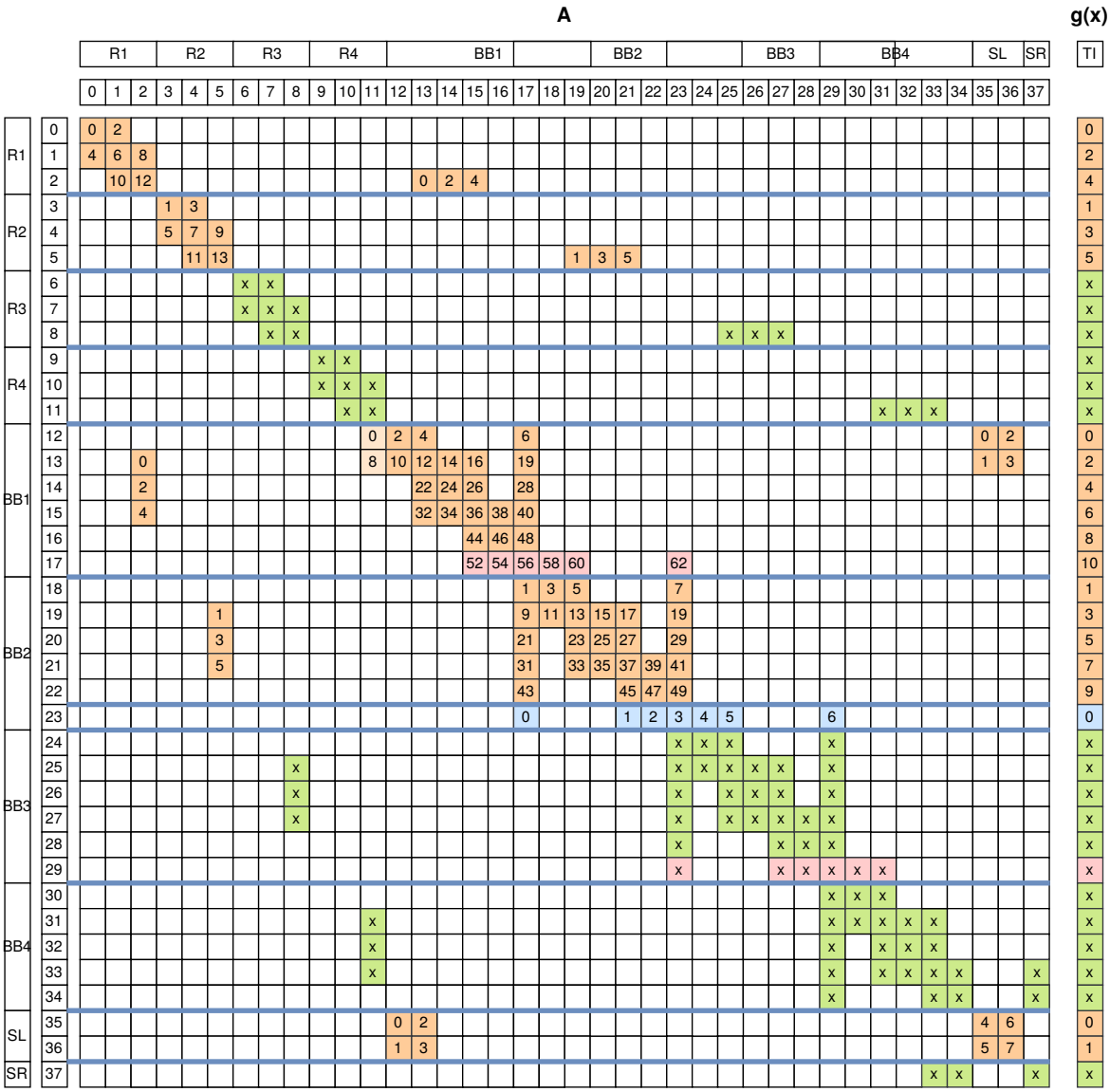

Figura 3.28: Organización de datos del primer núcleo del sistema de ecuaciones de la Figura 3.12 para una longitud vectorial de dos elementos. Los elementos numerados consecutivamente son elementos contiguos en memoria (en cada conjunto de submatrices distinto se vuelve a empezar a contar desde cero). Nótese que algunas entradas (como la [12, 11] o la [13, 11], marcadas en un color más claro), pese a ser nulas, deben estar presentes en la estructura de datos para mantener el patrón estructural. Las filas Schur compartidas se almacenan en vectores separados protegidos por *mutexes*, mientras que las filas Schur privadas se almacenan con el resto de datos del núcleo.

3.6. Análisis de rendimiento

A lo largo del proyecto se han usado los analizadores de rendimiento Valgrind [10] y Perf [11] para identificar los cuellos de botella de la implementación. A fecha de finalización del proyecto la mayor cantidad de recursos se invierte en construir la matriz de coeficientes (*make_matrix_ilves()*), en construir la matriz de términos independientes (*evaluate_constraint_function()*) y en corregir las posiciones de los átomos (*apply_jacobian()*), como se puede ver en la Figura 3.29 (resultados de Valgrind) y en la Tabla 3.1 (resultados de Perf).

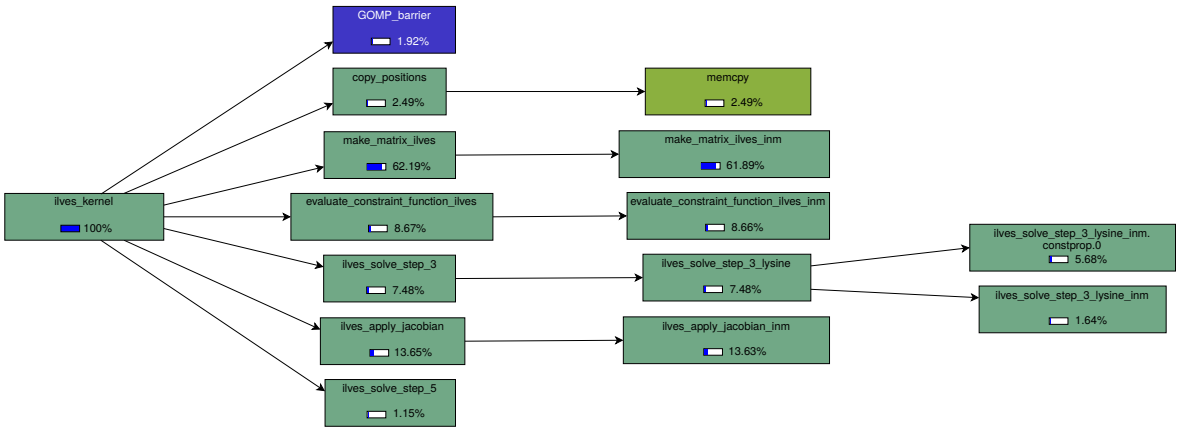

Figura 3.29: Grafo de porcentaje de ciclos de procesador invertidos por cada paso de ILVES según Valgrind. Las funciones que tienen un impacto menor al 1 % de ciclos no se muestran en el esquema.

Estas tres funciones son además las únicas que no explotan el paralelismo vectorial. En futuros trabajos se deberá analizar exhaustivamente dichas funciones para intentar mejorar su implementación y reducir el cuello de botella que ocasionan.

Función	**Ciclos (%)**
make_matrix_ilves	63,84 %
apply_jacobian	13,79 %
evaluate_constraint_function	8,71 %
ilves_solve_step_3	7,52 %
copy_positions	2,51 %
ilves_solve_step_5	1,17 %
ilves_solve_step_12	1,00 %
Ilves_solve_step_6	0,43 %
ilves_solve_step_4	0,39 %
ilves_solve_step_9	0,33 %
ilves_solve_step_10	0,15 %
ilves_solve_step_11	0,10 %
ilves_solve_step_7	0,01 %
ilves_solve_step_8	0,01 %
ilves_solve_step_1	0,01 %
ilves_solve_step_2	0,01 %
ilves_solve_step_14	0,01 %
ilves_solve_step_13	0,01 %

Tabla 3.1: Porcentaje de ciclos de procesador invertidos por cada paso de ILVES según Perf.

Capítulo 4

Resultados

En este capítulo se presentan los resultados de las pruebas ejecutadas para evaluar el rendimiento de la implementación de ILVES realizada durante el proyecto. Se comienza explicando la metodología utilizada para ejecutar los experimentos. En posteriores secciones se muestran los resultados de los experimentos ejecutados. Primero se hace una evaluación del rendimiento conseguido aplicando los distintos tipos de paralelismo utilizados en la implementación de ILVES. A continuación se compara el rendimiento de ILVES con el de SHAKE. Por último se compara la implementación de ILVES presentada en este proyecto con implementaciones anteriores.

4.1. Metodología

Los experimentos se han realizado en dos sistemas distintos. El primer sistema cuenta con un procesador Intel Xeon Gold 5120 a 2.2 GHz con 14 núcleos y 96 GiB de memoria. Este sistema soporta la extensión vectorial AVX512 de Intel, cuya longitud vectorial es de 512 bits, equivalente a ocho datos de 64 bits. El segundo sistema es un computador de escritorio con un procesador Intel Core i7-3770K a 3.7 GHz con 4 núcleos y 32 GiB de memoria. Este sistema soporta la extensión vectorial AVX, cuya longitud vectorial es de 256 bits, equivalente a cuatro datos de 64 bits. Excepto los experimentos indicados, todos se han ejecutado sobre el sistema con el Intel Xeon Gold.

En ambos sistemas se han usado los compiladores GCC-9.2.1 [12] e ICC-19.1.1 [13] para las distintas pruebas.

Para verificar el correcto funcionamiento de la implementación de ILVES se comparan los resultados obtenidos —posiciones finales de los átomos y error relativo de cada ligadura—, con los obtenidos al ejecutar una versión secuencial de ILVES implementada en un proyecto anterior [7].

Para estudiar el tiempo de ejecución se utiliza la biblioteca *time.h* de *C*. Para evaluar la precisión se utiliza el error máximo junto con una tolerancia indicada en la configuración de cada experimento.

Para todas las pruebas se han usado cadenas reales con ramificaciones idénticas de lisina, ya que es el tipo de molécula de los implementados con más interés. Se han utilizado distintos tamaños de molécula, que van desde los 500 átomos hasta los 100000. Se considera que la molécula tipo tiene 10000 átomos y que la tolerancia tipo es 10^{-12} para datos de doble precisión (8 bytes). Todos los experimentos se han realizado usando números reales de doble precisión.

Cada experimento se ha repetido 25 veces para obtener una media de los resultados. En el caso de las medidas de tiempo de ejecución, el coeficiente de variación entre repeticiones es menor a 5 % en todos los experimentos.

Los átomos de las moléculas simuladas comienzan con posiciones que cumplen todas las restricciones de ligadura. Todos los átomos se mueven de forma aleatoria hasta un 10 % de la longitud de las ligaduras para simular las fuerzas externas antes de ejecutar los algoritmos de imposición de ligaduras.

4.2. Evaluación de las implementaciones paralelas

Los siguientes experimentos muestran el impacto del compilador así como de la paralelización de los distintos pasos de ILVES. En primer lugar se compara el rendimiento conseguido con los compiladores GCC e ICC usando el modo de ejecución más rápido de ILVES sobre el sistema con el Intel Xeon Gold (14 cores y 256 bits de longitud vectorial). A lo largo de esta sección se podrá comprobar que, efectivamente, esta configuración consigue las mejores prestaciones sobre el entorno experimental que se esta utilizando. En la Figura 4.1 se puede ver el tiempo de ejecución de ILVES para distintos tamaños de molécula usando la tolerancia tipo. En general ICC es el compilador con mejor rendimiento, por lo que será el compilador que se use en los próximos experimentos.

La Figura 4.2 muestra el *speedup* de vectorizar los distintos pasos de ILVES frente a no vectorizarlos para distintos tamaños de molécula usando la tolerancia tipo. Como se ha comentado en la Sección 3.6 la mayoría del tiempo se invierte en construir la matriz y en actualizar las posiciones de los átomos. Dichos pasos no son vectorizables, por lo que solamente se consigue un *speedup* de, como mucho, 1.10 frente a no vectorizar. La Figura 4.3 repite el mismo análisis pero sin contar el tiempo que se invierte en los pasos no vectorizados. En dicho experimento se consigue un *speedup* de hasta 2.38 frente a no vectorizar. Aumentar la longitud vectorial implica un aumento de las filas Schur

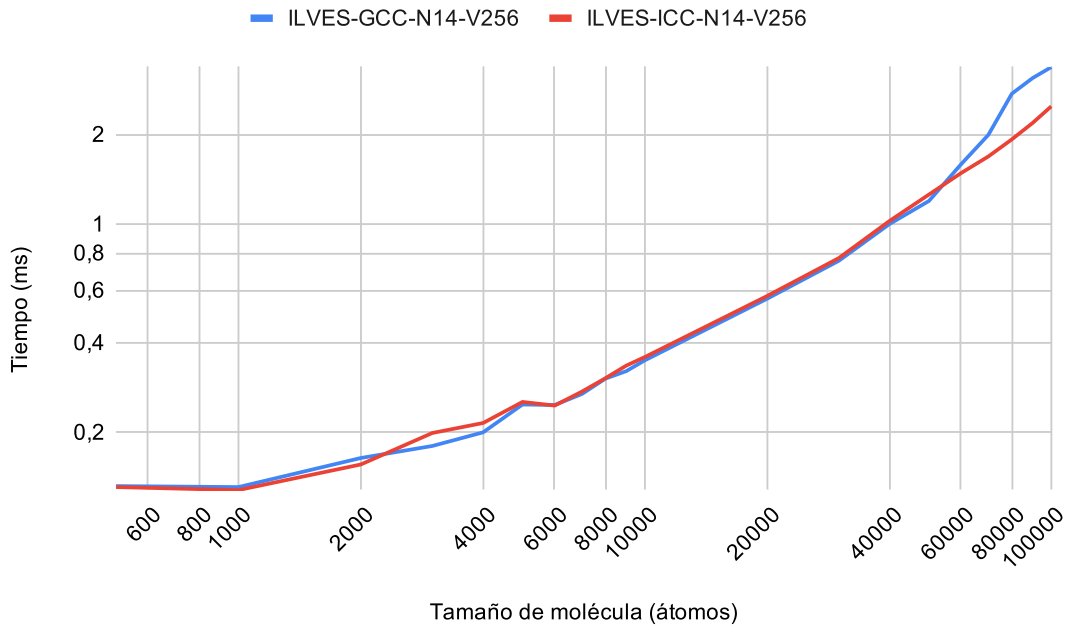

Figura 4.1: Tiempo de ejecución de ILVES (14 cores y 256 bits de longitud vectorial) usando los compiladores GCC e ICC en función del tamaño de molécula, utilizando la tolerancia tipo. Ambos ejes son logarítmicos.

privadas, esto resulta en un compromiso entre más paralelismo y una matriz Schur más grande, lo que puede provocar una perdida de rendimiento —en el experimento se consigue el mismo rendimiento con longitudes vectoriales de 256 y 512 bits—.

La Figura 4.4 muestra el impacto en tiempo de ejecución de usar paralelismo multinúcleo en los distintos pasos de ILVES para distintos tamaños de molécula usando la tolerancia tipo. Este experimento muestra una grandísima mejora cuando se usan todos los núcleos disponibles del procesador, consiguiendo un *speedup* de hasta 8.17 para moléculas grandes.

En las Figuras 4.5 y 4.6 se muestra el *speedup* de tres modos de ejecución de ILVES (vectorial, multinúcleo y vectorial + multinúcleo) frente a no aplicar paralelismo, para ambos sistemas y distintos tamaños de molécula, usando la tolerancia tipo. La mejora de usar paralelismo multinúcleo es muy notable en ambos sistemas, mientras que aplicar vectorización prácticamente no mejora los resultados. El modo de ejecución vectorial + multinúcleo siempre funciona igual o mejor que el modo secuencial, y consigue un *speedup* de hasta 8.78 para moléculas grandes en el equipo con el Intel Xeon Gold.

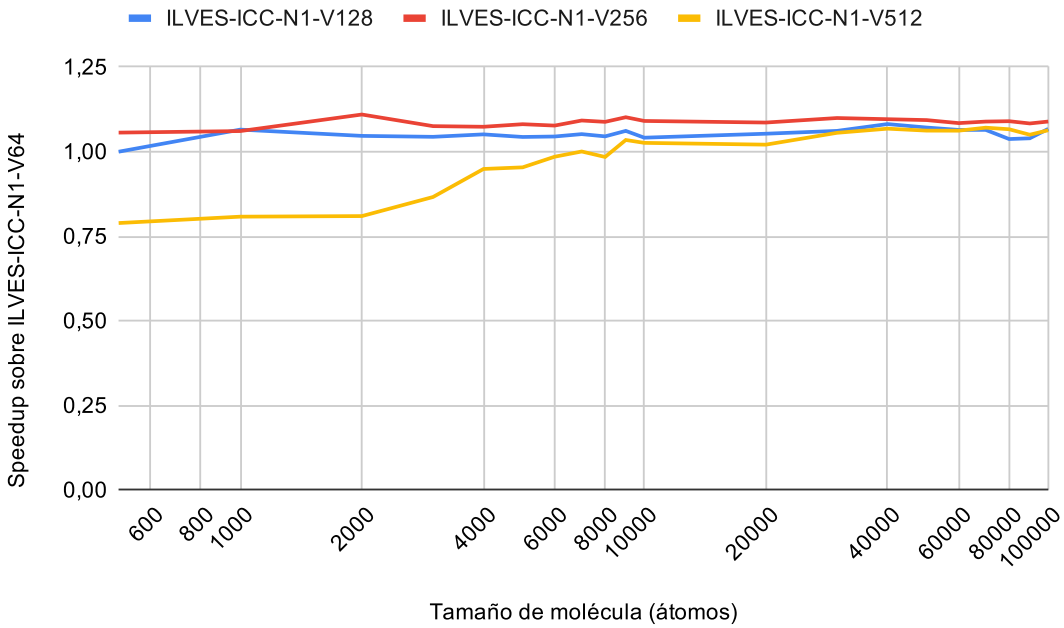

Figura 4.2: *Speedup* de ILVES ejecutado con distintas longitudes vectoriales (128, 256 y 512 bits) sobre el modo de ejecución escalar en función del tamaño de molécula. ILVES se ejecuta usando la tolerancia tipo y un solo núcleo. El eje X es logarítmico.

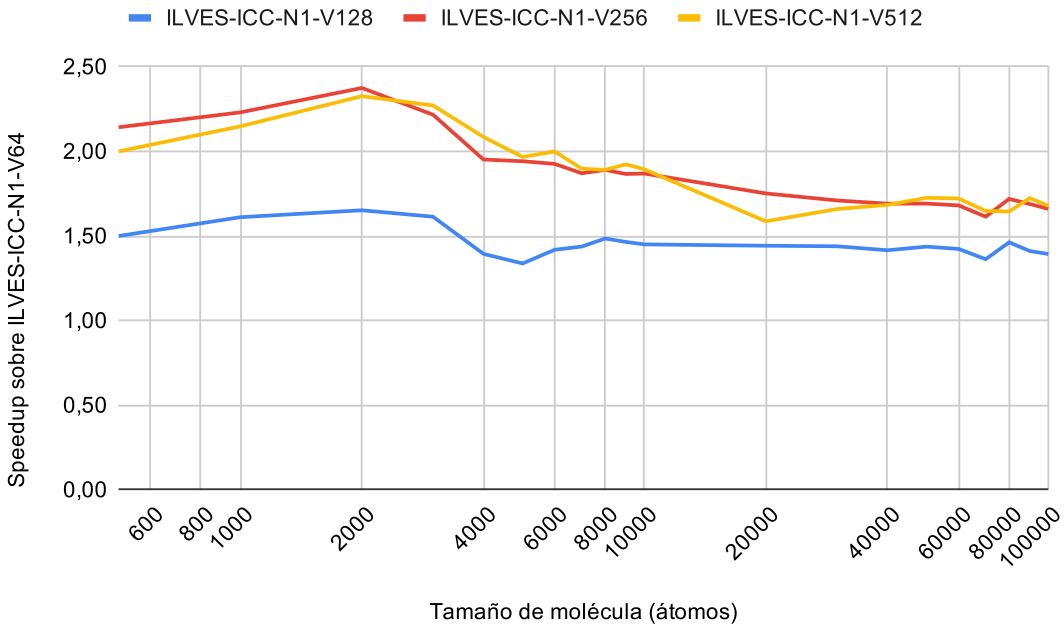

Figura 4.3: *Speedup* de ILVES ejecutado con distintas longitudes vectoriales (128, 256 y 512 bits) sobre el modo de ejecución escalar en función del tamaño de molécula. ILVES se ejecuta usando la tolerancia tipo y un solo núcleo. No se cuenta el tiempo de ejecución destinado a construir la matriz y a actualizar las posiciones de los átomos. El eje X es logarítmico.

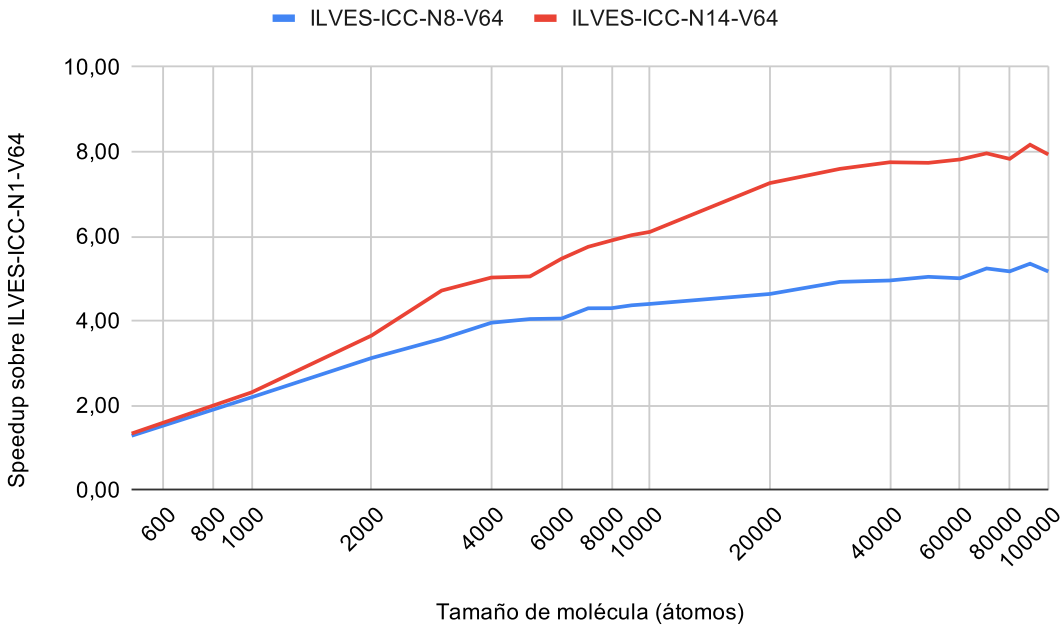

Figura 4.4: *Speedup* de ILVES ejecutado con distinto número de núcleos (8 y 14) sobre el modo de ejecución secuencial en función del tamaño de molécula. ILVES se ejecuta usando la tolerancia tipo y sin aplicar vectorización. El eje X es logarítmico.

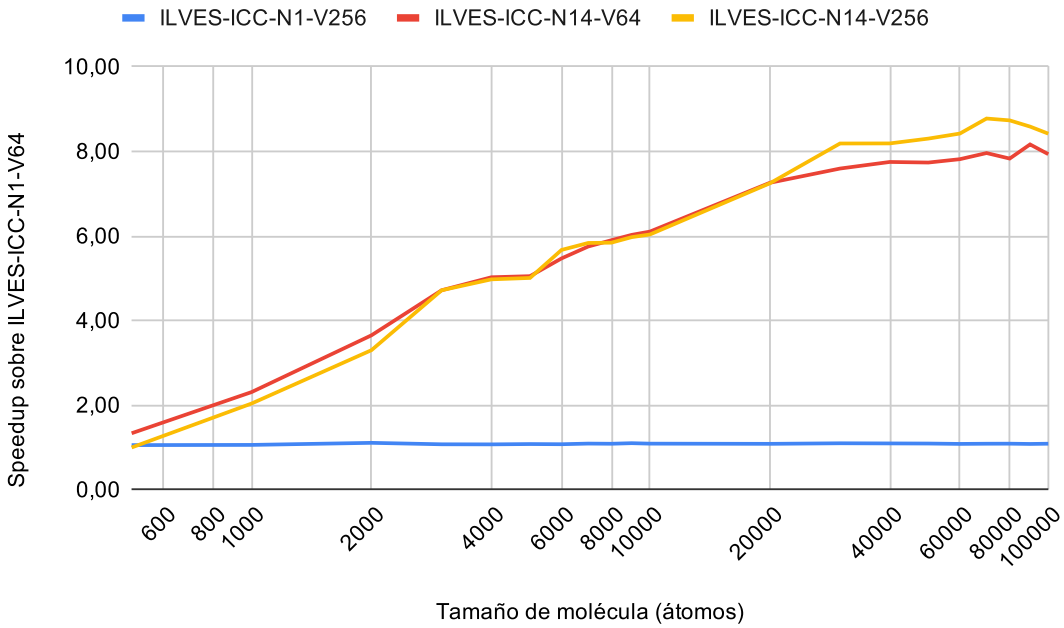

Figura 4.5: *Speedup* de ILVES ejecutado de distintas formas (vectorizando, usando paralelismo multinúcleo y vectorizando + usando paralelismo multinúcleo) sobre el modo de ejecución secuencial en función del tamaño de molécula. El experimento se ha ejecutado sobre el sistema con el Intel Xeon Gold usando la tolerancia tipo. El eje X es logarítmico.

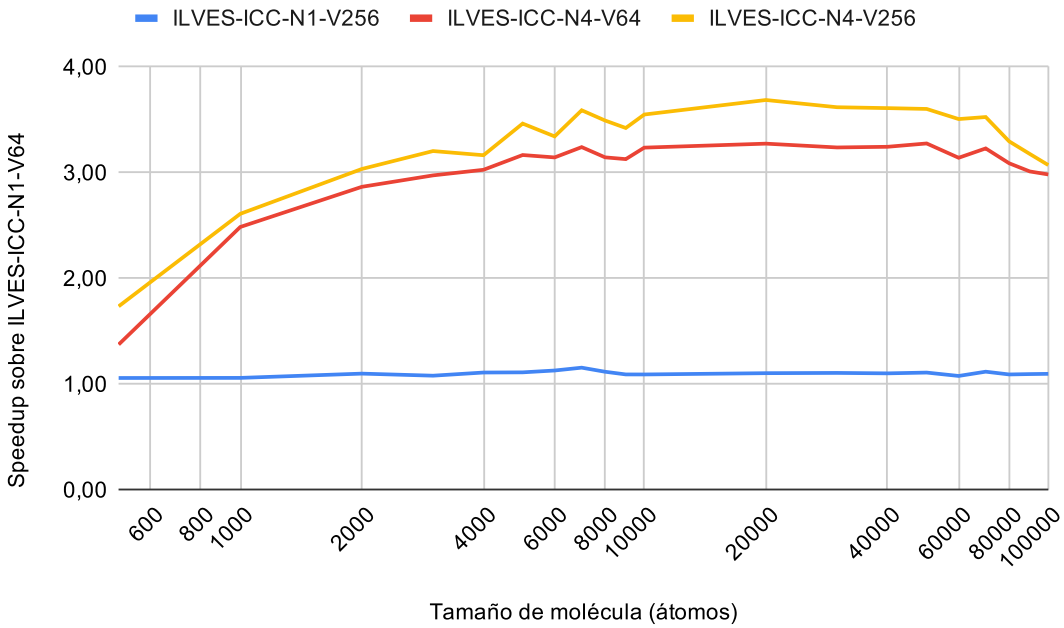

Figura 4.6: *Speedup* de ILVES ejecutado de distintas formas (vectorizando, usando paralelismo multinúcleo y vectorizando + usando paralelismo multinúcleo) sobre el modo de ejecución secuencial en función del tamaño de molécula. El experimento se ha ejecutado sobre el sistema con el i7-3770K usando la tolerancia tipo. El eje X es logarítmico.

4.3. Comparación con SHAKE

Se han realizado varios experimentos para comparar el rendimiento de ILVES con el de SHAKE. En el primero (Figura 4.7) se compara el error máximo en función del número de iteraciones para los algoritmos SHAKE e ILVES para la molécula tipo. ILVES converge cuadráticamente, por lo que necesita unas pocas iteraciones para alcanzar la precisión de la máquina, mientras que SHAKE converge linealmente, tardando más de 150 iteraciones en alcanzar la misma precisión.

En los siguientes experimentos (Figuras 4.8 y 4.9) se ha medido el tiempo de ejecución de SHAKE frente al modo de ejecución mas rápido de ILVES (14 cores y 256 bits de longitud vectorial). Se compara el tiempo de ejecución de ambos algoritmos para distintos valores de tolerancia usando la molécula tipo y para distintos tamaños de molécula usando la tolerancia tipo. En ambos experimentos el tiempo de ejecución de SHAKE es mucho mayor que el de ILVES. La diferencia de rendimiento aumenta conforme se reduce la tolerancia o crece el tamaño de la molécula. Para la tolerancia y el tamaño de molécula tipo, ILVES es 48 veces más rápido que SHAKE (usando ICC).

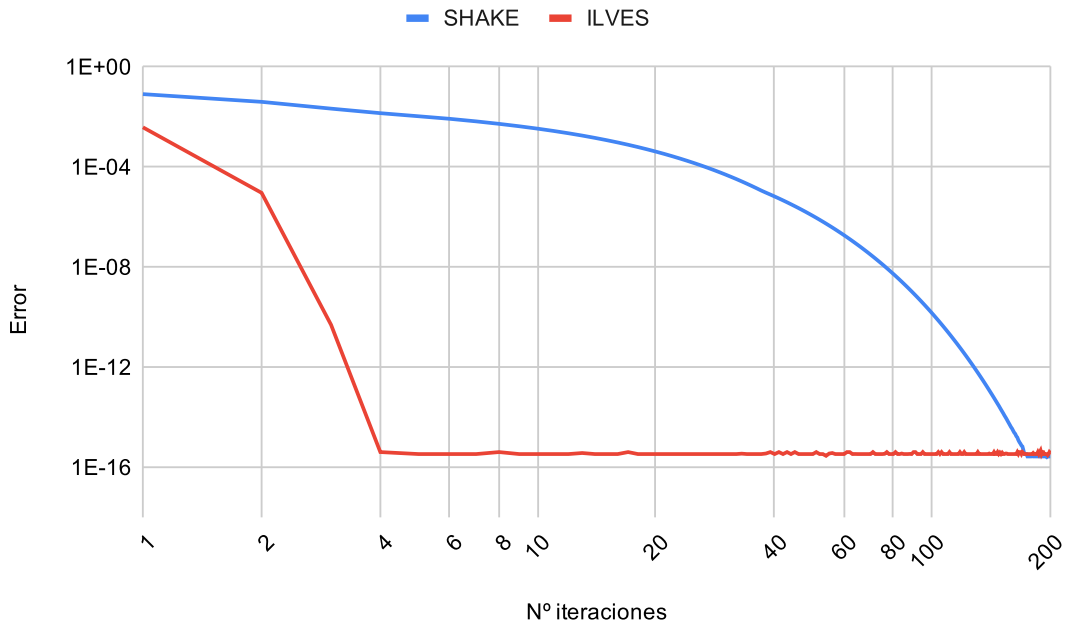

Figura 4.7: Error en función del número de iteraciones para la molécula tipo de ILVES y SHAKE. Ambos ejes son logarítmicos.

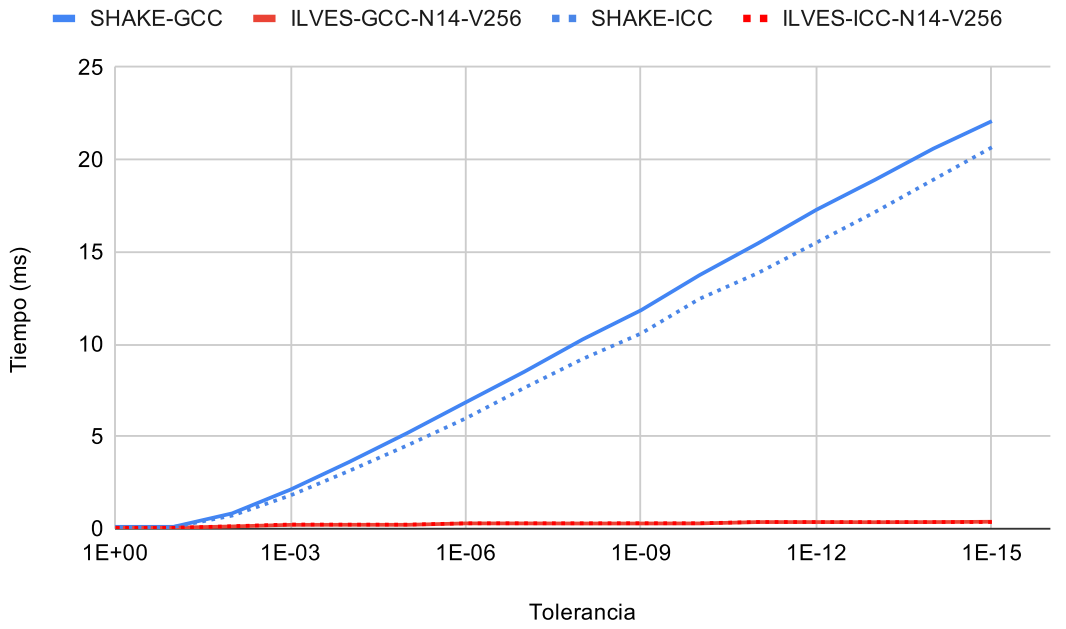

Figura 4.8: Tiempo de ejecución en función de la tolerancia de ILVES y SHAKE, usando los compiladores GCC e ICC para la molécula tipo. ILVES se ejecuta con 14 cores y 256 bits de longitud vectorial.

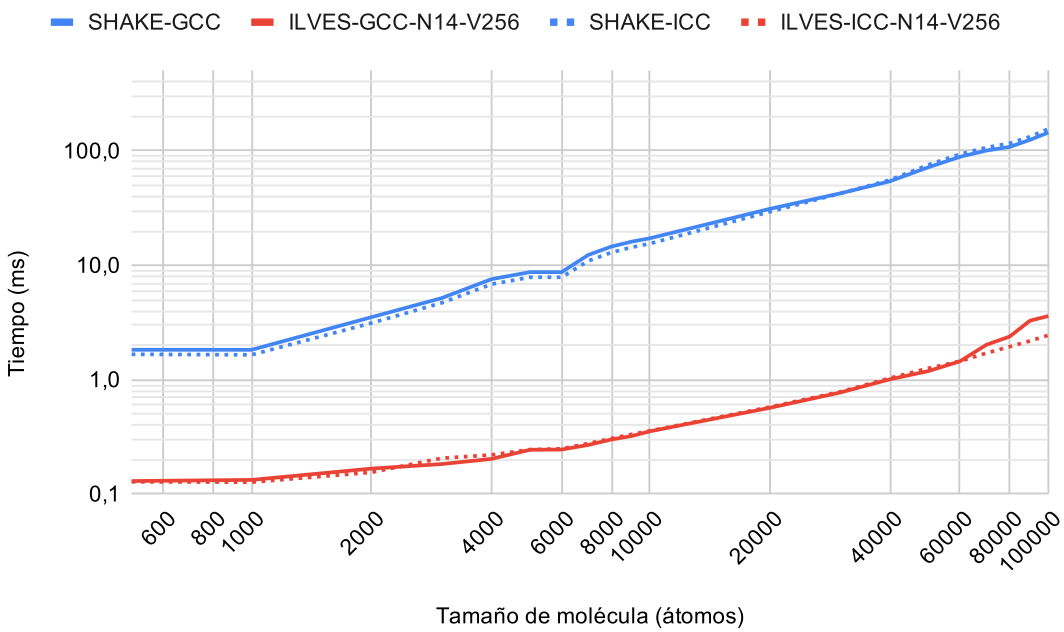

Figura 4.9: Tiempo de ejecución en función del tamaño de molécula de ILVES y SHAKE, usando los compiladores GCC e ICC para la tolerancia tipo. ILVES se ejecuta con 14 cores y 256 bits de longitud vectorial. Ambos ejes son logarítmicos.

4.4. Comparación con versiones previas de ILVES

Finalmente se ha comparado el rendimiento conseguido en la implementación realizada durante este proyecto con el obtenido en proyectos anteriores. En las Figuras 4.10 y 4.11 se compara una implementación secuencial de ILVES capaz de resolver sistemas de ecuaciones de cualquier tipo de molécula [7] (ILVES-SERIE), la implementación realizada por Rubén Langarita para moléculas lineales [4] (ILVES-LINEAL) y la implementación del presente trabajo (ILVES). Para ejecutar ILVES-LINEAL se utilizan moléculas lineales con el mismo número de átomos que las moléculas utilizadas para ejecutar ILVES e ILVES-SERIE.

Estos resultados son muy alentadores, ya que, además de aumentar hasta en 10 veces el rendimiento conseguido por la versión secuencial, se ha conseguido mantener e incluso mejorar el rendimiento de la versión ILVES-LINEAL, que también usa paralelismo vectorial y multinúcleo y resuelve sistemas de ecuaciones mucho más simples que los tratados en este proyecto (matriz de coeficientes tri-diagonal).

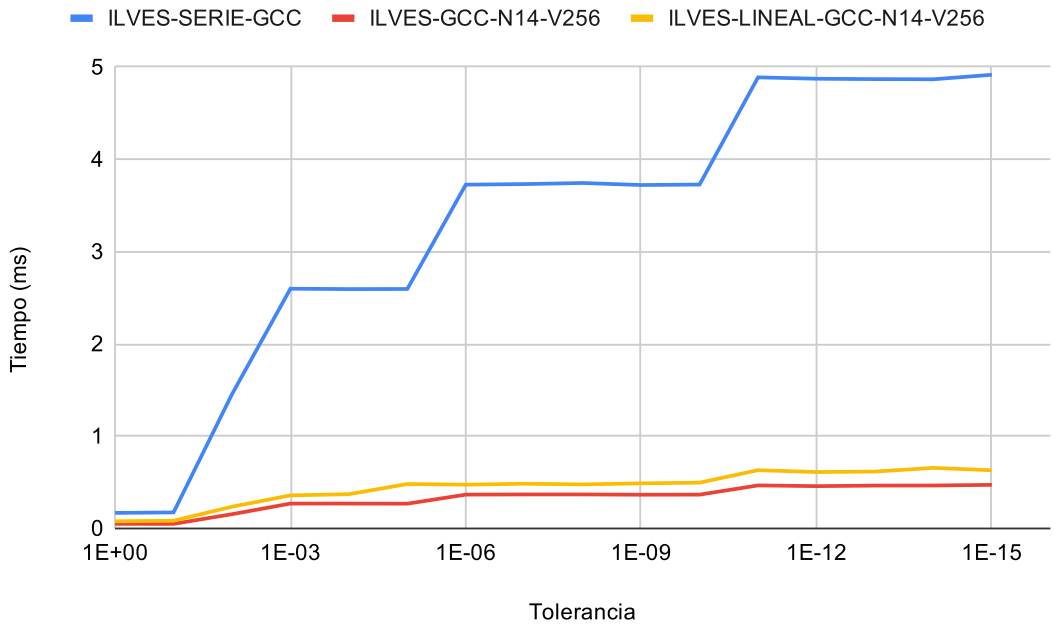

Figura 4.10: Tiempo de ejecución en función de la tolerancia de la molécula tipo para distintas implementaciones de ILVES. ILVES-LINEAL e ILVES se ejecutan con 14 cores y 256 bits de longitud vectorial.

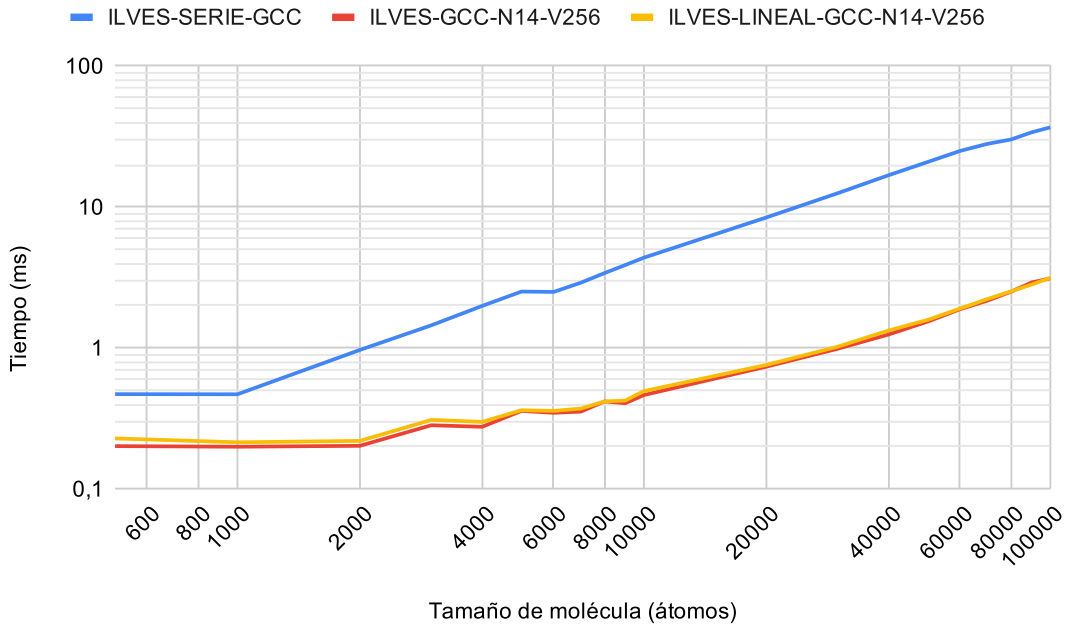

Figura 4.11: Tiempo de ejecución en función del tamaño de molécula, usando la tolerancia tipo para distintas implementaciones de ILVES. ILVES-LINEAL e ILVES se ejecutan con 14 cores y 256 bits de longitud vectorial. Ambos ejes son logarítmicos.

Capítulo 5

Conclusiones

En este proyecto se ha paralelizado el algoritmo de imposición de ligaduras ILVES, usando tanto paralelismo vectorial como multinúcleo. Aunque el objetivo del trabajo era hacer una implementación paralela de ILVES para cadenas lineales con ramificaciones lineales idénticas, se ha conseguido extender la implementación, primero para cadenas reales con ramificaciones lineales idénticas y por último para cadenas reales con ramificaciones reales idénticas, específicamente ramificaciones de lisina.

Primero se ha realizado un trabajo de investigación sobre el estado del arte de la dinámica molecular y la imposición de ligaduras. También se han estudiado a fondo proyectos anteriores realizados sobre ILVES, tanto sus documentos como su código.

Antes de comenzar la implementación se ha estudiado y desarrollado el algoritmo necesario para ejecutar ILVES de forma paralela para los tipos de molécula que se tratan en este proyecto. Para paralelizar ILVES ha sido necesario usar el método del complemento de Schur. Además, para que el compilador sea capaz de vectorizar los distintos pasos del algoritmo, se ha tenido que aplicar una reorganización de datos a la hora de construir el sistema de ecuaciones resultante de ILVES. Se ha tenido especial cuidado en que la implementación realizada en este proyecto sea lo más reutilizable posible, de manera que en futuros trabajos se tenga que reescribir la mínima cantidad de código posible. También se ha realizado un análisis de rendimiento de la implementación, identificando los cuellos de botella de la misma.

Tras realizar la implementación paralela de ILVES, se han evaluado sus prestaciones. Como ya se sabía, ILVES mejora mucho la convergencia de SHAKE, además la implementación paralela de ILVES mejora en hasta 48 veces el rendimiento de SHAKE en el sistema sobre el que se han ejecutado los experimentos. El impacto de vectorizar es pequeño, debido a que la mayor parte del tiempo de ejecución se dedica a ejecutar pasos

de ILVES que no se pueden vectorizar. Por otro lado, el impacto de aplicar paralelismo multinúcleo es muy notorio. La implementación paralela de ILVES es siempre mejor o igual que la versión secuencial, consiguiendo un *speedup* de hasta 8.78 para moléculas grandes. También se ha comparado el rendimiento conseguido en la implementación del presente trabajo con implementaciones anteriores de ILVES. Se ha mejorado el rendimiento conseguido anteriormente con una implementación paralela de ILVES para cadenas lineales sin ramificaciones, pese a que en este trabajo se tratan moléculas que resultan en un sistema de ecuaciones mucho mas complejo.

La próxima fase del proyecto será generalizar la implementación para cadenas reales con ramificaciones de cualquier tipo. El proyecto culminará con la integración de ILVES en GROMACS.

Se ha profundizado en temas de paralelismo, tanto vectorial como multinúcleo, matrices dispersas, desarrollo de código para aplicaciones científicas y computación de altas prestaciones (*high performance computing*, HPC). Además se ha hecho una introducción a nuevos campos de la física y las matemáticas, como son la dinámica molecular y la resolución de grandes sistemas de ecuaciones.

Los resultados y alcance del trabajo han sido muy positivos, por lo que se espera poder escribir una publicación en una revista internacional además de poder integrar muy pronto ILVES en GROMACS.

Bibliografía

[1] Geetika Gupta. *GPU-Accelerated Molecular Dynamics Applications Help Fight COVID-19*. 2020. URL: https://news.developer.nvidia.com/molecular-dynamics-gpu-applications-help-fight-covid-19-via-ngc (visitado 11-06-2020).

[2] Jean-Paul Ryckaert, Giovanni Ciccotti y Herman Berendsen. «Numerical-Integration of Cartesian Equations of Motion of a System with Constraints – Molecular-Dynamics of N-Alkanes». En: *Journal of Computational Physics* 23 (mar. de 1977), págs. 327-341.

[3] Berk Hess, Henk Bekker, Herman J. C. Berendsen y Johannes G. E. M. Fraaije. «LINCS: A linear constraint solver for molecular simulations». En: *Journal of Computational Chemistry* 18.12 (1997), págs. 1463-1472.

[4] Rubén Langarita Benítez. *Método paralelo de resolución de ecuaciones de ligadura para moléculas lineales*. TFG Ingeniería Informática, EINA. Directores: Jesús Alastruey Benedé y Pablo Ibáñez Marín. Universidad de Zaragoza, 2018.

[5] *GROMACS*. URL: http://www.gromacs.org (visitado 01-06-2020).

[6] María Astón Serrano Gracia. *Implementación e integración en GROMACS de un algoritmo eficiente y preciso para imponer ligaduras en simulaciones de dinámica molecular*. PFC Ingeniería Informática, EINA. Directores: Jesús Alastruey Benedé y Pablo García Risueño. Universidad de Zaragoza, 2013.

[7] Carl Christian Kjelgaard Mikkelsen, Jesús Alastruey-Benedé, Pablo Ibáñez-Marín y Pablo García Risueño. «Accelerating Sparse Arithmetic in the Context of Newton's Method for Small Molecules with Bond Constraints». En: *Parallel Processing and Applied Mathematics*. Cham: Springer International Publishing, 2016, págs. 160-171.

[8] Pablo García-Risueño, Pablo Echenique y J. L. Alonso. «Exact and efficient calculation of lagrange multipliers in biological polymers with constrained bond lengths and bond angles: Proteins and nucleic acids as example cases». En: *Journal of Computational Chemistry* 32.14 (2011), págs. 3039-3046.

[9] *OpenMP*. URL: https://www.openmp.org (visitado 29-05-2020).

[10] *Valgrind*. URL: https://www.valgrind.org (visitado 02-06-2020).

[11] *Perf*. URL: https://perf.wiki.kernel.org (visitado 02-06-2020).

[12] *GCC*. URL: https://gcc.gnu.org (visitado 17-06-2020).

[13] *ICC*. URL: https://software.intel.com/content/www/us/en/develop/tools/compilers/c-compilers.html (visitado 17-06-2020).

[14] Lorién López Villellas. *ILVES*. 2020. URL: https://github.com/LorienLV/Ilves (visitado 02-06-2020).

Lista de Figuras

Lista de Tablas

Anexos

Anexo A

Vectorización

La implementación de ILVES presentada en este proyecto hace uso de paralelismo vectorial. El paralelismo vectorial permite reducir el número de instrucciones ejecutadas en un bucle sin dependencias entre iteraciones, por ejemplo el bucle de la Figura A.1.

```
for (int i = 0; i < 400; ++i) {
    a[i] = b[i] + c[i];
}
```

Figura A.1: Bucle vectorizable.

En la versión escalar, para cada iteración del bucle hacen falta al menos cuatro instrucciones: cargar los operandos fuente en registros, operar con ellos y guardar el resultado de nuevo en memoria. Usando extensiones vectoriales se puede operar con registros vectoriales que almacenan más de un dato, lo que reduce el número de instrucciones necesarias. Se define como longitud vectorial al número de elementos que se pueden almacenar en un registro vectorial. Para el caso de una longitud vectorial de cuatro elementos, el número de iteraciones del bucle de la Figura A.1 se reduciría a 100, ya que una sola instrucción vectorial opera con cuatro elementos de cada vector.

Como se ha nombrado anteriormente, vectorizar solo es posible si no existen dependencias entre iteraciones del bucle. Por ejemplo, en el caso del bucle de la Figura A.2, vectorizar sería incorrecto, ya que, como los datos se cargan de cuatro en cuatro, al ejecutar la suma se estaría cargando el dato de la iteración $i + 1$ antes de que se guarde el resultado de la iteración i.

```
for (int i = 0; i < 399; ++i) {
    a[i + 1] = b[i] + a[i];
}
```

Figura A.2: Bucle no vectorizable.

En la Figura A.3 se muestran el análisis de dependencias de las instrucciones de los dos bucles presentados. En el bucle vectorizable todas las dependencias son a distancia cero, por lo que no existen dependencias entre iteraciones. En el bucle no vectorizable existe una dependencia a distancia uno, por lo que existen dependencias entre iteraciones.

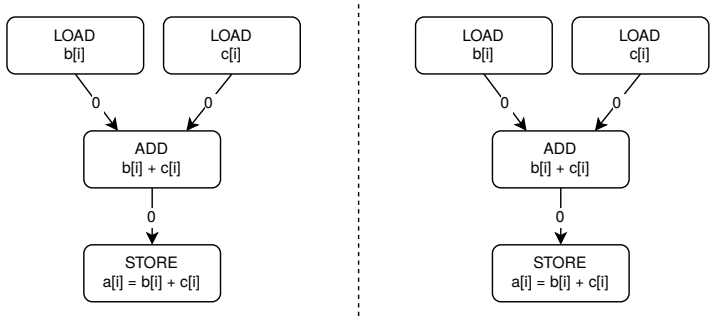

Figura A.3: Dependencias entre las instrucciones de la suma de vectores del bucle vectorizable (izquierda) y del bucle no vectorizable (derecha).

Para que un bucle sea vectorizable de forma eficiente, los datos con los que se opera deben estar contiguos en memoria. La mayoría de procesadores soportan instrucciones vectoriales que operan con datos no contiguos en memoria, pero son más lentas que si accedieran a datos contiguos. De igual manera normalmente se soporta trabajar con datos no alineados, pero es más eficiente trabajar con datos alineados a la longitud vectorial.

Anexo B

False Sharing

En sistemas multiprocesador cada procesador tiene una o varias memorias cachés locales que acceden a la memoria central del sistema. Los datos se traen de memoria central a las cachés locales en bloques de datos, normalmente de 64 bytes. El hardware se debe encargar de garantizar la coherencia entre varias cachés locales que acceden a los mismos datos de la memoria central. El fenómeno de *false sharing* ocurre cuando hilos de diferente procesador quieren modificar variables distintas que se encuentran en el mismo bloque de memoria. El mecanismo de coherencia debe invalidar dicho bloque en todos los procesadores que no hayan sido el primero en escribir, ocasionando fallos de caché, lo que inherentemente provoca una disminución del rendimiento.

Este fenómeno se puede ver ilustrado en la Figura B.1. Los hilos cero y uno quieren modificar variables distintas (i y j) que se encuentran en el mismo bloque de memoria. El bloque se carga en las cachés locales de ambos procesadores. Al modificarse al mismo tiempo dos variables del mismo bloque de memoria solo se produce una de las dos escrituras y se invalida el bloque en la caché del procesador que no ha conseguido escribir, esto obliga a dicho procesador volver a traer el bloque de memoria principal.

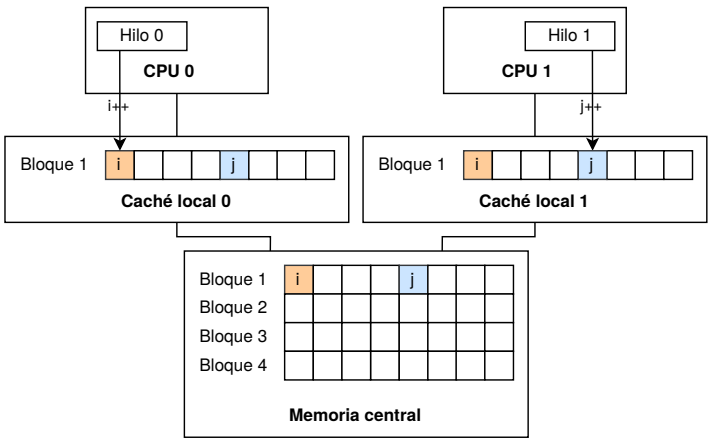

Figura B.1: Ilustración de *false sharing*. El núcleo cero y uno quieren modificar cada uno una variable que está en el mismo bloque de memoria.

El programador puede evitar este fenómeno ubicando en distintos bloques de memoria aquellas variables que son accedidas por distintos procesadores. Si no se trata, el *false sharing* puede ocurrir incluso en bibliotecas dedicadas exclusivamente al paralelismo, como *pthread.h* del lenguaje de programación *C*.

Para demostrar el impacto en rendimiento del *false sharing* se ha realizado una sencilla batería de experimentos en *C* (Figura B.2). Estos experimentos han sido lanzados en cuatro hilos, uno por núcleo, sobre un procesador Intel Core i7-3770K. El tiempo de ejecución de cada uno de ellos se ha medido mediante la biblioteca *sys/time.h* de *C*. Los tests 1 y 3 fuerzan la aparición de *false sharing*, mientras que los tests 2 y 4 lo evitan.

En la Tabla B.1 se puede ver el tiempo de ejecución de cada experimento. Claramente el *false sharing* es un fenómeno que reduce muy significativamente el rendimiento de programas que se ejecutan sobre varios procesadores.

```
#define REP 100000000
#define NUM_THREADS 4
#define ARRAY_LEN NUM_THREADS * MEM_BLOCK_SIZE

unsigned char test_array[ARRAY_LEN];
pthread_mutex_t mutex_array[ARRAY_LEN];

void test1(int thread_num) {
    for (int i = 0; i < REP; i++)
        test_array[thread_num] += 1;
}
void test2(int thread_num) {
    for (int i = 0; i < REP; i++)
        test_array[thread_num * MEM_BLOCK_SIZE] += 1;
}
void test3(int thread_num) {
    for (unsigned int i = 0; i < REP; i++) {
        pthread_mutex_lock(&mutex_array[thread_num]);
        pthread_mutex_unlock(&mutex_array[thread_num]);
    }
}
void test4(int thread_num) {
    for (unsigned int i = 0; i < REP; i++) {
        pthread_mutex_lock(&mutex_array[thread_num * MEM_BLOCK_SIZE]);
        pthread_mutex_unlock(&mutex_array[thread_num * MEM_BLOCK_SIZE]);
    }
}
```

Figura B.2: Experimentos para medir el impacto en rendimiento del *false sharing*.

Experimento	**Tiempo de ejecución (s)**
test1	1.48 s
test2	0.19 s
test3	13.34 s
test4	1.99 s

Tabla B.1: Resultados de los experimentos para medir el impacto en rendimiento del *false sharing*.

Anexo C

Código

El código del proyecto se puede encontrar en la plataforma GitHub [14]. El módulo src/ilves.{c, h} contiene la implementación paralela de ILVES desarrollada en este trabajo.