FAMAF Facultad de Matemática, Astronomía, Física y Computación — UNC Universidad Nacional de Córdoba

Optimización del Método de Funciones Sturmianas Generalizadas

por

Luis Ariel Biedma

Presentado ante la Facultad de Matemática, Astronomía, Física y Computación como parte de los requerimientos para la obtención del grado de Doctor en Matemática de la

Universidad Nacional de Córdoba

Diciembre 2023

Director: Dr. Flavio Dario Colavecchia
Codirector: Dr. Elvio Ángel Pilotta

Tribunal Especial
Titulares
Dr. Juan Randazzo (CNEA, UNCuyo)
Dr. Federico Manuel Pont (FAMAF)
Dr. Gabriel Eduardo Moyano (FAMAF)

Suplentes
Dr. Sebastián David Lopez (INENCO, UNS)
Dr. Juan Pablo Agnelli (FAMAF)
Dr. Alejandro Kolton (CNEA, UNCuyo)

«*This is fine.*»

Resumen

Esta tesis doctoral presenta la formulación y marco de trabajo llevados a cabo para resolver los denominados *Sistemas Lineales Latentes*: sistemas lineales cuya dimensión final no es conocida a priori, pero cuyos elementos pueden ser calculados computacionalmente. Este tipo de problemas se presenta al resolver las ecuaciones en derivadas parciales relacionadas con la Ecuación de Schrödinger, la cual describe sistemas físicos en Mecánica Cuántica, para simular sistemas atómicos de pocos cuerpos.

La formulación del algoritmo de resolución de sistemas lineales latentes se muestra como un problema de actualización de factorizaciones matriciales, el cual es resuelto mediante la aplicación de un algoritmo de factorización QR, utilizando técnicas de computación de alto desempeño y produciendo una ejecución de código rápida y eficiente.

El algoritmo es implementado utilizando el lenguaje de programación `C`, utilizando librerías de código abierto para la paralelización de código y resolución de sistemas lineales.

Finalmente se muestra, mediante estudios de escalabilidad y comparación con resultados presentes en la bibliografía, cómo la implementación permite simular procesos físicos, utilizando como ejemplos la doble fotoionización de las partículas de Helio y Agua, con eficiencias mayores a las reportadas hasta el momento con otras herramientas, dando la posibilidad de conseguir resultados más allá del actual estado del arte.

Agradecimientos

A mis directores. Flavio siempre estuvo disponible para cuando lo necesitara a pesar de que mi ritmo decayó, extrañamente estábamos en una etapa similar de *transición de carreras* durante todo este tiempo. Elvio me acompañó durante la licenciatura y ahora durante el doctorado, sentándose conmigo durante los primeros estudios y siempre ha sido un buen jefe de grupo.

Al resto de mi comisión académica (Nicolás W., con Germán y Cristina al comienzo) por darme una mano y al jurado, que me dio retroalimentación valiosa para mejorar el contenido de esta tesis.

A la Universitat Jaume I (Castellón, España) y las personas que formaron parte del grupo que me recibió durante mis visitas y fueron una tremenda ayuda para poder avanzar en los temas con los que trabajé, especialmente a Enrique, Rocío y Goran.

A mi familia, que siempre me apoyó y pudo mantenerme hasta que terminara mi título de grado, lo que me permitió enfocarme en los estudios para después poder subsistir por mi cuenta. Abuela, lo logré...

A Johanna, que es mi compañera de vida y me banca siempre. Tengo mucha suerte de haberme encontrado con ella y ojalá sigamos creciendo juntos.

A la gente de Invera, que me recibió y pude ser parte de una empresa de tecnología, algo que quería experimentar y fue el comienzo de lo que es la mayor parte de mi carrera en este momento. Es importante que podamos nutrirnos en conjunto la academia y la industria, para que el país pueda crecer y quizás ayudar a más gente.

Finalmente a FAMAF, la Universidad Nacional de Córdoba, la Universidad Pública en general y al CONICET por permitirme realizar este doctorado, no podría haberlo logrado sin estas instituciones, que deben perdurar en el tiempo y son parte de lo que hace grande a la Argentina. Espero poder seguir siendo parte de este sistema por siempre.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Una breve introducción a la mecánica cuántica

En palabras simples, puede decirse que la mecánica cuántica es el campo del estudio de la relación entre la luz y la materia. Es la rama de la Física que estudia la naturaleza a escalas espaciales pequeñas, los sistemas atómicos, subatómicos, sus interacciones con la radiación electromagnética y otras fuerzas, en términos de cantidades observables. Estos procesos están presentes en una variedad de aplicaciones, desde la producción de energía mediante reactores de plasma al estudio de reacciones atmosféricas.

La mecánica cuántica surge a comienzos del siglo XX. El estado de la Física hasta ese momento puede estar definido por dos importantes frases de científicos eminentes de la época:

- Pierre-Simon Laplace (1820): *Podemos mirar el estado presente del universo como el efecto del pasado y la causa de su futuro. Se podría concebir un intelecto que en cualquier momento dado conociera todas las fuerzas que animan la naturaleza y las posiciones de los seres que la componen; si este intelecto fuera lo suficientemente vasto como para someter los datos a análisis, podría condensar en una simple fórmula el movimiento de los grandes cuerpos del universo y del átomo más ligero; para tal intelecto nada podría ser incierto y el futuro, así como el pasado, estarían frente a sus ojos.*

- Albert Michelson, Premio Nobel de Física de 1907 (1903): *Las leyes fundamentales y hechos de las ciencias físicas han sido completamente descubiertos[...]. Nuestros próximos descubrimientos estarán basados en la búsqueda del sexto decimal.*

En resumen, algunas de las mentes más capaces de la época pensaban que era muy poco el avance que podía generarse en cuanto a nuevas teorías de la Física y todo estaba descubierto, el problema era que no podíamos tener en cuenta todas las variables del universo o que los experimentos no eran lo suficientemente sofisticados, ya sea por falta de capacidad de medición o la posibilidad de recrear las condiciones de sus hipótesis. Durante el primer tercio del siglo XX, esto fue cambiando, en parte gracias a tres experimentos clave: radiación de cuerpo negro, efecto fotoeléctrico y espectroscopía.

1.1.1. Radiación de Cuerpo Negro

La materia emite radiación electromagnética cuando tiene una temperatura mayor al cero absoluto en la mayoría de los casos. Un cuerpo negro es un cuerpo ideal que absorbe toda la radiación electromagnética que recibe. Algunos objetos ordinarios

similares a un cuerpo negro pueden ser un cuarto perfectamente aislado que tiene un agujero muy pequeño en su pared.

La radiación de un cuerpo negro tiene un espectro de frecuencia continuo, que sólo depende de su temperatura, llamado el espectro o ley de Planck. A medida que la temperatura disminuye, el pico de la curva de radiación del cuerpo negro se mueve a intensidades más bajas y longitudes de onda más largas. Previo al año 1900, existían dos modelos que intentaban explicar la radiación de cuerpo negro:

- La ley de Rayleigh y Jeans denotaba que la intensidad de radiación debía crecer indefinidamente a medida que se reducía la frecuencia de onda. Esta aproximación coincidía con el espectro obtenido a altas frecuencias de onda, pero era completamente equivocada en bajas frecuencias, lo que se conoce como la Catástrofe Ultravioleta.

- Además, a partir de las leyes de la termodinámica, Wilhelm Wien derivó una ley para describir el espectro de temperatura de un cuerpo negro [1]. Esta ley se basaba en una curva exponencial, cuyo valor tendía a infinito para altas frecuencias de onda, lo cual tampoco coincidía con los resultados experimentales.

La Física de esa época no podía explicar por qué el espectro observado de la radiación de un cuerpo negro, hasta que en 1900 el físico alemán Max Planck derivó de manera heurística una fórmula, asumiendo que, hipotéticamente, un oscilador cargado eléctricamente en una cavidad que contenía radiación de cuerpo negro sólo podía cargar su energía en un incremento minimal (E), proporcional a la frecuencia de su onda electromagnética asociada. Su descubrimiento fue pionero en la Física Moderna y sentó las bases de la Teoría Cuántica.

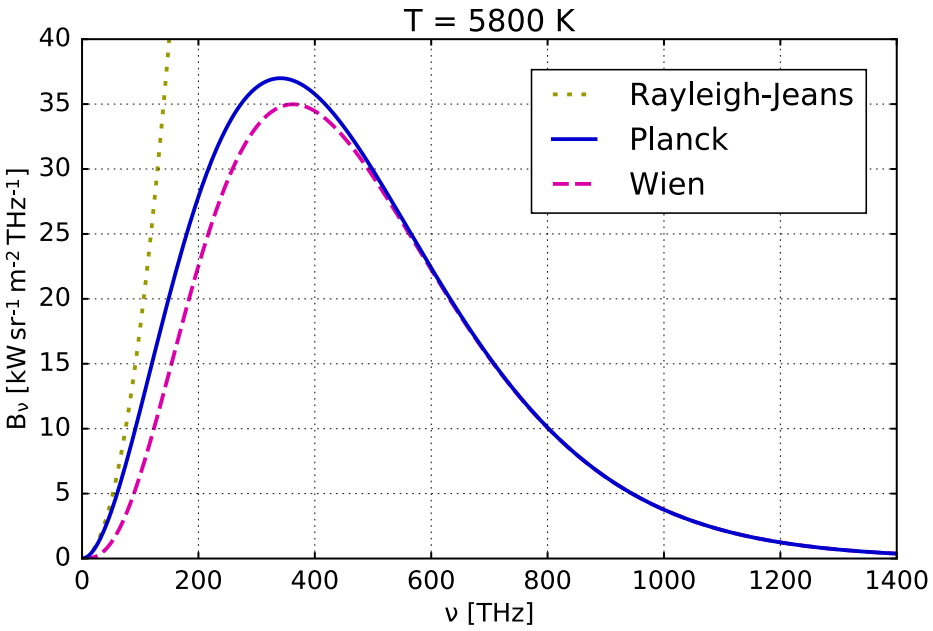

FIGURA 1.1: Comparación de la ley de Rayleigh–Jeans, aproximación de Wien y la ley de Planck, para un cuerpo a 5800°K.

1.1.2. Efecto Fotoeléctrico

Cuando un material, típicamente metálico, es irradiado (impactado por una onda), el mismo puede liberar electrones, denominados *fotoelectrones*. En el pasado, la disciplina del electromagnetismo clásico predecía que las ondas de luz continuas transferían energía a los electrones, los cuales luego serían emitidos cuando acumularan energía suficiente, esto implicaba que:

- Si se incrementa la intensidad de la luz con la que se golpea al material, la magnitud del campo eléctrico será mayor, lo que resulta en una mayor cantidad de energía.
- Si se incrementa la frecuencia de onda de la luz, la magnitud del campo eléctrico se mantendrá, por lo que la energía permanecería constante.

Los resultados experimentales no mostraban lo mismo: los electrones son expulsados sólo cuando la luz excede una cierta frecuencia, sin importar su intensidad o tiempo de exposición. Como un rayo con baja frecuencia y alta intensidad no puede alcanzar la energía requerida para producir fotoelectrones (lo que hubiera ocurrido si la energía de la luz se acumulara durante el tiempo de exposición a una onda continua), Albert Einstein tuvo la hipótesis de que un rayo de luz no es una onda que se propaga a través del espacio, sino un conjunto de paquetes de energía discretos, con el nombre de *fotones*. Esta es la teoría por la cual Einstein obtuvo su Premio Nobel.

1.1.3. Espectroscopía

Este campo de estudio de la Física fue crucial para el desarrollo de la mecánica cuántica. La espectroscopía se utiliza para estudiar la luz emitida o absorbida por los átomos y las moléculas.

La radiación electromagnética es una forma de energía producida por perturbaciones eléctricas y magnéticas oscilantes, o por el movimiento de partículas cargadas eléctricamente a través del vacío o la materia. Los campos eléctrico y magnético se combinan en ángulos rectos y la onda combinada se mueve de forma perpendicular a ambos [2].

La radiación electromagnética puede viajar a través del vacío, mientras que la mayoría de las ondas debe viajar a través de alguna sustancia. Por ejemplo, las ondas de sonido requieren un medio gaseoso, sólido o líquido para poder ser escuchadas.

Un espectro de emisión puede ser producido por un gas a baja presión excitado por medio de calor, o por colisiones con electrones. Alrededor del siglo XX se mostró que los espectros atómicos eran discretos, lo que implicaba que la energía en un átomo estaba cuantizada. Por ejemplo, una carga de alto voltaje a través de un gas de hidrógeno a baja presión genera una luz roja. Esta observación llevó a la creación de la teoría cuántica y a la comprensión de que los átomos y las moléculas solo pueden tener ciertos valores de energía discretos.

En resumen, los tres experimentos mencionados anteriormente desafiaron las ideas clásicas de la Física y llevaron al desarrollo de la Mecánica Cuántica. Estos experimentos ayudaron a los físicos a comprender la naturaleza dual de la luz, la naturaleza cuántica de los electrones y la cuantización de la energía en los átomos y las moléculas.

1.2. La Ecuación de Schrödinger

La radiación electrónica es expulsada en forma de fotones, los cuales pueden pensarse como paquetes de energía lumínica que viaja a la velocidad de la luz como ondas

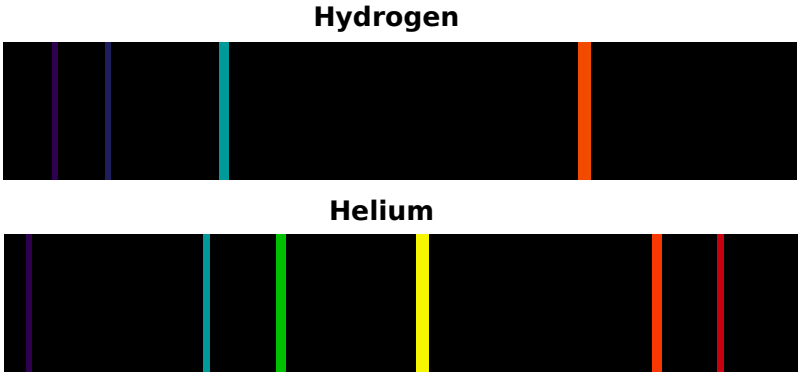

FIGURA 1.2: Espectro de emisión del hidrógeno y helio.

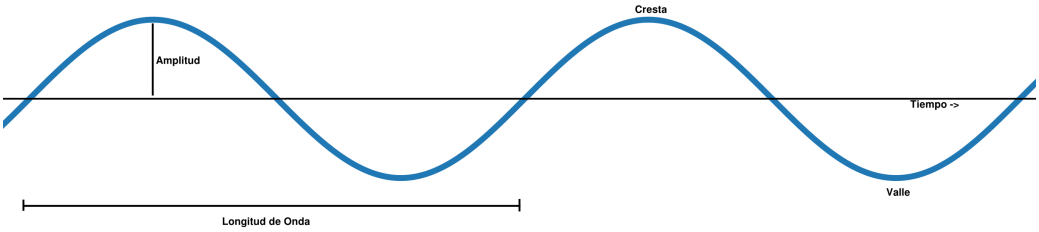

FIGURA 1.3: Elementos de una onda.

armónicas cuantizadas [3]. Las ondas descriptas en la Subsección 1.1.3 tienen ciertas características de importancia que se describen a continuación:

- Amplitud: Es la distancia desde un desplazamiento vertical máximo de la onda hasta la mitad de la misma. Mide la magnitud de la oscilación de una onda particular (su altura). Denota la intensidad o brillo de una onda en comparación con otras.
- Longitud de onda: Se representa con la letra λ es la distancia de un ciclo completo de la oscilación. Ondas con longitudes grandes (como las de radio) llevan poca energía y viceversa (como los rayos X).
- Frecuencia: Se define como el número de ciclos por segundo, y se expresa con la unidad del Hertz (Hz). Se suele representar con la letra η

La frecuencia y la longitud de onda se relacionan directamente a través de la ecuación $c = \lambda\nu$, donde c es la velocidad de la luz (las ondas electromagnéticas viajan a la velocidad de la luz en el vacío). La frecuencia es directamente proporcional a la energía de un sistema y se puede expresar como $E = \frac{h}{2\pi}\nu = \hbar\nu$, donde h es la constante de Planck ($6{,}62607 \times 10^{-34} J$) y $\hbar$ se denomina *constante de Planck reducida*. La energía puede agruparse en categorías, basada en su longitud de onda, en el espectro electromagnético.

A comienzos del siglo XX, la comunidad física comenzó a reconocer que la materia (tal como la radiación electromagnética) poseía comportamientos similares a las ondas. Aunque se sabía que la radiación obedecía las Ecuaciones de Maxwell, la materia no lo hacía.

En 1926, el físico austríaco Erwin Schrödinger formuló la que sería conocida como la Ecuación de Schrödinger (dependiente del tiempo). Se comienza con la definición del Hamiltoniano del sistema, que representa la energía total del mismo, y se expresa en términos de los operadores de posición y momento de las partículas que lo conforman.

Se utiliza la función de onda para describir el estado cuántico del sistema. Se aplica el principio de incertidumbre de Heisenberg para obtener la relación entre el operador de posición y el operador de momento. Esto implica que no se pueden medir simultáneamente la posición y el momento de una partícula con precisión arbitraria.

Teniendo en cuenta todos estos elementos es posible modelar la ecuación de Schrödinger, que describe cómo cambia la función de onda del sistema a lo largo del tiempo. La ecuación tiene la siguiente forma:

$$-\frac{\hbar}{2m}\nabla^2\Psi(\mathbf{r},t) + V(\mathbf{r})\Psi(\mathbf{r},t) = i\hbar\frac{\partial\Psi}{\partial t}(\mathbf{r},t), \tag{1.1}$$

donde Ψ representa a la función de onda, $\hbar$ a la constante de Planck reducida y V al potencial. Esta ecuación describe efectivamente a la materia como una onda que fluctúa en el espacio y en el tiempo. Como la porción imaginaria de la ecuación dicta su dependencia en el tiempo, se suele buscar la solución del problema independiente del tiempo (que también puede tener componentes complejas en el potencial V), utilizando el método de sustitución para la solución de ecuaciones diferenciales, considerando $\Psi = e^{i\frac{Et}{\hbar}}\phi$:

$$-\frac{\hbar^2}{2m}\nabla^2\phi(\mathbf{r}) + V(\mathbf{r})\phi(\mathbf{r}) = E\phi(\mathbf{r}). \tag{1.2}$$

Más allá de la utilidad de esta simplificación para describir a una onda en el espacio libre, es de mayor interés resolver el problema relacionado a un sistema donde las ondas están confinadas a una región pequeña, por ejemplo, un electrón confinado en un espacio pequeño alrededor del núcleo de un átomo.

En este entorno, se considerarán soluciones ψ a las funciones que cumplen con ciertos requisitos:

- Es cuadrado normalizable en $(-\infty, \infty)$ mediante el operador vectorial integración: $\int \|\psi\psi'\| = 1$.
- Debe ser continua como función de la posición.
- Su derivada debe ser continua como función de la posición.

El espacio formado por este tipo de funciones es conocido como $\mathcal{L}^2$. Para resolver la ecuación de Schrödinger y encontrar la función de onda del sistema, se utilizan diversas técnicas matemáticas, a continuación se describirá el proceso de obtención de la solución.

1.3. Solución de la Ecuación

Un método clave para resolver ecuaciones diferenciales que describen el mundo físico es expandir su solución en una base de funciones. Esto transforma el problema de obtener la solución de una ecuación diferencial en el de calcular los coeficientes de su expansión. Usualmente, la base está definida matemáticamente en un espacio vectorial de dimensión infinita. Así, se presenta el reto de calcular los valores relevantes para el sistema físico bajo estudio, eligiendo una base lo suficientemente grande para que el cálculo sea útil, de acuerdo a las restricciones de memoria y poder de cómputo

que se tenga a mano. Más aún, la dimensión óptima del conjunto base no suele ser conocida *a priori* y es una práctica usual realizar muchas ejecuciones del tipo *prueba y error* para encontrar un tamaño satisfactorio para la base.

Se puede establecer el problema en mayor detalle de la siguiente manera: se considera un operador diferenciable $\hat{A}(r)$, dado por las variables $r = \{r_1, r_2, \ldots, r_n\}$, en un espacio vectorial de dimensión infinita y una EDP general no-homogénea dada por:

$$\hat{A}(r)\Psi(r) = \Psi_0(r) \tag{1.3}$$

Para este problema, es de interés encontrar la solución $\Psi(r)$ para un dado conjunto de condiciones de borde. Se considera un conjunto base dado por las funciones $\{\phi_n(r)\}$. Entonces, las funciones $\Psi(r)$ y $\Psi_0(r)$ pueden presentarse en forma de expansión por su base:

$$\Psi(r) = \sum_n x_n \phi_n(r), \qquad \Psi_0(r) = \sum_n b_n \phi_n(r). \tag{1.4}$$

Esta expansión incluye los infinitos elementos de nuestra base. Para poder utilizar esta expresión en un entorno computacional y obtener una solución numérica para la Ec. (1.3), es necesario seleccionar sólo una cantidad N de elementos de nuestra base, tal que

$$\Psi(r) \approx \Psi^{(N)}(r) = \sum_{n=1}^{N} x_n \phi_n(r), \qquad \Psi_0(r) \approx \Psi_0^{(N)}(r) = \sum_{n=1}^{N} b_n \phi_n(r). \tag{1.5}$$

De esta manera, la Ec. (1.3) es proyectada en la base truncada seleccionada, obteniendo un sistema lineal para los coeficientes $x^{(N)} = \{x_1, x_2, \ldots, x_N\}$:

$$A^{(N)} x^{(N)} = b^{(N)}, \tag{1.6}$$

donde la matriz $A^{(N)}$, de tamaño $N \times N$, posee elementos $A^{(N)}_{nm} = \langle \phi_n, \hat{A}\phi_m \rangle$ y el vector $b^{(N)} = \{b_1, b_2, \ldots, b_N\}$ es la representación aproximada de la condición inicial Ψ_0 en la base truncada.

1.3.1. Doble Fotoionización

La doble fotoionización (DPI) es un fenómeno que ocurre cuando un fotón cargado de energía es absorbido por un átomo, resultando en la emisión de dos electrones. La DPI resulta en un ion doblemente cargado y dos electrones salientes, que están sujetos a fuerzas de Coulomb de largo alcance. El estudio de este proceso es de interés fundamental para la Física Atómica, ya que permite directamente atacar el problema de Coulomb de tres cuerpos. Se puede extraer información de la DPI realizando mediciones de la sección eficaz diferencial triple (TDCS), que permite la determinación completa de la energía y el patrón angular de la reacción de separación. Es posible realizar simulaciones para aproximar numéricamente estas funciones de onda y estudiar en mayor detalle este proceso.

La doble fotoionización del helio ha sido un problema de interés durante décadas. Es uno de los sistemas más simples en los que la emisión al continuo de dos electrones puede ocurrir (mediante la absorción de un único fotón). Además provee información sobre las dinámicas de Coulomb de tres cuerpos en el continuo para objetos más complicados, ionización electrón-átomo y en otros problemas de ruptura de Coulomb en general.

Mediciones de este y otros sistemas similares pueden encontrarse en amplitud en la bibliografía [4, 7, 8, 9, 10, 11, 12, 13, 14, 5, 6], las cuales fueron obtenidas gracias al desarrollo de técnicas de medición de dinámica completa como espectrómetros Time of Flight [15] o COLTRIMS [16]. Desde el punto de vista teórico, hay varios enfoques que pueden aplicarse a este problema, como cálculos con ondas parciales hiperesféricas[4], Escalado Complejo Exterior (ECS) mediante B-splines [17], cálculos de Convergent Close Coupling (CCC) [18], cálculos con R-matrices hiperesféricas [19] y expansión sturmiana con funciones complejas [20].

En años más recientes, se ha desarrollado un enfoque espectral basado en el concepto de Funciones Sturmianas Generalizadas (GSF por *Generalized Sturmian Functions*), que permite representar cualquier tipo de condiciones de borde físicas [21]. Este enfoque se basa en imponer condiciones a la base de funciones en la coordenada en la que son definidas. Estas pueden ser condiciones de cúspide de Kato [22], condiciones de bordes de caja, comportamiento de decaimiento exponencial asociado con los estados ligados [23, 24], o condiciones de flujo salientes (o entrantes), incluyendo las fases de Coulomb dependientes de coordenadas [17]. Las Funciones Sturmianas de Coulomb (CSF), introducidas por H. Shull y P. O. Löwdin [25], corresponden a un caso particular de las GSF. Estas funciones son utilizadas para problemas de scattering y reciben su nombre por su cercanía a la teoría de Sturm-Liouville.

La propuesta de las GSF ha probado ser exitosa para describir problemas de estados ligados atómicos [22] y también para estados de scattering estacionarios (independientes del tiempo) [26]. La ventaja de un enfoque espectral por sobre los métodos de grilla reside en la reducción del número de elementos necesarios para representar a la función de onda. Esto se refleja en el tamaño de la representación matricial del hamiltoniano, que se relaciona directamente a los recursos computacionales necesarios para realizar el cálculo. De cualquier manera, no siempre es fácil obtener la solución deseada, especialmente cuando las condiciones de borde están relacionadas con procesos de scattering. Se ha podido tener éxito en este tipo de simulaciones [27], mostrado en modelos de ondas s para procesos del tipo (e, 2e) [26] o (e, 3e) [28].

El objetivo de este enfoque es el de mostrar resultados para un sistema de Coulomb de desintegración de tres cuerpos, considerado en toda su dimensionalidad, prediciendo mediciones absolutas y compararlas con otros datos obtenidos en la teoría. Teniendo en cuenta esto, la doble fotoionización del helio desde su estado fundamental, mediante la absorción de un único fotón, constituye un sistema ideal por dos razones:

- sus medidas y datos teóricos (para chequeo de las soluciones) están disponibles en la literatura, y
- al existir un sólo *cuanto* de momento angular transferido al sistema, facilita el seguimiento del tamaño que puede tomar el sistema lineal asociado.

A continuación, se exponen las ecuaciones utilizadas para el estado ligado de un átomo de helio y la función de onda de scattering luego de la absorción. Además se presentarán los fórmulas empleadas para evaluar las secciones eficaces. Luego, se propone la serie de ondas parciales de las funciones de onda y los parámetros utilizados para construir el sistema lineal de ecuaciones. Para las próximas ecuaciones, se asumen las unidades atómicas ($m = \hbar = e = 1$) a menos que se describa lo contrario.

1.3.2. Enfoque perturbativo

Con el objetivo de describir las dinámicas de la doble ionización de helio por el impacto de un fotón en un esquema independiente del tiempo, se busca calcular la

función de onda de scattering Ψ_{sc}, la cual contiene la mezcla entrelazada de estados finales de la colisión. La misma puede obtenerse resolviendo la siguiente ecuación de Scrödinger no homogénea, a partir de la Ec. (1.2):

$$[H - E]\,\Psi^{+}_{sc}(\mathbf{r}_1, \mathbf{r}_2) = \hat{\varepsilon}.(\bigtriangledown_{\mathbf{r}_1} + \bigtriangledown_{\mathbf{r}_2})\Psi_0(\mathbf{r}_1, \mathbf{r}_2), \tag{1.7}$$

donde H es el hamiltoniano del sistema, E es la energía del sistema, $\hat{\varepsilon}$ es el vector de polarización del laser, y Ψ_0 es el estado ligado inicial del átomo, con energía E_0, que satisface:

$$H\Psi_0(\mathbf{r}_1, \mathbf{r}_2) = E_0\Psi_0(\mathbf{r}_1, \mathbf{r}_2), \tag{1.8}$$

mientras que la conservación de energía se rige por la ecuación:

$$E_0 + \omega = E,$$

donde ω es la energía del fotón. El lado derecho de Ec. (1.7) tiene dos operadores gradiente asociados al *gauge* de velocidad, mientras que el *gauge* de longitud muestra: $\omega\hat{\varepsilon}.(\mathbf{r}_1 + \mathbf{r}_2)\Psi_0(\mathbf{r}_1, \mathbf{r}_2)$. Los vectores $\mathbf{r}_i$ $(i = 1, 2)$ representan la ubicación de los electrones con respecto al núcleo, que se considera en reposo en el centro de masa ubicado en el origen de coordenadas. Las coordenadas esféricas asociadas a estos dos vectores serán utilizadas para facilitar la solución de las ecuaciones.

La Ec. (1.7) admite múltiples soluciones regulares, las cuales pueden construirse como combinaciones lineales entre la solución particular y la solución de la ecuación homogénea. Para describir de manera apropiada la física del problema bajo consideración, es necesario obtener la solución que se comporte como una onda saliente en ambas coordenadas de los electrones a grandes distancias. Se ha demostrado que esta solución es única, y que sus propiedades asintóticas contienen la información detallada para el proceso de dispersión [29].

El siguiente paso es evaluar los observables físicos. Se puede emplear las fórmulas usadas en [17], donde la matriz de transición es la siguiente:

$$f(\mathbf{k}_1, \mathbf{k}_2) = \left\langle \Phi^{-}(\mathbf{k}_1, \mathbf{r}_1)\Phi^{-}(\mathbf{k}_2, \mathbf{r}_2)|E - T + V|\Psi^{+}_{sc}\right\rangle. \tag{1.9}$$

A partir de ella, se pueden extraer la sección eficaz diferencial triple (TDCS)

$$\frac{d^3\sigma}{dE_1 d\Omega_1 d\Omega_2} = \frac{4\pi^2}{\omega c} k_1 k_2 \left|f(\mathbf{k}_1, \mathbf{k}_2)\right|^2, \tag{1.10}$$

y la sección eficaz diferencial simple (SDCS)

$$\frac{d\sigma}{dE_1} = \int \frac{d^3\sigma}{dE_1 d\Omega_1 d\Omega_2} d\Omega_1 d\Omega_2.. \tag{1.11}$$

En las Ecs. (1.9) a (1.11) Φ^{-} representan a las funciones coulombianas (normalizadas por una función δ en momento), $\mathbf{k}_i$ $(i = 1, 2)$ es el momento final del electrón i, T es la energía cinética del sistema y $V(r_1, r_2) = -2/r_1 - 2/r_2$.

Las funciones Φ^{-} son funciones coulombianas con condición asintótica de onda entrante:

$$\Phi^{-}(\mathbf{k}_i, \mathbf{r}_i) = \Gamma(1 - i\eta)e^{-\pi\eta/2}e^{i\mathbf{k}_i\cdot\mathbf{r}_i}\,{}_1F_1(+i\eta, 1, -i\|\mathbf{k}_i\|\|\mathbf{r}_i\| - i\mathbf{k}_i\cdot\mathbf{r}_i) \tag{1.12}$$

donde ${}_1F_1$ es la función hipergeométrica confluente, y $\eta = 2/k$, para $i = 1, 2$.

1.3.3. Expansión de Ψ_{sc}^{+}

Las soluciones de las Ecs. (1.7) y (1.8) son obtenidas a través de una expansión sturmiana de la forma:

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) = \sum_{l_1,l_2} \mathcal{A}_S R_{l_1,l_2}^{L,M}(r_1, r_2) \mathcal{Y}_{l_1,l_2}^{L,M}(\hat{\mathbf{r}}_1, \hat{\mathbf{r}}_2) \tag{1.13}$$

donde

$$R_{l_1,l_2}^{L,M}(r_1, r_2) = \sum_{n_1,n_2} a_{n_1,n_2,l_1,l_2}^{L,M} \frac{S_{n_1}(r_1)}{r_1} \frac{S_{n_2}(r_2)}{r_2} \tag{1.14}$$

y donde $a_{n_1,n_2,l_1,l_2}^{L,M}$ son los coeficientes de expansión, las funciones $\mathcal{Y}$ son los armónicos biesféricos [30, 23], $\mathcal{A}_S$ es el operador de simetría definido de la forma:

$$\mathcal{A}_S \Psi(\mathbf{r}_1, \mathbf{r}_2) = \frac{1}{\sqrt{2}} \left[\Psi(\mathbf{r}_1, \mathbf{r}_2) + (-1)^S \Psi(\mathbf{r}_2, \mathbf{r}_1) \right], \tag{1.15}$$

y las funciones radiales S_n son soluciones de la ecuación sturmiana:

$$\left[-\frac{1}{2} \frac{d^2}{dr^2} + U(r) - \hat{E} \right] S_n(r) = -\beta_n V(r) S_n(r) \tag{1.16}$$

donde β es su autovalor. A su vez las soluciones de la ecuación deben respetar la condición de cúspide

$$\lim_{r \to 0} \frac{d\Phi(r)}{dr} = -Z_{in} \Phi(r) \tag{1.17}$$

y la condición de comportamiento asintótico de la solución

$$\lim_{r \to \infty} \Phi(r) \propto e^{-\kappa r - \frac{Z_{as}}{\kappa} log(2\kappa r)}, \tag{1.18}$$

donde $\kappa = \sqrt{-2E} > 0$, Z_{as} y Z_{in} representan a la carga asintótica y la carga en la zona próxima a la nube de electrones vistas por el electrón, respectivamente. Además $\hat{E}$, U y V son los potenciales que pueden elegirse de forma apropiada para optimizar la convergencia [22]. Las soluciones de la Ec. (1.16) se evalúan numéricamente [31].

Para este caso, se utilizaron dos bases diferentes para expandir cada uno de los estados Ψ_0 y Ψ_{sc}^{+}. Para ambas funciones se fija $U(r) = -2/r$ y no se incluyen las barreras centrífugas, incluso cuando las funciones S aparecen multiplicando a las funciones biesféricas $\mathcal{Y}$ para $l_i > 0$ $(i = 1, 2)$ en la Ec. (1.13). Por lo tanto, estos términos no son removidos de las ecuaciones acopladas cuando se hace el reemplazo de las expresiones 1.13 en la Ec. (1.7), y los elementos de la matriz separable correspondiente necesitan ser evaluados. La ventaja de este procedimiento es evitar la evaluación de múltiples integrales bidimensionales asociadas con los elementos de la matriz de repulsión electrónica, que de otra manera deberían evaluarse para cada valor de momento angular l, dado que los conjuntos de funciones base son diferentes. Los recursos computacionales (de tiempo de cómputo y memoria de disco) son reducidos considerablemente si se procede de esta manera, comparados con utilizar una base para cada l. Para Ψ_0 se usa la base obtenida fijando el valor negativo $\hat{E} = -1$ en la Ec. (1.16), el cual otorga al conjunto un comportamiento de caída. En forma más general, el potencial U podría elegirse de diferentes maneras, en particular puede tomarse el utilizado en [22], viendo

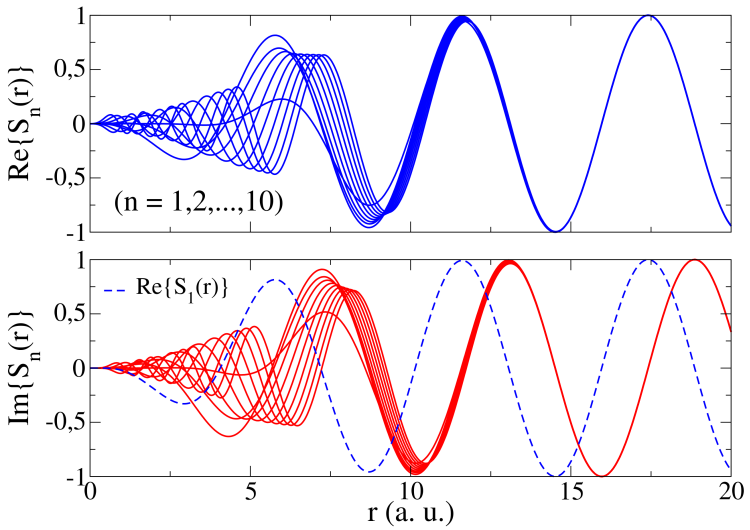

FIGURA 1.4: Funciones Sturmianas Generalizadas con condiciones de flujo saliente típicas impuestas en $r = 10$ *a. u.*.

al potencial con la forma $U(r) = U_{aux}(r) + \tilde{U}(r)$, donde

$$U_{aux}(r) = -\frac{Z_{as}}{r} + \frac{(-Z_{in} + Z_{as})e^{-\alpha r}}{r}. \tag{1.19}$$

Luego, la función correspondiente a $\tilde{U}$ es una perturbación que no modifique el comportamiento coulombiano de U_{aux} en los límites a cero e infinito, por ejemplo:

$$\tilde{U}(r) = \tilde{U}_0 e^{-\sigma(r-r_0)^2} \tag{1.20}$$

Luego, el potencial generador V puede ser el siguiente:

$$V(r) = -\frac{e^{-\lambda r}}{r}[r^{\delta} e^{-\gamma r^2} + (1 - e^{-\gamma r^2})] \tag{1.21}$$

con $\lambda > 0$ y $\delta > 0$, el cual lleva a las bases a tener las condiciones asintótica y de cúspide asociadas a la carga de U (GSF) correctas, sin agregar singularidades a las funciones solución. Se puede utilizar el conjunto de Funciones Sturmianas de Coulomb (CSF) para $l_i = 0$ fijando $V = -1/r$, que no incluyen la condición de cúspide, pero esto no resulta en diferencias en las secciones eficaces. Los resultados presentados aquí fueron evaluados con una base radial de CSF de ondas s simetrizadas apropiadamente, con 6 elementos para cada coordenada radial, fijando $L = M = 0$, $l_1 = l_2 = 0, 1, 2, 3, 4$ en la Ec. (1.13), lo que lleva a un total de 105 orbitales. Con estos valores se encontraron energías por debajo de $-2{,}903\,231$, mientras que el valor exacto es de $-2{,}903\,724$ [32]. La precisión numérica con la que se representa el estado es más que suficiente para el propósito actual.

Se emplean conjuntos base con condiciones de flujo saliente para describir la función de onda de scattering Ψ^{+}_{sc}, con una energía $\hat{E}$ igual al doble de la energía en el continuo del sistema físico [27]. Las sturmianas de flujo saliente se muestran en la Figura 1.4, donde se grafican las partes real e imaginaria de varios elementos, normalizados para que su valor en $r = 10$ *unidades atómicas* coincida con las condiciones de borde. Desde ese punto en adelante, todas las funciones de onda tienen el mismo comportamiento. Notar que la diferencia de fase entre las partes real e imaginaria es

$\pi/2$.

Normalmente se utiliza un pozo cuadrado como potencial generador para representar a las ondas continuas, pero el mismo resulta ser una buena elección para el *gauge* de longitud. De cualquier manera, debido a la estructura del lado izquierdo de la ecuación en el *gauge* de velocidad cercana a la región $r_1 = r_2 = 0$, la convergencia con esta base no es tan buena como para el *gauge* de longitud. Esta expansión se mejora agregando el potencial de Yukawa a V para concentrar las oscilaciones alrededor de esa región. Esta elección da resultados precisos en ambas estimaciones. Se utilizan los valores $L = 1$, $M = 0$, $l_1 = l$ y $l_2 = l + 1$, con $l = 0, 1, 2, 3$ en la Ec. (1.13), como se sugiere en [17]. Agregar más términos de ondas parciales no resulta en cambios en la TDCS. El dominio radial elegido es de 35 $u.\,a.$ (30 $u.\,a.$) para 20 eV (40 eV), para los cuales se utilizan 35 (50) orbitales radiales, obteniendo un total de 4900 (10000) funciones en la base.

Para encontrar los coeficientes lineales $a^{L,M}_{n_1,n_2,l}$ de la Ec. (1.13) se reemplazan en las Ecs. (1.7) y (1.8) y se proyectan hacia la izquierda por todos los elementos de la base identificados por $\{l', n_1', n_2'\}$. Este es el método clásico de Galerkin. Se obtiene un problema de autovalores generalizado para la Ec. (1.8) [22], cuyo mínimo autoestado de energía es el estado fundamental del átomo de Helio. Este sistema puede resolverse utilizando rutinas de álgebra lineal computacionales de forma sencilla. Una vez que ese estado se normaliza, se lo utiliza en la Ec. (1.7) para construir otro sistema de ecuaciones lineales de la forma $\mathbf{H}.\mathbf{a} = \mathbf{b}$ para los coeficientes $\mathbf{a}$ relacionados a Ψ^+_{sc}. Este nuevo sistema queda bien determinado y sus soluciones también pueden obtenerse utilizando paquetes de álgebra lineal numérica.

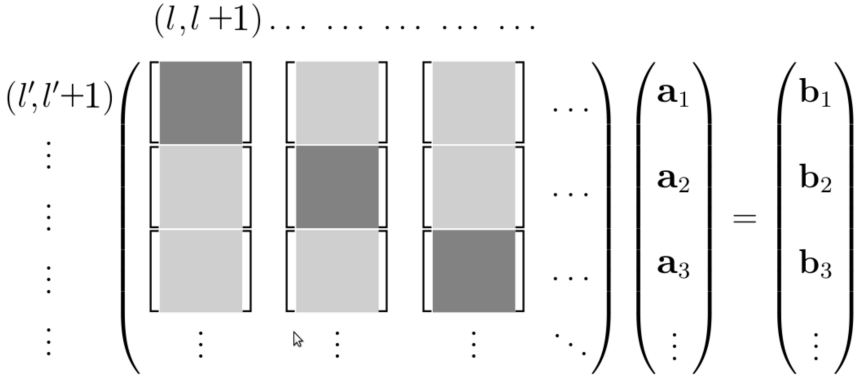

FIGURA 1.5: Estructura por bloques del conjunto de ecuaciones lineales, cuya solución da la expansión en coeficientes a_ν de Ψ^+_{sc}. l y l' toman los valores 0, 1, 2, 3.

De cualquier manera, se prefiere utilizar métodos iterativos, pues su convergencia es más rápida que la de los métodos exactos, y su estructura permite implementar sencillamente el problema en un único procesador si sólo un sub-bloque de la matriz se almacena en la memoria RAM en cada momento. En este caso se probaron distintos métodos, encontrando que el método del Gradiente Conjugado Cuadrado Precondicionado resultó ser el más estable utilizando un precondicionamiento adecuado. Como se sugiere en la literatura, una buena elección para la matriz de precondicionamiento es la solución del sistema obtenido reemplazando todos los elementos matriciales por cero fuera de los bloques diagonales, donde los bloques se definen por los elementos matriciales cuyos números cuánticos angulares $\{l', l\}$ están fijos.

Se encuentra para la estructura de esta matriz (intrínsicamente relacionada con el conjunto base) que, luego de elegir un vector aleatorio $\mathbf{a}^{(0)}$ como primera propuesta de solución, el error, definido como la norma del vector $(\mathbf{H}.\mathbf{a}^{(n)}) - \mathbf{b}$, decrece al ritmo de casi un orden de magnitud por iteración del método. Se utilizó como criterio de parada del método iterativo que el error relativo sea menor que una tolerancia igual a 10^{-10}. Soluciones típicas del sistema de ecuaciones pueden verse en la Figura 1.6, donde se muestra las partes real e imaginaria de un término directo (i.e. sin aplicar el operador de anti simetrización $\mathcal{A}_S$). En el gráfico se observa que la función obtenida muestra un correcto comportamiento asintótico, que puede interpretarse como una onda esférica con origen en el origen de coordenadas.

Una vez que la onda parcial es evaluada, se calculan las secciones eficaces diferenciales triple y simple, a través de las fórmulas de las Ecs. (1.9) a (1.11). Un análisis más detallado de las fórmulas y su expresión para cada onda parcial pueden encontrarse en la referencia [17], donde se muestra cómo las integrales de volumen pueden transformarse en superficies integrales en un valor fijo del hiper radio.

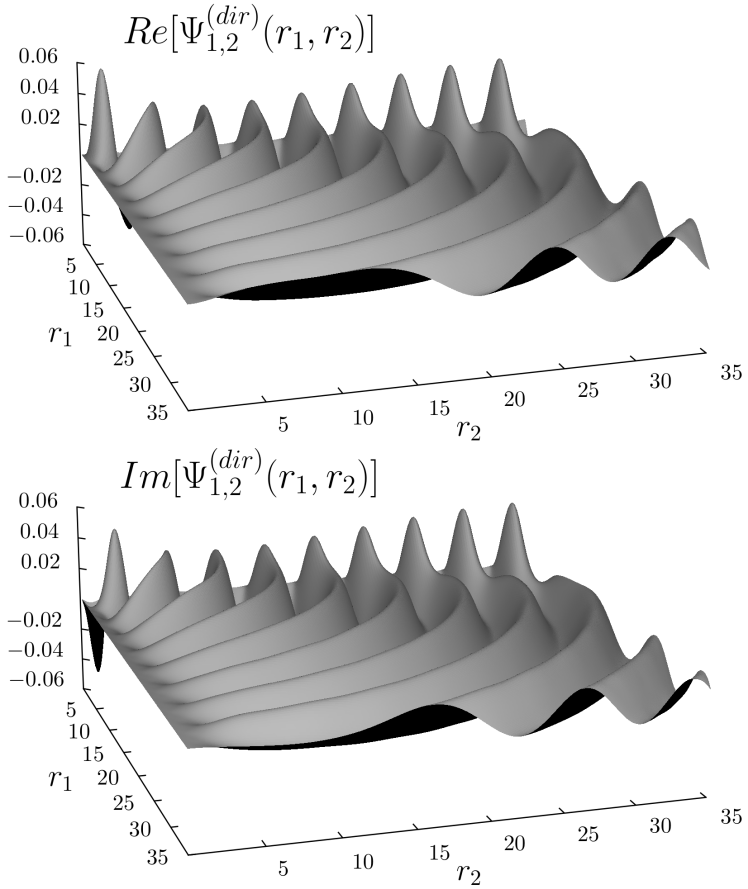

FIGURA 1.6: Partes real e imaginaria de la función de onda directa asociada al término de la segunda onda parcial de Ψ_{sc}^{+} para 20 eV.

1.4. Desventajas del Método Actual

Si bien los métodos iterativos para la resolución de sistemas lineales pueden lograr una convergencia que requiera una cantidad de iteraciones relativamente baja

(utilizando menos operaciones de punto flotante), existe la posibilidad de que los utilizados no converjan, o que sólo logren convergencia para estructuras muy particulares de una matriz. Por ejemplo, el clásico método de Jacobi requiere que la matriz sea *fuertemente diagonal dominante*, es decir:

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|.$$

Otros tipos de métodos iterativos utilizan la teoría de los espacios de Krylov: comienzan con una propuesta de solución inicial x e iteran sobre la misma, minimizando el residuo $(Ax - b)$. El Gradiente Conjugado Cuadrado Precondicionado es un método que sigue este estilo, pero incluso si se consigue un buen precondicionador, la convergencia puede ser errática.

Además, es común con los problemas relacionados a la ecuación de Schrödinger, sobre todo cuando se utilizan más combinaciones de momentos angulares l, que la estructura de la matriz requiera un precondicionador más difícil de calcular que uno diagonal por bloques. Esto resulta en una mayor cantidad de iteraciones sobre el sistema.

Otra particularidad de resolver sistemas lineales relacionados con las simulaciones atómicas es que la cantidad de funciones base a utilizar (lo que define el tamaño del sistema) no se conoce de antemano, por lo que es necesario encontrar soluciones para varios sistemas, buscando la convergencia de la sucesión de secciones eficaces para asegurar la estabilidad de la onda que soluciona la ecuación de Schrödinger. Esto se traduce en realizar muchas veces la tarea de conseguir un precondicionador y utilizar un método iterativo para diversos tamaños de matriz; un proceso completamente ineficiente.

1.5. Contribuciones y Estructura de la Tesis

Las contribuciones y los resultados mostrados en esta Tesis fueron presentados en forma de posters, exposiciones orales y publicaciones de proceedings [33, 34, 35, 36].

En el Capítulo 2 se presentan los fundamentos de álgebra lineal numérica y computación de alto desempeño que sirven de introducción al tema y se muestran las librerías y lenguajes de programación utilizados para la solución de sistemas lineales con métodos directos de alto desempeño. En el Capítulo 3 se presenta la propuesta para resolver los múltiples sistemas lineales relacionados con la Ecuación de Schrödinger (y otros problemas con una estructura similar) mediante el *método de matrices latentes* (matrices cuya dimensión no es conocida *a priori*, pero cuya estructura en bloques y tiempo de cómputo se conocen), utilizando la descomposición QR de alto desempeño. En el Capítulo 4 se presentan los diferentes experimentos numéricos realizados para mostrar la eficiencia del método diseñado la solución de sistemas lineales con matrices latentes, junto con la solución de dos problemas de colisiones atómicas: doble fotoionización de las partículas de helio y agua. Finalmente, en el Capítulo 5 se presentan las conclusiones finales y se expone el posible trabajo a realizar en el futuro.

Capítulo 2

Álgebra Lineal Numérica de Alto Desempeño

Los sistemas de ecuaciones lineales se presentan en prácticamente todas las áreas de la matemática, desde la definición de conceptos básicos como el punto, la recta y el plano, hasta la resolución aproximada de ecuaciones en derivadas parciales. La forma de resolverlos ha sido objeto de estudio desde tiempos antiguos (existen escritos chinos del año 200 A.C. en los cuales figura el método de eliminación gaussiana [37]), con la mejora constante de los algoritmos más eficientes, gracias a los avances en el hardware existente.

En este capítulo se introducirán algunos métodos de resolución de sistemas lineales y se estudiarán algunos de sus algoritmos para conocer su cantidad de operaciones, luego se procederá describir los conceptos a tener en cuenta cuando se trabaja en Álgebra Lineal Numérica de Alto Desempeño, seguido de la descripción de librerías de cómputo que permiten conseguir algoritmos eficientes para resolver sistemas lineales de gran porte.

2.1. Resolución de Sistemas Lineales

Un sistema lineal se representa de forma matricial como

$$Ax = b,$$

donde A es una matriz de dimensión $n \times n$, b es un vector de m dimensiones (ambos conocidos) y resolverlo implica encontrar $x \in \mathbb{K}^n$ tal que $Ax = b$, donde utilizaremos $\mathbb{K}$ para describir un cuerpo como el de los números reales $\mathbb{R}$ o los complejos $\mathbb{C}$.

Los sistemas lineales de interés para las aplicaciones físicas de la matemática suelen tener un gran porte, en el orden de los cientos de miles de dimensiones, siendo representados por matrices con una cantidad de información que puede superar *1 Terabyte* de datos. Cuando se trata de resolver sistemas muy grandes, la cantidad de operaciones aritméticas de punto flotante (*flops*) realizada es crucial, ya que permite estimar si un problema puede resolverse en un tiempo razonable.

Los sistemas lineales más fáciles de resolver (luego de los sistemas relacionados con matrices diagonales) son los que se relacionan con matrices triangulares.

Durante el desarrollo de esta tesis se referirá a las matrices relacionadas con sistemas lineales como *cuadradas* (misma cantidad de filas y columnas), pero muchos de estos resultados valen además para matrices rectangulares, sólo que sus soluciones pueden no existir o no ser únicas.

Definición 1. Una matriz $A \in \mathbb{K}^{n \times n}$ es triangular inferior (superior) si $a_{i,j} = 0$ para $i < j$ $(i > j)$.

De manera visual, una matriz de la forma

$$L = \begin{bmatrix} l_{0,0} & & & & 0 \\ l_{1,0} & l_{1,1} & & & \\ l_{2,0} & l_{2,1} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n-1,0} & l_{n-1,1} & \dots & l_{n-1,n-2} & l_{n-1,n-1} \end{bmatrix}$$

es triangular inferior y, análogamente, una matriz triangular superior tiene la forma

$$U = \begin{bmatrix} u_{0,0} & u_{0,1} & u_{0,2} & \dots & u_{0,n-1} \\ & u_{1,1} & u_{1,2} & \dots & u_{1,n-1} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-2,n-1} \\ 0 & & & & u_{n-1,n-1} \end{bmatrix}$$

Si todos los elementos diagonales de esta matriz son no nulos, entonces el sistema lineal asociado tiene solución y es única. Si se desea encontrar la solución de un sistema lineal cuya matriz asociada es triangular inferior o superior, es muy sencillo elaborar un algoritmo y se provee un ejemplo para ambos tipos de sistemas a continuación:

Algoritmo 1: Algoritmo clásico para la solución de un sistema triangular inferior.

Sea $A \in \mathbb{K}^{n \times n}$ triangular inferior, $b \in \mathbb{K}^n$.
Definir $x \leftarrow b$;
for *i=0, . . . , n-1* **do**
  for *j=0, . . . , i-1* **do**
    $x_i \leftarrow x_i - a_{i,j}x_j$;
  end
  $x_i \leftarrow x_i/a_{i,i}$;
end

Algoritmo 2: Algoritmo clásico para la solución de un sistema triangular superior.

Sea $A \in \mathbb{K}^{n \times n}$ triangular superior, $b \in \mathbb{K}^n$.
Definir $x \leftarrow b$;
for *i=n-1, n-2, . . . , 0* **do**
  for *j=n-1, . . . , i+1* **do**
    $x_i \leftarrow x_i - a_{i,j}x_j$;
  end
  $x_i \leftarrow x_i/a_{i,i}$;
end

Cualquiera de estos dos algoritmos precisa del orden de n^2 *flops*, y la notación para esto es $\mathcal{O}(n^2)$. Para resolver sistemas densos (pocos elementos nulos) es necesario requerir a técnicas más elaboradas, como las descomposiciones matriciales.

2.2. Descomposiciones Matriciales

La solución de sistemas lineales densos de gran porte requiere una gran cantidad de operaciones y las computadoras utilizan aritmética de punto flotante, lo que hace imposible representar de forma exacta a todos los números y es necesario mantener bajo control los errores realizados a medida que se realizan cálculos. Por estas razones, históricamente, se ha desarrollado una diversa gama de métodos de descomposición para resolver distintos tipos de sistemas lineales, los cuales son la base para el álgebra lineal que realizan las computadoras, ya que necesitan una cantidad de operaciones mucho menor que el método clásico de buscar la matriz inversa de A y multiplicar por el vector b, además de reducir el error numérico.

2.2.1. Factorización LU

La primera descomposición a tratar en este trabajo es la **Factorización LU**, la cual proviene del concepto de la eliminación gaussiana. A partir de una matriz $A \in \mathbb{K}^{n\times n}$, se busca obtener

$$A = LU,$$

donde $L \in \mathbb{C}^{n\times n}$ es una matriz triangular inferior con unos en la diagonal y $U \in \mathbb{C}^{n\times n}$ es triangular superior (en otras versiones del algoritmo, los unos se encuentran en la matriz U, pero no se pierde generalidad al asumir cualquiera de las dos formas).

$$\begin{bmatrix} a_{0,0} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{1,0} & 1 & 0 \\ l_{2,0} & l_{2,1} & 1 \end{bmatrix} \begin{bmatrix} u_{0,0} & u_{0,1} & u_{0,2} \\ 0 & u_{1,1} & u_{1,2} \\ 0 & 0 & u_{2,2} \end{bmatrix}.$$

Bajo ciertas condiciones, es posible demostrar que esta descomposición existe y es única. Para eso, es necesario definir el concepto de *menor principal*:

Definición 2. Para una matriz $A \in \mathbb{R}^{n\times n}$ y $k \leq n$, su k *menor principal* se define como la submatriz $A_{TL} \in \mathbb{R}k \times k$ formada por las primeras k filas y columnas de A.

Luego de esta definición, se puede presentar el siguiente:

Teorema 1. Sea $A \in \mathbb{K}^{n\times n}$. Entonces una matriz triangular superior U puede producirse mediante eliminación gaussiana a partir de esta matriz. Luego, A posee una factorización de la forma $A = LU$, donde L es una matriz triangular inferior con unos en la diagonal y sus elementos no nulos son los multiplicadores de la eliminación gaussiana. Además, si todos los menores principales de A son invertibles, entonces esta descomposición es única.

Esta es una de las primeras descomposiciones matriciales vistas en cualquier curso de álgebra lineal numérica por su facilidad de cálculo, la intuición debajo de la misma (ya que se obtiene siguiendo los pasos de la eliminación gaussiana) y el hecho de que permite expresar al sistema lineal como la solución sucesiva de dos sistemas triangulares, cuya cantidad de operaciones se vuelve despreciable en relación a su conteo operacional: $\mathcal{O}(\frac{2}{3}n^3)$ *flops* para una matriz $n \times n$, un orden mayor al de la resolución de un sistema triangular.

En el Algoritmo 3 se presenta un ejemplo en pseudocódigo de los pasos necesarios para obtener la descomposición LU de A.

Algoritmo 3: Algoritmo clásico para la descomposición LU.

```
for i=0, ..., n-1 do
    α ← a_{i,i} ;
    for j=i+1, ..., n-1 do
        a_{j,i} ← a_{j,i} / α ;
        for p=i+1, ..., n-1 do
            a_{j,p} ← a_{j,p} − a_{j,i} a_{i,p} ;
        end
    end
end
```

Luego de obtener la descomposición LU de la matriz A, el sistema $Ax = b$ se convierte en $LUx = b$ y entonces se resuelven dos sistemas triangulares para conseguir la solución final: $Ly = b$ y $Ux = y$, cuyo costo computacional es despreciable en comparación con el de factorizar la matriz.

Existen dos posibles problemas al calcular esta descomposición:

- la posibilidad de que un *pivot* (elemento de la diagonal) se convierta en cero, no pudiendo continuar con la resolución del sistema
- la inestabilidad numérica del algoritmo que puede surgir del hecho de que los valores fuera de la diagonal sean muy grandes en relación a la misma.

Por estas razones, se recurre a la permutación de filas de la matriz, una estrategia conocida como *pivoteo parcial*.

El algoritmo de descomposición LU que utiliza pivoteo parcial no es muy distinto al clásico, sólo necesita un paso anterior a la operación de reducción por filas: en el paso i, buscar el elemento $a_{h,i}$ (con $h \in \{i, \ldots, n-1\}$) tal que $|a_{h,i}| \geq |a_{k,i}|, \forall k \in \{i, \ldots, n-1\}$ y luego intercambiar las filas h e i mediante la premultiplicación de A por la matriz elemental correspondiente, que es igual a la identidad con las filas h e i intercambiadas. Para ahorrar trabajo en la cantidad de operaciones, sólo se intercambia las filas de la matriz, no se realiza la premultiplicación de manera explícita.

Algoritmo 4: Algoritmo para la descomposición LU con pivoteo parcial.

```
for i=0, ..., n-1 do
    Encontrar h t.q. |a_{h,i}| ≥ |a_{k,i}|, ∀k ∈ {i, ..., n−1} e intercambiar filas h e i
     de A si es necesario. ;
    α ← a_{i,i} ;
    for j=i+1, ..., n-1 do
        a_{j,i} ← a_{j,i} / α ;
        for p=i+1, ..., n-1 do
            a_{j,p} ← a_{j,p} − a_{j,i} a_{i,p} ;
        end
    end
end
```

El resultado final de aplicar este algoritmo es que se obtienen 3 matrices: P, L y U con P una matriz de permutación (identidad con las filas intercambiadas), L triangular inferior y U triangular superior, tales que $PA = LU$. Nuevamente, el proceso para resolver $Ax = b$ es similar al anterior, incurriendo en aproximadamente

la misma cantidad de operaciones, ya que la premultiplicación por P sólo requiere el intercambio de filas, razón por la cual no se la almacena explícitamente, si no que se guarda una lista que tiene los índices correspondientes a las permutaciones realizadas. Es importante notar que tampoco se obtienen explícitamente dos matrices L y U, ya que ambas pueden depositarse en la misma matriz A, ahorrando mucho almacenamiento en memoria.

2.2.2. Factorización QR

La segunda descomposición a presentar es la **Factorización QR** donde, para una matriz $A \in \mathbb{K}^{n\times n}$, se busca representarla como

$$A = QR,$$

donde $Q \in \mathbb{K}^{n\times n}$ es una matriz *ortogonal* (para el caso de $\mathbb{K} = \mathbb{R}$) o *unitaria* ($\mathbb{K} = \mathbb{C}$), i.e. $QQ^T = I$, lo que representa que esta matriz, multiplicada por su transpuesta (en $\mathbb{R}$, transpuesta conjugada en $\mathbb{C}$), resulta en la matriz identidad. Luego, $R \in \mathbb{K}^{n\times n}$ es una matriz triangular superior. La descomposición QR es normalmente una de las mejores formas de resolver el problema de aproximación por cuadrados mínimos, por lo que suele aplicarse a sistemas rectangulares. Durante el transcurso de esta tesis, se utilizará el término *ortogonal* incluso si la matriz es compleja y se entenderá que la misma es *unitaria*.

Resolver un sistema ortogonal precisa de la multiplicación por la matriz transpuesta conjugada, el cual lleva $\mathcal{O}(n^3)$ operaciones. Luego, el resto del conteo operacional se atribuye al proceso de la factorización matricial, con un orden de $\mathcal{O}(\frac{4}{3}n^3)$ operaciones, el doble del conteo de la descomposición LU.

Se deja, para mayor formalidad, la proposición que define la descomposición:

Proposición 1. Sea $A \in \mathbb{K}^{n\times n}$, entonces existe una matriz $Q \in \mathbb{K}^{n\times n}$ tal que $QQ^H = I$ y una matriz triangular superior $R \in \mathbb{K}^{n\times n}$ tales que $A = QR$.

Existen 3 maneras de conseguir la descomposición QR de una matriz:

- Ortonormalización de Gram-Schmidt.
- Rotaciones de Givens.
- Transformaciones/Reflexiones de Householder.

Para este trabajo se tratará el último método, ya que consigue una mejor estabilidad numérica y puede obtenerse una mejor velocidad de cálculo, pero los 3 métodos siguen el mismo concepto: avanzar por cada columna de la matriz A y buscar transformaciones ortogonales que la lleven a una forma triangular R, para luego juntar todas las transformaciones en una única matriz Q. Para conocer más sobre los primeros dos métodos de descomposición QR, puede referirse a [38].

Una *transformación o reflexión de Householder* (ambos nombres son utilizados indistintamente) toma un vector y lo refleja a través de un hiperplano. Se encadenarán estas transformaciones por cada columna de A para convertir en ceros todos los elementos debajo de la diagonal.

Dado $u \in \mathbb{K}^n$, se considera el hiperplano $\mathcal{H} = u^\perp = \{v \in \mathbb{K}^n | u^H v = 0\}$ y se define $Q \in \mathbb{K}^{n\times n}$ como la reflexión de $\mathbb{K}^n$ a través de $\mathcal{H}$. Puede escribirse $x \in \mathbb{K}^n$ como combinación lineal de u y v para algún $v \in \mathcal{H}$, $x = \alpha u + \beta v$, con $\alpha, \beta \in \mathbb{K}$. Entonces existe una transformación lineal $Q : \mathbb{K}^n \to \mathbb{K}^n$ de la forma $Q(\alpha u + \beta v) = -\alpha u + \beta v$ o, equivalentemente, tal que:

$$Qu = -u, \quad Qv = v, \quad \forall v \in \mathcal{H}.$$

Además, esta transformación tiene una fórmula cerrada, definida en la siguiente proposición:

Proposición 2. Sea $u \in \mathbb{K}^n$ con $u \neq 0$. Si $Q = I - \rho u u^H$, donde se definen $\rho = \frac{2}{\|u\|_2^2}$ y $\|u_2\|_2^2 = \sum_{i=0}^{n-1} u_i^2$, entonces

- $Qu = -u$,
- $Qv = v$ si $u^H v = 0$,
- $Q = Q^H$,
- $Q = Q^{-1}$.

La matriz $Q = I - \rho u u^H$ es la reflexión de Householder para cualquier $u \neq 0$ y en la Figura 2.1 se muestra de manera gráfica cómo afecta Q a un vector x, en el caso que se quiera reflejarlo a la coordenada dada por el vector e_1, donde $e_1 = [1, 0, \ldots, 0]$. En esta figura, v representa al vector u si su norma fuera 1, es decir la dirección de la reflexión.

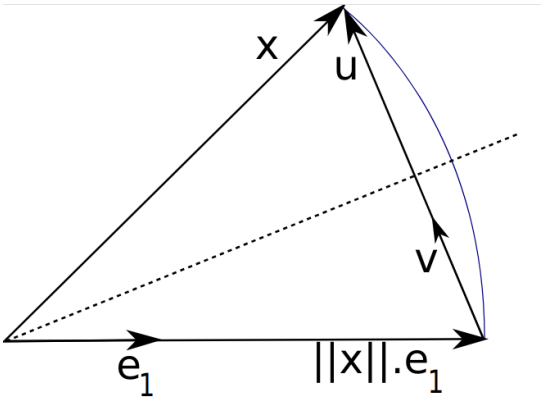

FIGURA 2.1: Efecto de una transformación de Householder. Fuente: https://en.wikipedia.org/wiki/QR_decomposition

Si puede obtenerse una transformación de Householder que lleve x a $\|x\|_2 e_1$ para cualquier $x \in \mathbb{K}^n, x \neq 0$, con $e_1 = [1, 0, \ldots, 0]$, entonces se pueden seguir estos pasos para obtener la descomposición QR de la matriz A:

1. Iterar por la diagonal de $A \in \mathbb{K}^{n \times n}$,
2. En el paso i, tomar el conjunto $\mathcal{I} = \{i, \ldots, n\}$ y el vector formado por la porción de columna de A denotada como $a_{\mathcal{I},i}$, de dimensión $n - i + 1$,
3. Buscar $u \in \mathbb{K}^{n-i+1}$ y su reflexión de Householder relacionada Q_i tal que $Q_i a_{\mathcal{I},i} = \gamma e_1$.
4. Aplicar esa transformación a la matriz A.
5. Luego de $n - 1$ pasos, la matriz A tendrá forma triangular superior (la R buscada) y la secuencia de matrices ortogonales Q_i formará la matriz Q, ya que la multiplicación de matrices ortogonales es ortogonal.

El algoritmo para obtener la reflexión de Householder que lleva un $x \in \mathbb{K}^n$ a $\|x\|_2 e_1$ es el siguiente:

Algoritmo 5: Algoritmo para obtener la transformación de Householder.

Entrada: Sea $x = [x_0, x_1, \ldots, x_{n-1}]$;
Definir $\sigma = \sum_{i=1}^{n-1} x_i^2$, o 0 si $n = 1$;
Definir $\mu = \sqrt{x_0^2 + \sigma}$;
if $x_0 \leq 0$ **then**
  $\gamma = x_0 - \mu$;
else
  $\gamma = -\sigma/(x_0 + \mu)$;
end
Salida: $\rho = 2\gamma^2/(\gamma^2 + \sigma)$, $u = x/\gamma$ y $u_0 = 1$;

El pseudocódigo para obtener la descomposición QR de una matriz cuadrada se expone a continuación:

Algoritmo 6: Algoritmo para la descomposición QR mediante transformaciones de Householder.

Definir $Q = I \in \mathbb{K}^{n \times n}$;
Definir $J = \{0, \ldots, n-1\}$;
for *i=0, . . . , n-1* **do**
  Definir $\mathcal{I} = \{i, \ldots, n-1\}$ y $x \leftarrow a_{\mathcal{I},i}$;
  Obtener u y ρ que definen la matriz de Householder que lleva x a $\|x\|_2 e_1$ mediante el Algoritmo 5 ;
  $A_{\mathcal{I},\mathcal{I}} \leftarrow A_{\mathcal{I},\mathcal{I}} - \rho u (u^H A_{\mathcal{I},\mathcal{I}})$;
  $Q_{\mathcal{J},\mathcal{I}} \leftarrow Q_{\mathcal{J},\mathcal{I}} \rho u u^H$;
end

Una vez obtenida la descomposición QR de la matriz A, el sistema $Ax = b$ se convierte en $QRx = b$ y se puede resolver mediante la resolución del sistema triangular superior $Rx = Q^H b$. Al igual que para la descomposición LU, la resolución del sistema requiere una cantidad despreciable de operaciones de punto flotante en relación a las necesarias para obtener la factorización.

Notar que se utiliza el producto matricial con conjuntos de subíndices, lo que define bloques de cada matriz que se está utilizando. Esto es de utilidad para resumir las operaciones, que requieren la anidación de dos bucles y pueden hacer más difícil de leer algunos algoritmos. Nuevamente, todas las operaciones se hacen sobre la matriz A para poder ahorrar espacio de almacenamiento, guardando los elementos de u en la parte triangular inferior de la matriz, necesitando sólo un vector de longitud $n-1$ para almacenar los distintos ρ.

Si bien es de utilidad conocer estos algoritmos y sus pseudocódigos asociados, los mismos no serán implementados, debido a que es imposible competir en velocidad de cómputo con las soluciones creadas por la comunidad científica hace años y que están en constante actualización. Antes de introducir algunas de estas herramientas, se presentan a continuación las nociones básicas de la computación de alto desempeño.

2.3. Computación de Alto Desempeño

La definición más básica de *algoritmo computacional* es la de "un conjunto de instrucciones que, desarrolladas por una computadora, proveen la respuesta a un problema dado". El objetivo de la Computación de Alto Desempeño (HPC por sus siglas en inglés) es el de buscar algoritmos que resuelvan un problema de la manera más eficiente, dada una arquitectura de cómputo en particular.

La eficiencia puede tener varias ramas, como puede ser obtener mayor velocidad de ejecución (resolver el problema lo más rápido posible) o incurrir en el menor gasto de energía eléctrica para resolver el problema. En este trabajo se concentrará en aprovechar las arquitecturas disponibles para disminuir el tiempo de ejecución y estudiar cómo cambia de acuerdo a la disponibilidad de algunos recursos. Primero, es necesario entender algunos conceptos.

2.3.1. Procesamiento, Memoria y Almacenamiento

Desde el año 1940, la descripción de alto nivel de la mayoría de las computadoras no ha cambiado mucho y siguen la llamada *arquitectura von Neumann*, cuyos cinco componentes primarios son:

- Unidad de Procesamiento Central (CPU)
- Entrada (Input)
- Salida (Output)
- Almacenamiento de trabajo (Working storage)
- Almacenamiento permanente (Permanent storage)

Este diseño presenta memoria que almacena el programa a ejecutar y datos y una unidad de procesamiento que ejecuta las instrucciones del programa, operando sobre los datos extrayendo, ejecutando y almacenando (*fetch - execute - store cycle*). Para aquellas personas que deseen estudiar estos conceptos en mayor detalle, pueden referirse a [39].

La gran mayoría de las arquitecturas de CPU de la actualidad poseen 2 o más procesadores que trabajan en forma simultánea y pueden hacer trabajo en conexión utilizando memorias propias (caché) de diferentes niveles junto con la memoria RAM (almacenamiento de trabajo). Si bien la cantidad de transistores que poseen los chips ha ido disminuyendo constantemente en el tiempo, otros parámetros, como la frecuencia y la utilización de la electricidad han puesto un límite a la velocidad que cada hilo de procesamiento puede alcanzar. Además, muchos de los problemas importantes de la ciencia utilizan cantidades de memoria grandes y esto también pone un límite a la velocidad de ejecución de los programas. Por esa razón, es importante utilizar de la mejor manera los múltiples hilos que los nuevos CPUs ponen a disposición. Un resumen de cómo han evolucionado los procesadores se puede ver en la Figura 2.2.

En este trabajo se utilizarán arquitecturas de uno o dos procesadores con varios núcleos/hilos lógicos que comparten memoria RAM y se intentará obtener el mayor provecho de ellas.

2.3.2. Eficiencia y Bloques

En el camino a lograr una alta eficiencia en el cálculo matricial numérico y aprovechar la creciente disponibilidad de recursos, surge el concepto de la partición de

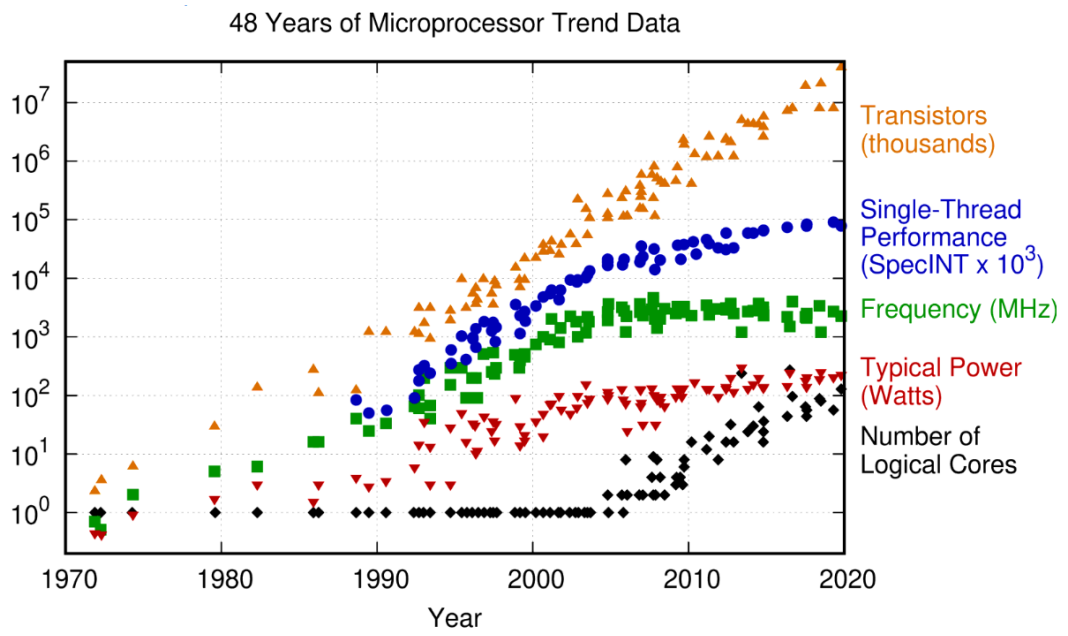

FIGURA 2.2: Evolución de los procesadores a lo largo de 48 años. Fuente: https://github.com/karlrupp/microprocessor-trend-data

las matrices en bloques. Esta idea es simple pero útil, ya que se vuelve una herramienta potente para la demostración de teoremas y la construcción de algoritmos. En términos formales, dada una matriz $A \in \mathbb{R}^{n \times m}$, $\{n_1, \ldots, n_k\}$ y $\{m_1, \ldots, m_h\}$, donde $n_i, m_j > 0, \forall i, j$ tales que $\sum_{i=1}^{k} n_i = n$ y $\sum_{j=1}^{h} m_j = m$, se puede definir una *partición en bloques de* A como

$$A = \begin{bmatrix} A_{0,0} & A_{0,1} & \ldots & A_{0,h-1} \\ A_{1,0} & A_{1,1} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ A_{k-1,0} & \ldots & \ldots & A_{k-1,h-1} \end{bmatrix},$$

donde $A_{i,j} \in \mathbb{R}^{n_i \times m_j}$.

Jack Dongarra expone lo siguiente en [40]:

"Una importante nueva tendencia para la resolución de problemas científicos es la **computación distribuida**. En informática distribuida, los ordenadores conectados por una red se utilizan colectivamente para resolver un sólo problema grande."

El concepto de *paralelización* es esencial para poder lograr un buen rendimiento en cualquier tarea de álgebra lineal numérica. Se refiere a la capacidad de realizar varios cálculos de forma simultánea, operando bajo el principio de que problemas grandes, a veces, pueden dividirse en problemas más pequeños, que normalmente se resuelven al mismo tiempo. Por ejemplo, se puede considerar la multiplicación matricial:

$$\left[\begin{array}{cc|cc} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{array}\right] \left[\begin{array}{cc|cc} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{array}\right] = \left[\begin{array}{cc|cc} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ \hline c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{array}\right]$$

Es posible expresar cada porción del resultado como una suma de varias multiplicaciones matriz-matriz más pequeñas, de la siguiente forma:

$$\begin{bmatrix} c_{0,0} & c_{0,1} \\ c_{1,0} & c_{1,1} \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} + \begin{bmatrix} a_{0,2} & a_{0,3} \\ a_{1,2} & a_{1,3} \end{bmatrix} \begin{bmatrix} b_{2,0} & b_{2,1} \\ b_{3,0} & b_{3,1} \end{bmatrix}$$

$$\begin{bmatrix} c_{0,2} & c_{0,3} \\ c_{1,2} & c_{1,3} \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \begin{bmatrix} b_{0,2} & b_{0,3} \\ b_{1,2} & b_{1,3} \end{bmatrix} + \begin{bmatrix} a_{0,2} & a_{0,3} \\ a_{1,2} & a_{1,3} \end{bmatrix} \begin{bmatrix} b_{2,2} & b_{2,3} \\ b_{3,2} & b_{3,3} \end{bmatrix}$$

$$\begin{bmatrix} c_{2,0} & c_{2,1} \\ c_{3,0} & c_{3,1} \end{bmatrix} = \begin{bmatrix} a_{2,0} & a_{2,1} \\ a_{3,0} & a_{3,1} \end{bmatrix} \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} + \begin{bmatrix} a_{2,2} & a_{2,3} \\ a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} b_{2,0} & b_{2,1} \\ b_{3,0} & b_{3,1} \end{bmatrix}$$

$$\begin{bmatrix} c_{2,2} & c_{2,3} \\ c_{3,2} & c_{3,3} \end{bmatrix} = \begin{bmatrix} a_{2,0} & a_{2,1} \\ a_{3,0} & a_{3,1} \end{bmatrix} \begin{bmatrix} b_{0,2} & b_{0,3} \\ b_{1,2} & b_{1,3} \end{bmatrix} + \begin{bmatrix} a_{2,2} & a_{2,3} \\ a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} b_{2,2} & b_{2,3} \\ b_{3,2} & b_{3,3} \end{bmatrix}$$

Cada uno de los bloques de la multiplicación pueden ser calculados por diferentes procesadores presentes en la arquitectura de cómputo para realizar, en el tiempo que toma una multiplicación, 3 o más de estas. El método para realizar esto se denomina mapeo o *map*. Al terminar por separado cada resultado, se juntan todos los resultados en la matriz final, algo que se conoce como método de reducción o *reduce*. Estos dos métodos pueden realizarse en conjunto por la vectorización de operaciones y el uso de varios núcleos de cómputo trabajando en conjunto y se puede, además, distribuir el trabajo necesario para completar un proceso entre diferentes procesadores con memoria compartida. Las próximas secciones de este capítulo presentarán herramientas y paradigmas para aprovechar al máximo las arquitecturas disponibles.

2.4. Basic Linear Algebra Subprograms

Todo algoritmo de álgebra lineal en su forma básica se desarrolla en una seguidilla de bucles o *loops*, ya que es la forma más simple de entender el trabajo que se realiza (y facilita el conteo operacional). En el caso de la descomposición LU, este pseudocódigo fue propuesto por Gauss en el siglo 19. Por muchos años, este tipo de procesos constituyó el estado del arte en cuanto a rendimiento o *performance* de las tareas que se realizaban, pero con el avance del hardware disponible se buscaron nuevas formas de aprovechar los recursos, para lograr resolver problemas cada vez más grandes en la misma cantidad de tiempo. Con este objetivo, los algoritmos en bucles fueron lentamente reemplazados por algoritmos en bloques. La cantidad de *flops* por unidad de tiempo es actualmente la forma de medir la performance de un algoritmo.

Con el correr del tiempo, los microprocesadores fueron incrementando su capacidad de realizar flops. A finales de 1971, Intel fabricó el primer microprocesador en un chip disponible comercialmente, *i4004*. Podía realizar 92.000 instrucciones por segundo y hoy en día, el procesador *i9 9900* de Intel, para computadoras de casa, tiene un pico de alrededor de 412 millones de instrucciones por segundo [41], lo que se traduce en, al menos, un factor de 4470 en la diferencia de velocidades de procesamiento. Esta diferencia es mayor aún si tenemos en cuenta que las instrucciones se han hecho más complejas con el correr del tiempo, con la posibilidad de realizar 2 flops por ciclo de reloj del procesador y colocar más de un par de valores a los cuales aplicarles una operación, algo que se conoce como *vectorización*.

Es importante tener en cuenta que los flops son un indicador muy simple y crudo de la cantidad de trabajo que está involucrada en el desarrollo de un algoritmo, ya que ignora varias de las otras tareas que se deben realizar. La más importante de ellas es la obtención de datos desde la memoria para realizar operaciones, seguida de la operación de guardado en memoria de los resultados. En la mayoría de las computadoras, las operaciones de acceso a memoria (o *memops*) son más lentas que las flops y las mismas deben realizarse prácticamente la misma cantidad de veces, por lo que la velocidad de un algoritmo puede verse fuertemente afectada por la organización del tráfico de memoria.

De todas estas ideas surgió la librería **Basic Linear Algebra Subprograms (BLAS)** [42], un desarrollo de rutinas de bajo nivel para realizar operaciones comunes de álgebra lineal numérica, tales como suma o copia de vectores, multiplicación por escalares, productos internos, combinaciones lineales y multiplicaciones matriz-vector o matriz-matriz. Fueron construidas en el lenguaje `Fortran` en 1979 y son el principal bloque de construcción de cualquier proyecto que contenga alguna porción que requiera álgebra lineal, como por ejemplo `Mathematica`, `Octave`, `MATLAB` o la librería `Numpy` de `Python`, ya que BLAS se ha convertido en una especificación general más allá de su lenguaje de origen.

La funcionalidad de BLAS se encuentra organizada en 3 "niveles":

- **Nivel 1**: Las rutinas en este nivel se definen como *operaciones vectoriales* y entre ellas se encuentran el producto interno de vectores $x.y = \sum_{i=1}^{n} x_i y_i$, las normas vectoriales como $\|x\|_2 = (\sum_{i=1}^{n} x_i^2)^{\frac{1}{2}}$ y el producto escalar y suma $y \leftarrow \alpha x + y$, también conocida como **axpy**. Si bien son operaciones simples, el problema aquí es que la cantidad de *memops* requeridas es casi la misma que la cantidad de *flops*, algo muy ineficiente.

- **Nivel 2**: Aquí se encuentran las *operaciones matriz-vector*, un poco más eficientes que las anteriores, entre las que se encuentran, por ejemplo, la multiplicación matriz vector generalizada o **gemv** $y \leftarrow \alpha Ax + \beta y$ y la resolución del sistema triangular $Tx = y$.

- **Nivel 3**: El último nivel de BLAS contiene las *operaciones matriz-matriz*, entre las que se incluye la multiplicación matricial generalizada **gemm**: $C \leftarrow \alpha AB + \beta C$, donde además A y B pueden estar transpuestas o conjugadas. Este nivel es el más eficiente de todos en la relación *memomps* vs. *flops* y es el que consigue la mejor performance.

En base a esta definición en niveles, queda claro que el objetivo para lograr la mejor performance para resolver problemas de álgebra lineal numérica es encontrar alternativas que requieran el uso, en su mayoría, de operaciones de Nivel 3.

2.4.1. Librerías

LAPACK

Implementaciones eficientes de los 3 tipos de operaciones descritos anteriormente permitieron la capacidad de programar rutinas que facilitaran la utilización de estrategias del álgebra lineal numérica en sistemas de gran porte y en diversas arquitecturas computacionales. La librería con el conjunto más importante de estas rutinas y la más usada hoy en día es LAPACK (Linear Algebra PACKage), desarrollada en conjunto por las siguientes instituciones estadounidenses:

- University of Tennessee;
- University of California, Berkeley;
- University of Colorado Denver;
- NAG Ltd.

LAPACK fue lanzada por primera vez en 1992 bajo una licencia abierta [43]. Su implementación original fue hecha en el lenguaje `Fortran` y provee rutinas para resolver sistemas lineales, calcular soluciones de cuadrados mínimos, obtener autovalores y autovectores, además de conseguir descomposiciones en valores singulares. Además, proveen funciones para el cálculo de las factorizaciones de matrices asociadas con todos los problemas mencionados anteriormente. Pueden aplicarse en matrices densas y con forma de banda, además de funcionar para matrices reales y complejas, en simple y doble precisión.

La implementación original de LAPACK está disponible en forma abierta bajo una licencia BSD gracias a Netlib[1], siendo auspiciadas por empresas como Mathworks e Intel. Gracias a esto, diversas organizaciones y compañías han trabajado en agregar funcionalidad o adaptabilidad para ciertas arquitecturas, generando nuevas librerías. En esta tesis se utilizaron dos de ellas: OpenBLAS y MKL de Intel.

OpenBLAS

OpenBLAS [2] es una implementación de código abierto (open-source) de BLAS y LAPACK con varias optimizaciones agregadas para tipos específicos de procesadores, desarrollada en el lenguaje `C` en el Laboratorio de Software Paralelo y Ciencias de la Computación del Institute of Software, Chinese Academy of Sciences (ISCAS).

OpenBLAS surgió a partir de la librería GotoBLAS2, desarrollada por Kazushige Goto en el Texas Advanced Computing Center de la Universidad de Texas en Austin. Actualmente, su principal desarrollador y mantenedor es Zhang Xianyi [44]. Esta librería se puede instalar fácilmente en sistemas operativos Windows, Linux y macOS.

Intel oneAPI MKL

La Intel oneAPI Math Kernel Library, más comúnmente conocida como Intel Math Kernel Library (Intel MKL), es una librería de rutinas matemáticas para aplicaciones científicas, de ingeniería y finanzas. Posee diversas funciones, como Transformadas Rápidas de Fourier (FFT), rutinas de álgebra lineal para matrices dispersas, ajuste de datos, estadística e implementaciones personalizadas de BLAS y LAPACK.

Fue presentada por primera vez en 2003 y se encuentra bajo desarrollo continuo. Esta librería soporta procesadores de cualquier compañía, pero está optimizada para funcionar más rápido sobre procesadores Intel y está disponible en sistemas operativos Windows, Linux y macOS. Además de su implementación común, ofrecen una versión open-source disponible en la plataforma Github [3]. Debido a que los diversos procesadores utilizados en el trabajo de esta tesis pertenecían a la empresa Intel, se decidió continuar con esta librería para la mayoría de los experimentos numéricos.

[1] `http://www.netlib.org/lapack/`
[2] `http://www.openblas.net/`
[3] `https://github.com/oneapi-src/oneMKL`

2.5. Paralelismo Basado en Tareas

El Paralelismo Basado en Tareas o *Task-Based Parallelism* es un paradigma de paralelismo en memoria compartida conceptualmente simple. Su objetivo es separar un cómputo en las tareas que lo componen, estableciendo dependencias entre ellas, para luego ejecutarse de forma concurrente. De esta forma, el flujo de datos y uso de recursos está guiado por las dependencias entre nuestras tareas, e.g. si la tarea B requiere datos generados por la tarea A, entonces la tarea B *depende de* la tarea A y no puede ejecutarse hasta que ella no termine.

Las tareas y sus dependencias pueden verse como nodos y aristas, respectivamente, de un grafo acíclico dirigido (o *DAG* por sus siglas en inglés, *directed acyclic graph*) y el mismo puede recorrerse en su orden topológico, ejecutando las tareas en cada nodo a medida que se avanza y se va cumpliendo con sus restricciones de ejecución.

Afortunadamente, este modelo computacional es trivialmente paralelizable en arquitecturas multi-núcleo actuales. Dado un conjunto de tareas interdependientes y otro de hilos computacionales, cada hilo elige de forma repetitiva una tarea que satisfaga todas sus dependencias en el DAG y la ejecuta, quedando a la espera de que otros hilos terminen sus tareas en caso que no haya ninguna disponible, hasta que todas las tareas del DAG se completen.

Podemos explicar este paradigma con un ejemplo simple: la Descomposición LU sin pivoteo de una matriz de 3×3 bloques.

$$A = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} \\ A_{1,0} & A_{1,1} & A_{1,2} \\ A_{2,0} & A_{2,1} & A_{2,2} \end{bmatrix} = \begin{bmatrix} L_{0,0} & 0 & 0 \\ L_{1,0} & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & L_{2,2} \end{bmatrix} \begin{bmatrix} U_{0,0} & U_{0,1} & U_{0,2} \\ 0 & U_{1,1} & U_{1,2} \\ 0 & 0 & U_{2,2} \end{bmatrix} = LU$$

Es posible tener una idea de las tareas a realizar viendo el producto final:

$$LU = \begin{bmatrix} L_{0,0}U_{0,0} & L_{0,0}U_{0,1} & L_{0,0}U_{0,2} \\ L_{1,0}U_{0,0} & L_{1,0}U_{0,1} + L_{1,1}U_{1,1} & L_{1,0}U_{0,2} + L_{1,1}U_{1,2} \\ L_{2,0}U_{0,0} & L_{2,0}U_{0,1} + L_{2,1}U_{1,1} & L_{2,0}U_{0,2} + L_{2,1}U_{1,2} + L_{2,2}U_{2,2} \end{bmatrix}$$

Luego, puede describirse la solución (y las tareas necesarias), en 3 pasos o niveles, de la siguiente manera:

Paso 1

Se comienza con el bloque $A_{0,0}$, calculando su factorización LU, o sea $A_{0,0} = L_{0,0}U_{0,0}$. Ésta es la única tarea que no posee dependencias (salvo la existencia de los datos).

Tarea(s) [Dependencia(s)]:

- `SolveLU(`$A_{0,0}$`)` [Ninguna]

Calcular la fila 0 y la columna 0 de bloques a partir de este resultado es simple, si se tienen $L_{0,0}$ y $U_{0,0}$:

$$U_{0,j} = L_{0,0}^{-1} A_{0,j}, \qquad L_{j,0} = A_{j,0} U_{0,0}^{-1}, \quad j = 1, 2.$$

Es importante tener en cuenta que estas matrices nunca se invierten de manera explícita; se realiza una operación de solución de sistemas lineales a izquierda (`SolveLeft`$(L_{0,0}, A_{0,j})$) y otra a derecha (`SolveRight`$(A_{j,0}, U_{0,0})$).

Al terminar este paso, se obtienen los bloques

$$\begin{bmatrix} L_{0,0} & 0 & 0 \\ L_{1,0} & \star & \star \\ L_{2,0} & \star & \star \end{bmatrix} \begin{bmatrix} U_{0,0} & U_{0,1} & U_{0,2} \\ 0 & \star & \star \\ 0 & \star & \star \end{bmatrix}$$

Tarea(s) [Dependencia(s)]:

- `SolveLeft`$(L_{0,0}, A_{0,j})$ [`SolveLU`$(A_{0,0})$], $j = 1, 2$
- `SolveRight`$(A_{j,0}, U_{0,0})$ [`SolveLU`$(A_{0,0})$], $j = 1, 2$

Paso 2

Observar que $A_{1,1} = L_{1,0}U_{0,1} + L_{1,1}U_{1,1}$, y restar lo ya calculado:

$$A_{1,1} - L_{1,0}U_{0,1} = L_{1,1}U_{1,1}$$

Luego se sigue de forma similar al final del paso anterior para obtener $U_{1,2}$ y $L_{2,1}$:

$$U_{1,2} = L_{1,1}^{-1}(A_{1,2} - L_{1,0}U_{0,2}), \quad L_{2,1} = (A_{2,1} - L_{2,0}U_{0,1})U_{1,1}^{-1}.$$

Si bien $A_{2,2}$ todavía no debe factorizarse, se puede actualizar durante este paso, de la misma forma que se hace con el resto de los bloques:

$$A_{2,2} \leftarrow A_{2,2} - L_{2,0}U_{0,2}$$

Se puede crear una nueva tarea para esta actualización, introduciendo la función

$$\texttt{Update}(A_{i,j}, L_{i,k}, U_{k,j}) = A_{i,j} - L_{i,k}U_{k,j}$$

Al finalizar este paso, se tiene

$$\begin{bmatrix} L_{0,0} & 0 & 0 \\ L_{1,0} & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & \star \end{bmatrix} \begin{bmatrix} U_{0,0} & U_{0,1} & U_{0,2} \\ 0 & U_{1,1} & U_{1,2} \\ 0 & 0 & \star \end{bmatrix}.$$

Tarea(s) [Dependencia(s)]:

- `Update`$(A_{i,j}, L_{i,0}, U_{0,j})$ [`SolveRight`$(A_{i,0}, U_{0,0})$, `SolveLeft`$(L_{0,0}, A_{0,j})$], $i, j = 1, 2$,
- `SolveLU`$(A_{1,1})$ [`Update`$(A_{1,1}, L_{1,0}, U_{0,1})$].
- `SolveLeft`$(L_{1,1}, A_{1,2})$ [`SolveLU`$(A_{1,1})$],
- `SolveRight`$(A_{2,1}, U_{1,1})$ [`SolveLU`$(A_{1,1})$],

Paso 3

Luego de los pasos anteriores, sólo resta actualizar y factorizar $A_{2,2}$, que ya había sido actualizada en un paso anterior:

$$A_{2,2} - L_{2,1}U_{1,2} = L_{2,2}U_{2,2}$$

y se consigue terminar con

$$\begin{bmatrix} L_{0,0} & 0 & 0 \\ L_{1,0} & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & L_{2,2} \end{bmatrix} \begin{bmatrix} U_{0,0} & U_{0,1} & U_{0,2} \\ 0 & U_{1,1} & U_{1,2} \\ 0 & 0 & U_{2,2} \end{bmatrix}.$$

Tarea(s) [Dependencia(s)]:

- `Update`$(A_{2,2}, L_{2,1}, U_{1,2})$ [`Update`$(A_{2,2},\ L_{2,0}, U_{0,2})$].
- `SolveLU`$(A_{2,2})$ [`Update`$(A_{2,2}, L_{2,1}, U_{1,2})$]

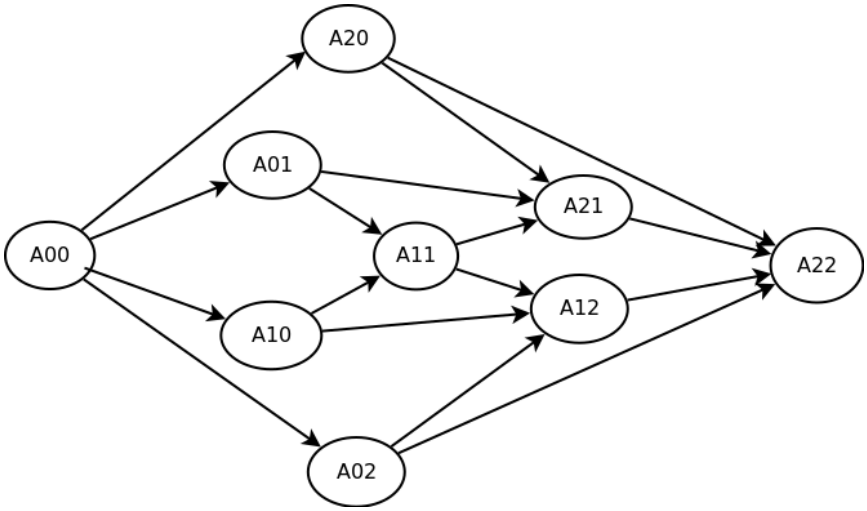

FIGURA 2.3: Dependencia de recursos en la descomposición LU sin pivoteo de 3×3 bloques.

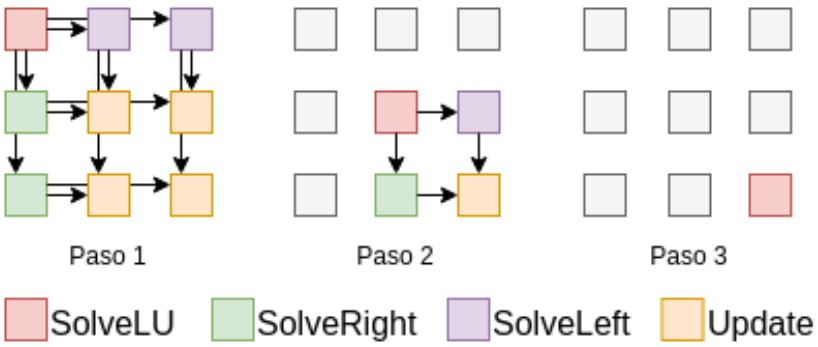

FIGURA 2.4: DAG de recursos y tareas en la descomposición LU sin pivoteo de 3×3 bloques.

La Figura 2.3 muestra la dependencia que existe en los recursos (datos) que se requiere para poder realizar las tareas y en la Figura 2.4 se muestran las dependencias entre recursos y tareas que debería realizar una computadora para finalizar correctamente la descomposición LU sin pivoteo de una matriz de 3×3 bloques. Varias de estas tareas podrían realizarse en paralelo, e.g. las tareas `SolveR` y `SolveL` del paso 1 solamente requieren de la existencia de la descomposición LU de $A_{0,0}$, pero no tienen interdependencias entre ellas, por lo que varios hilos podrían realizarlas al mismo tiempo. Para realizar la descomposición LU del bloque $A_{2,2}$, es necesario esperar a que

todas las tareas anteriores sean completadas (sólo depende del último `Update`, pero debe respetar todas las dependencias hacia atrás).

Si bien el concepto del paralelismo basado en tareas puede ser simple de entender, a través de los años han surgido diferentes formas de implementarlo. Una de las primeras implementaciones de este tipo de sistemas es la extensión Cilk para el lenguaje de programación `C`, en la cual cada hilo de programación se encargaba de generar las próximas tareas que dependían de la tarea que estaba realizando, para ser tomadas por otros hilos que estuvieran libres, una modalidad denominada *work-stealing scheduling* [45].

Las implementaciones más recientes siguen otro esquema:

Esquema de Trabajo

1. Se encapsulan las distintas tareas a realizar en funciones.
2. Se proveen las dependencias entre aquellas tareas que requieren de acceso a los mismos datos, describiendo cuáles parámetros de una tarea son de entrada, salida o ambas.
3. A partir de ésto, las dependencias completas se extraen de forma automática, se genera el DAG, una cola de trabajo, y un *scheduler* se encarga de repartir las tareas a los hilos a medida que se liberan y se van cumpliendo restricciones.

De los pasos enumerados, se desprenden algunas dificultades a tener en cuenta. La primera de ellas es que el usuario es quien debe encargarse de particionar el cálculo en tareas. En teoría, todo programa puede convertirse a una representación en tareas: cada línea puede ser una tarea, cuyas dependencias son generadas por los inputs. Este conjunto de tareas básicas puede reducirse nuevamente, en teoría, fusionando tareas que comparten dependencias y/o recursos, usando una variedad de algoritmos de grafos. Usualmente, la descomposición de un programa en tareas involucra pensar nuevamente en los algoritmos utilizados para que puedan ser mejor aprovechados en un paradigma de cálculo por tareas, como ocurre con la descomposición LU por bloques y en otras descomposiciones matriciales [46]. Un proceso de esta índole requiere mucha evaluación y no es algo que las computadoras puedan realizar de manera sencilla.

Finalmente, otra dificultad que puede surgir es la correcta granularidad de las tareas: si la descomposición del programa resulta en pocas tareas con mucho cómputo, será difícil lograr un buen paralelismo y balance de recursos computacionales. De manera recíproca, si las tareas son muy pequeñas, el tiempo de selección y ordenamiento de tareas, que suele ser constante, puede reducir drásticamente las mejoras de rendimiento obtenidas por la paralelización.

Atacar estas dificultades no es trivial y soluciones que consigan la mejor performance en algunas arquitecturas de cómputo, pueden no ser las óptimas para otras, por lo que se opta por establecer unas pequeñas reglas a seguir al momento de aplicar el paralelismo basado en tareas:

- Cada tarea maximizará la tasa de cómputo y datos requerida,
- Los recursos requeridos para cada tarea deben poder entrar en algún cache del procesador.

El balance entre ambas reglas puede ayudar a optimizar la eficiencia de cada programa.

En la próxima sección se introducen 3 librerías de código abierto (open source) hechas en el lenguaje de programación `C`, que implementan el paralelismo basado en

tareas. Las 3 librerías fueron utilizadas en distintas implementaciones para la resolución de sistemas lineales en esta tesis.

2.5.1. Librerías

QuickSched

`QuickSched` es una librería de código abierto desarrollada por Pedro Gonnet, Aidan Chalk y Matthieu Schaller bajo una licencia GNU General Public License v3.0, disponible en `https://gitlab.cosma.dur.ac.uk/swift/quicksched`. Su funcionamiento tiene dos diferencias principales con el Esquema de Trabajo en 2.5:

- Provee la capacidad de establecer *conflictos* entre tareas, i.e. conjuntos de tareas que pueden ejecutarse en cualquier orden, pero no de manera concurrente.
- El DAG de tareas, dependencias y conflictos debe ser creado por el usuario de manera explícita antes de la ejecución del programa. Esto ayuda a mejorar la velocidad de ejecución y repartición de tareas por parte del *scheduler*.

Si bien el enfoque propuesto por esta librería otorga mucha flexibilidad y, utilizado correctamente, puede mejorar la velocidad de ejecución de tareas en relación a otras librerías disponibles [47] (además de resolver algunos problemas que estas librerías no pueden enfrentar), la creación del grafo de tareas correcto puede ser una carga muy grande para el usuario, ya que no hay forma automática de detectar si las dependencias que programa son realmente las que desea utilizar.

OmpSs

`OmpSs` es un modelo de programación paralela desarrollado y mantenido en el Barcelona Supercomputing Center (BSC) desde hace más de una década [48], que permite explotar el paralelismo en memoria compartida en sistemas de múltiples arquitecturas. Permite desarrollar código en `C` y `C++` y es posible descargarlo en `https://pm.bsc.es/ompss-downloads`.

OmpSs es el primer modelo utilizado en esta tesis que sigue el Esquema de Trabajo en 2.5, que sólo requiere del uso de directivas al compilador en forma de comentarios `#pragma` para facilitar la paralelización en tiempo de corrida del código [4]. `OmpSs` es uno de los modelos que se mantiene a la vanguardia de la paralelización (permite paralelizar en GPUs y FPGAs) y cuyas contribuciones son a veces introducidas al proyecto de la próxima subsección, debido a su similaridad de uso.

OpenMP

Según su manual de especificaciones[5], `OpenMP` es "el estándar de facto para la programación en memoria compartida de sistemas, usada desde sistemas embebidos hasta las supercomputadoras más grandes equipadas con aceleradoras". Funciona en `C/C++` y `Fortran` y, al igual que `OmpSs`, sigue el modelo de programación *fork-join* (un hilo encargado de realizar la división de tareas y juntar los resultados). Un ejemplo gráfico del modelo *fork-join* puede verse en la Figura 2.5. Al igual que `OmpSs`, `OpenMP` sólo requiere el uso de `pragmas` para habilitar su funcionamiento [49].

[4] El compilador es capaz de leer estas directivas y agregar el código necesario para paralelizar estas instrucciones, facilitando la tarea del usuario.

[5] `https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-1.pdf`

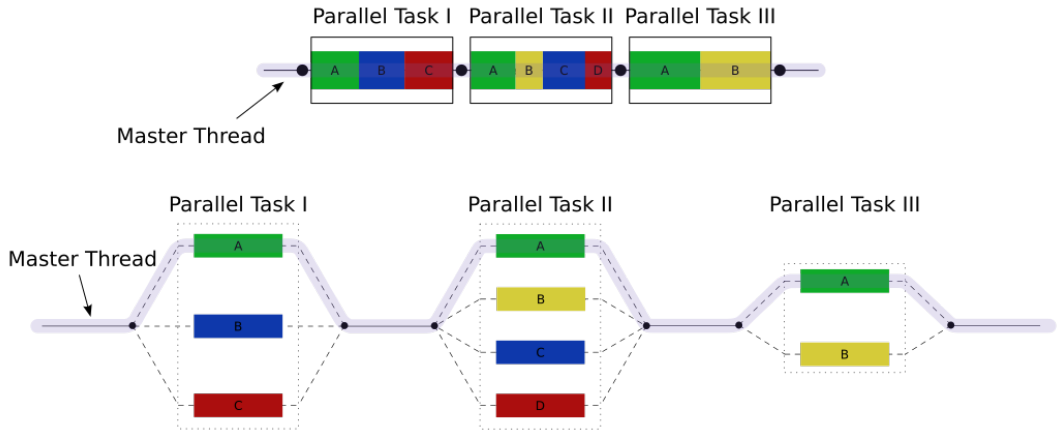

FIGURA 2.5: Ejemplo gráfico del modelo de *fork-join* de `OpenMP`. Fuente: `https://en.wikipedia.org/wiki/OpenMP`

Al momento de la escritura de esta tesis, la versión disponible de `OpenMP` más reciente es la 5.1, pero los algoritmos utilizados en esta tesis fueron programados utilizando la versión 4.5 (la primera en soportar la asignación de prioridades en el paralelismo basado en tareas). Debido a su estabilidad y el hecho de ser un estándar de la industria y la academia, `OpenMP` es la elegida para los últimos experimentos numéricos de este trabajo y los que se realicen en un futuro.

Capítulo 3

Métodos de Álgebra Lineal para Sistemas Latentes

En el problema de Coulomb de pocos cuerpos, el sistema de colisión atómica más simple es el *scattering elástico* de un electrón de un núcleo de hidrógeno. En este proceso, dos partículas cargadas interactúan entre sí, siguiendo las reglas de la interacción de Coulomb de largo alcance del mundo cuántico. Agregar una partícula a este sistema lleva a una ecuación en derivadas parciales de seis dimensiones (las posiciones de las dos partículas en los ejes x, y, z) que no posee una forma analítica y requiere un cálculo numérico. El enfoque más preciso para trabajar con estos sistemas físicos consiste en encontrar la solución numérica de la ecuación de Schrödinger para el problema de pocos cuerpos que representa la colisión. Éste ha sido el método de elección para sistemas de pequeña escala atómica pero a la vez muy complejos. Aún en la actualidad, este problema sigue siendo extremadamente difícil de resolver.

Se considerará la ecuación de Schrödinger para los estados continuos que resulta de las colisiones en un sistema de pocos cuerpos, que puede obtenerse a partir de la solución del problema independiente del tiempo

$$(H - E)\Psi = W\Psi_0, \tag{3.1}$$

donde Ψ_0 es el estado inicial del proceso y E es la energía total del sistema. El operador (posiblemente diferenciable) W es responsable de las transiciones desde el estado inicial conocido Ψ_0 al estado colisional, en el continuo, desconocido Ψ. Para el caso de dos electrones en un núcleo atómico, se considera un sistema de coordenadas dado por los vectores $\boldsymbol{r_1}$ y $\boldsymbol{r_2}$, que representan la posición de cada electrón con respecto al núcleo. De esta manera, la Ec. (3.1) (ecuación en derivadas parciales) tiene seis dimensiones. El enfoque utilizado en esta Tesis para resolver este problema numéricamente es el *método espectral*, donde se introduce un conjunto de funciones base para representar a la solución Ψ. Una de las principales cualidades de este procedimiento es que las consideraciones físicas pueden introducirse directamente en los elementos de la base.

Cualquier método espectral en las coordenadas de los electrones introduce un conjunto base $\{\Xi^L_{nm}(\boldsymbol{r_1}, \boldsymbol{r_2})\}$ para expandir la función de onda Ψ:

$$\Psi(\boldsymbol{r_1}, \boldsymbol{r_2}) = \sum_{n=0,m=0}^{N-1,M-1} x_{n,m}\Xi^L_{n,m}(\boldsymbol{r_1}, \boldsymbol{r_2}). \tag{3.2}$$

Esto representa la aplicación de la Ec. (1.4) a este sistema atómico en particular. En este punto es necesario aclarar que la ecuación depende de forma implícita del momento angular total L. Este hecho está representado por el supraíndice en la

definición de la base. El Hamiltoniano es una matriz latente en este conjunto base y cuyos elementos pueden calcularse de la forma:

$$\boldsymbol{H}^{L}_{n',m',n,m} = \langle \Xi^{L}_{n',m'} | H - E | \Xi^{L}_{n,m} \rangle, \tag{3.3}$$

la cual es una matriz cuadrada de $NM \times NM$ elementos complejos. $\langle x|H-E|y\rangle$ denota el producto interno entre x e y mediante el operador $H - E$, o también puede verse como el producto interno entre x y $(H - E)y$ (denotado como $\langle x, |H - E|y\rangle$). Desde ahora, se utilizará la notación $\boldsymbol{H}$ para referirse al sistema lineal dado por el operador $H - E$.

La solución del sistema lineal $\boldsymbol{Hx} = \boldsymbol{b}$ da un conjunto de coeficientes $\boldsymbol{x} = \{x_{nm}\}$ que determinan la solución Ψ. El vector del lado derecho del sistema lineal, $\boldsymbol{b}$, es la proyección en la base de la acción del operador W sobre el estado inicial Ψ_0.

Los elementos de la matriz del Hamiltoniano $\boldsymbol{H}^{L}_{n'm'nm}$ son integrales en seis dimensiones calculadas *a priori* en el espacio coordenado generado por el conjunto $\{\boldsymbol{r_1}, \boldsymbol{r_2}\}$. La complejidad del cálculo de estos elementos puede reducirse si se tienen en cuenta simetrías físicas al seleccionar elementos de la base. Se asume el uso de coordenadas esféricas para cada electrón: $\boldsymbol{r_i} = (r_i, \theta_i, \phi_i)$ para $i = 1, 2$. Con esta elección, cada elemento de la base Ξ^{L}_{nm} está definido como un producto de dos funciones para un electrón:

$$\Xi^{L}_{n_a n_b}(\boldsymbol{r_1}, \boldsymbol{r_2}) = \frac{S_{n_a l_a}(r_1)}{r_1} \frac{S_{n_b l_b}(r_2)}{r_2} \mathcal{Y}^{LM}_{l_a l_b}(\hat{\boldsymbol{r_1}}, \hat{\boldsymbol{r_2}}), \tag{3.4}$$

donde $1 \leq n_a \leq N_a$ y $1 \leq n_b \leq N_b$ y $\{N_a, N_b\}$ son el número de elementos utilizados para cada conjunto base de cada electrón. Los índices l_a y l_b representan al momento angular de cada electrón y satisfacen $|L - l_a| \leq l_b \leq L + l_a$. Esto implica que, para un dado L, hay varios pares de valores (l_a, l_b). Esto significa que los valores óptimos para estos parámetros no son conocidos *a priori* y, usualmente, se realiza un proceso de *prueba y error* de cálculos hasta que se consigue la precisión requerida.

Normalmente, varios pares de momentos angulares son utilizados para un L dado (por ejemplo, desde 6 a 16) y el tamaño de la base $N_a = N_b$ incluye varias docenas de elementos. Por lo tanto, el tamaño del sistema lineal a resolver puede tener varias cientas de miles de incógnitas. Además, el Hamiltoniano es una matriz densa con entradas complejas que deben calcularse en aritmética de precisión doble, llevando los requerimientos de memoria a los cientos de GigaBytes [50].

El flujo de trabajo comúnmente adoptado para obtener la mejor solución $\Psi(r)$ es resolver el sistema dado por la Ec. (3.1), para una secuencia incremental de valores $N_1 < N_2 < N_3 \ldots$ de N, hasta que se consigue una convergencia *adecuada*. Esta convergencia y el criterio de parada asociado están usualmente determinados por la precisión requerida en el cálculo de la magnitud física, la cual está acotada por datos de experimentación, en los casos en los que los mismos existan.

Este procedimiento presenta dos importantes desventajas. Primero, se puede calcular la matriz completa $A^{(N)}$ para cada $N_1 < N_2 < N_3 \ldots$, entonces varios elementos de la matriz calculan repetidamente, ya que todos los elementos de $A^{(N_i)}$ están incluidos en los cálculos para $N \geq N_i$, para $i = 1, 2, \ldots$. Para evitar estos recálculos costosos, las diferentes matrices $A^{(N_i)}$ pueden almacenarse en el disco duro, pudiendo acceder a los datos desde archivos, ya que las restricciones de memoria pueden alcanzarse rápidamente para matrices de un tamaño mediano. En este escenario, el cálculo es limitado por la transferencia de entrada y salida de datos desde el disco rígido, la cual es lenta. La segunda gran desventaja es que, aunque las librerías para la resolución de problemas de álgebra lineal numérica proveen la funcionalidad para atacar

una gran cantidad de problemas, todas ellas necesitan conocer el tamaño, estructura y datos de las matrices desde el comienzo, en cualquiera de sus varias implementaciones y librerías, sin importar que estén basadas en algoritmos secuenciales, multi-hilo o distribuidos.

3.1. Matrices Latentes

En este trabajo, se introduce un nuevo concepto de matrices, clasificadas como *latentes*. Las principales características de estas matrices son:

- La dimensión de una matriz latente no es conocida *a priori*;
- Una matriz latente tiene una estructura de bloques conocida;
- Cada elemento de una matriz latente puede calcularse con un algoritmo numérico bien definido;
- El tiempo (latencia) de cálculo de cada elemento puede determinarse para un equipo computacional dado.

La física de colisiones atómicas provee ejemplos importantes de problemas que pueden resolverse mediante métodos de resolución de sistemas lineales sobre matrices latentes, para los cuales deben diseñarse algoritmos y software de álgebra lineal numérica adecuados.

Esta tesis se encarga de explorar estos algoritmos y su aplicación a la solución de problemas de Coulomb de Pocos Cuerpos, logrando las siguientes contribuciones:

- Se reformula la solución de un sistema de ecuaciones lineales con datos latentes como un problema de actualización de factorizaciones matriciales.
- Se analiza el paralelismo del problema, proponiendo una solución asíncrona que junte la ejecución de múltiples problemas de actualización, para exponer un mayor grado de concurrencia de tareas.
- Se implementan algoritmos en tres librerías distintas, para conseguir posibles soluciones de problemas de Coulomb de pocos cuerpos: `QuickSched`, `OmpSs` y `OpenMP`.
- Se evalúa la performance de las soluciones de paralelismo de tareas creadas, comparando su rendimiento con aquel de una de las herramientas disponible hasta el momento.

3.2. Solución de Sistemas Lineales Latentes

Se describirá con mayor formalidad el problema subyacente en la solución de problemas de pocos cuerpos de Coulomb, que conecta a los sistemas lineales con los datos latentes.

La búsqueda de la solución al problema descripto al principio de este capítulo lleva a la necesidad de resolver una secuencia de sistemas lineales de la forma $\mathbf{Hx} = \mathbf{b}$, que puede organizarse en un ciclo de "niveles", donde se genera y resuelve el problema en el nivel s; luego, se prueba si la solución obtenida en ese nivel es satisfactoria para el problema global; y, en caso de no serlo, se procede al siguiente nivel $(s+1)$, repitiendo el ciclo (emulando la prueba y error).

1.	Generar los datos iniciales del problema: $A^{(0)}, b^{(0)}$.
2.	Resolver el sistema lineal denso $m_0 \times m_0$, $A^{(0)}x^{(0)} = b^{(0)}$.
3.	Resolver la secuencia de niveles:
	para $s = 1, 2, \ldots$ hasta convergencia
3.1	Generar los datos del problema para el s-vo nivel: $B^{(s-1)}$ de tamaño $n_{s-1} \times m_s$, $C^{(s-1)}$ de tamaño $m_s \times n_{s-1}$, $D^{(s-1)}$ de tamaño $m_s \times m_s$, donde $n_{s-1} = \sum_{k=0}^{s-1} m_k$; y $c^{(s-1)}$ tiene m_s componentes.
3.2	Resolver el sistema lineal $n_s \times n_s$ para el s-vo nivel: $A^{(s)}x^{(s)} = b^{(s)} \equiv \begin{bmatrix} A^{(s-1)} & B^{(s-1)} \\ C^{(s-1)} & D^{(s-1)} \end{bmatrix} \begin{bmatrix} y^{(s-1)} \\ z^{(s-1)} \end{bmatrix} = \begin{bmatrix} b^{(s-1)} \\ c^{(s-1)} \end{bmatrix}$.
	fin bucle

FIGURA 3.1: Algoritmo RLLS.

Para describir este problema de manera formal, se asume que el sistema lineal a ser resuelto inicialmente (i.e., en el nivel 0) está dado por

$$A^{(0)}x^{(0)} = b^{(0)}, \tag{3.5}$$

donde $A^{(0)}$ es la matriz de coeficientes $m_0 \times m_0$, $b^{(0)}$ es el vector del lado derecho, $x^{(0)}$ es la solución buscada, y $b^{(0)}, x^{(0)}$ son ambos de tamaño m_0.

En el caso en que la solución de la Ec. (3.5) no sea satisfactoria, se extiende el sistema lineal a

$$A^{(1)}x^{(1)} = b^{(1)} \equiv \begin{bmatrix} A^{(0)} & B^{(0)} \\ C^{(0)} & D^{(0)} \end{bmatrix} \begin{bmatrix} y^{(0)} \\ z^{(0)} \end{bmatrix} = \begin{bmatrix} b^{(0)} \\ c^{(0)} \end{bmatrix}, \tag{3.6}$$

donde $A^{(1)}$ es de tamaño $n_1 \times n_1$, $n_1 = m_0 + m_1$, y las componentes de 3.6 son particionadas conforme a este tamaño.

Esta dependencia entre los niveles 0 y 1 puede generalizarse a cualquier par de niveles consecutivos, $s-1$ y s, en la forma del algoritmo para la solución de *sistema lineales latentes recurrentes* (RLLS por sus siglas en inglés) mostrada en la Figura 3.1.

El sistema lineal denso en el Paso 2, correspondiente a la Ec. (3.5), puede resolverse mediante cualquier algoritmo de descomposición matricial convencional, como las factorizaciones LU y QR [38], incurriendo en un costo de $2m_0^3/3$ o $4m_0^3/3$ flops, respectivamente. Puede despreciarse el costo de resolver los sistemas triangulares resultantes, ya que sólo necesitan del orden de m_0^2 operaciones. La solución de este sistema lineal, en cualquier CPU, puede obtenerse mediante el uso de cualquier librería de álgebra lineal densa como Intel MKL, PLASMA [51], y FLAME [52].

La solución al sistema lineal en el Paso 3.2 es un poco más desafiante. En general, si se asume la utilización de una factorización QR, una solución simple que ataca este problema en aislamiento requeriría $4n_s^3/3$ flops, con este número creciendo rápidamente con el índice de niveles s y la dimensión de los subproblemas m_j, $j = 0, 1, 2, \ldots$.

Una solución más elaborada puede evitar el gran trabajo de cálculo realizado previamente por el enfoque simple, explotando el hecho de que $A^{(s-1)}$ es una submatriz principal de $A^{(s)}$ para aprovechar una factorización existente de la matriz anterior. En álgebra lineal, esto se conoce como un *problema de actualización*, y se han desarrollado soluciones eficientes de alto rendimiento, algunas de ellas utilizando las descomposiciones LU y QR para matrices densas.

La mayoría de los sistemas lineales pueden ser resueltos de varias formas distintas. En particular, al evaluar si una técnica es la adecuada, se utilizan 3 variables importantes:

- Velocidad (¿qué tan rápido es el algoritmo?)
- Almacenamiento (¿cuál algoritmo usa la menor cantidad de almacenamiento?)
- Estabilidad Numérica (¿las respuestas que da son satisfactorias o siquiera correctas?)

Naturalmente, resulta casi imposible que un algoritmo o técnica sea el mejor en las tres categorías, dejando la decisión final a la persona que debe resolver el problema según sus prioridades.

En este tipo de problemas, la *estabilidad numérica* debe ser una de las principales prioridades, ya que la solución obtenida debe ser luego llevada al dominio de las funciones base y calcular integrales sobre ella. Cuando la solución de un problema se obtiene mediante aritmética de punto flotante, el resultado final es afectado por errores de redondeo, los cuales se van multiplicando a medida que se realizan las distintas operaciones.

La estabilidad numérica de un algoritmo puede medirse utilizando la norma de las matrices resultantes y es importante tener en cuenta que, a medida que se vayan realizando actualizaciones de una descomposición, la misma se puede volver cada vez más inestable.

A continuación, se describirán las descomposiciones LU y QR con actualización en mayor detalle y se discutirán sus ventajas y desventajas.

3.2.1. Actualización de la Descomposición LU

En [53], Quintana-Ortí y Van de Geijn presentan el problema de actualización de la descomposición LU, teniendo en cuenta la siguiente partición de una matriz:

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

conociendo la descomposición LU con pivoteo parcial de la submatriz A, denotada de la siguiente forma: $LU(A) = [L\backslash U, p]$, donde L y U se almacenan en A y p es un vector que simboliza las permutaciones del pivoteo. El objetivo es reutilizar el trabajo realizado sobre A para completar la factorización de M. Se propone el siguiente esquema:

Paso 1: Descomposición LU por bloques de A

La forma de obtener la descomposición LU de alto rendimiento para cualquier matriz $A \in \mathbb{K}^{m \times n}$ es dividirla en bloques

$$A = \begin{bmatrix} A_{0,0} & A_{0,1} & \dots & A_{0,N-1} \\ A_{1,0} & A_{1,1} & \ddots & A_{1,N-1} \\ \vdots & \ddots & \ddots & \vdots \\ A_{M-1,0} & A_{M,1} & \dots & A_{M-1,N-1} \end{bmatrix}$$

de manera que todos tengan una dimensión que pueda entrar en el caché del procesador que se está utilizando. Uno de los posibles procedimientos para obtener la

descomposición LU por bloques se muestra en el Algoritmo 7. Esto puede denotarse como $A \leftarrow \text{LU}(A) = [\{L\backslash U\}, p]$.

Algoritmo 7: Algoritmo para obtener la descomposición LU con pivoteo parcial en bloques.

Definir k de forma que los bloques de dimensión k entren en caché;
for *i=0, . . . , N-1* **do**
  Definir los índices $\mathcal{I} = i+1, \ldots, M-1$ y $\mathcal{J} = i+1, \ldots, N-1$, logrando la división de la matriz A comenzando desde el bloque (i,i) $\begin{bmatrix} A_{i,i} & A_{i,\mathcal{J}} \\ A_{\mathcal{I},i} & A_{\mathcal{I},\mathcal{J}} \end{bmatrix}$;
  Realizar la descomposición LU con pivoteo parcial de la matriz $(m-ik) \times k$: $A^i = \begin{bmatrix} A_{i,i} \\ \hline A_{\mathcal{I},i} \end{bmatrix} = [\left(\begin{array}{c} L\backslash U_{i,i} \\ \hline L_{\mathcal{I},i} \end{array}\right), p_i]$;
  Aplicar pivoteo p_i al resto de la matriz a la derecha $p_i^T \begin{bmatrix} A_{i,\mathcal{J}} \\ \hline A_{\mathcal{I},\mathcal{J}} \end{bmatrix}$;
  Actualizar los bloques a la derecha: $A_{i,\mathcal{J}} \leftarrow L_{i,i}^{-1} A_{i,\mathcal{J}}$;
  Actualizar resto de la matriz: $A_{\mathcal{I},\mathcal{J}} \leftarrow A_{\mathcal{I},\mathcal{J}} - L_{\mathcal{I},i} U_{i,i}$;
end

En este algoritmo es importante notar que, durante la descomposición, se está "avanzando por la diagonal", esto quiere decir que luego de terminar cada paso del bucle, todos los bloques a la izquierda y por arriba de $A_{i,i}$ ya no deben modificarse y por eso son dejados de lado al comienzo de un nuevo paso.

Paso 2: Actualización de B de acuerdo a la descomposición de A

Este es el paso más simple, donde sólo es necesario realizar la sustitución hacia delante:

$$B \leftarrow L^{-1} P(p) B,$$

donde $P(p)$ denota la aplicación de la permutación dada por el vector p.

Paso 3: Factorización de $\begin{bmatrix} U \\ \hline C \end{bmatrix}$

Es necesario continuar con la descomposición, realizando eliminación gaussiana sobre la matriz formada por U y C:

$$\begin{bmatrix} U \\ \hline C \end{bmatrix} \leftarrow \text{LU}\left(\begin{array}{c} U \\ \hline C \end{array}\right) = \left[\left(\begin{array}{c} \bar{L}\backslash\bar{U} \\ \hline \check{L} \end{array}\right), r\right].$$

$\check{L}$ denota al bloque C luego de pasar por la eliminación gaussiana; este paso también es conocido como *pivoteo incremental*. Notar que, dada la naturaleza del pivoteo, L y U pueden ser modificadas y por eso se las reemplaza en la notación por $\bar{L}$ y $\bar{U}$, respectivamente. Si la matriz U fuera completamente densa, esta operación puede requerir una cantidad de operaciones grande, pero al ser triangular superior, es posible aprovechar su estructura para realizar la descomposición con pivoteo, ya que sólo es necesario mirar los coeficientes de C para encontrar pivots. El algoritmo propuesto

en [53] facilita esto, utilizando unos bloques extra para almacenar $\bar{L}$ (cuyo tamaño es muy pequeño en relación a la dimensión total de la matriz original M).

Paso 4: Actualización de $\left[\frac{B}{D}\right]$ de acuerdo a la descomposición de $\left[\frac{U}{C}\right]$

Otro paso de baja complejidad que, gracias a la descomposición teniendo en cuenta la estructura de U, necesita una cantidad de operaciones despreciable con relación a la descomposición LU de M.

$$\left[\frac{B}{D}\right] \leftarrow \left(\frac{\bar{L}}{\check{L}}\right)^{-1} P(r) \left[\frac{B}{D}\right]$$

Paso 5: Descomposición LU por bloques de D

Se repite exactamente lo que se propone en A al bloque D, aplicando el Algoritmo 7. Esto resulta en reemplazar $D \leftarrow \mathrm{LU}(D) = [\{\tilde{L}\backslash\tilde{U}\}, s]$.

Complejidad y Estabilidad Numérica

Al completar el problema de actualización de la descomposición LU con pivoteo parcial por bloques de M, se incurre en una cantidad de operaciones igual a $\frac{2}{3}n^3 + kn_A(\frac{n_A}{2}+n_D)$, donde n_A y n_D son las mayores dimensiones de A y D, respectivamente. Eligiendo un tamaño de bloques k pequeño, la cantidad final de operaciones es muy cercana a la de la descomposición original de la matriz M.

La estabilidad numérica de este algoritmo puede medirse utilizando el cociente $\rho = \frac{\|L\|\|U\|}{\|A\|}$. Cada vez que se realiza la descomposición de LU con pivoteo parcial para un bloque matricial, el coeficiente *rho* va creciendo. Stewart, en su libro Matrix Algorithms, muestra que *rho* puede crecer al ritmo de 2^{n-1} para una matriz de dimensión n [54]. En la práctica, este coeficiente suele estar más cerca de $n^{2/3}$ [53], pero es importante notar que, a medida que la matriz A va creciendo en cantidad de bloques y se deban realizar las actualizaciones correspondientes, estos errores van a multiplicarse y la precisión de la solución del sistema lineal será cada vez peor.

3.2.2. Actualización de la Descomposición QR

En [55], Gunter y Van de Geijn presentan el problema de actualización de la descomposición QR, teniendo en cuenta la siguiente partición de una matriz:

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

conociendo la descomposición QR de la submatriz A, denotada de la siguiente forma: $QR(A) = Q\backslash R$, donde los vectores necesarios para las transformaciones de Householder que dan lugar a Q y la matriz triangular superior R se almacenan en A. El objetivo es reutilizar el trabajo realizado sobre A para completar la factorización de M. Antes de pasar al procedimiento, es importante revisar los conceptos (a través de dos artículos) que permiten que la descomposición QR pueda lograr el mejor rendimiento.

Representación WY

Como se discutió en el Capítulo 2, la mejor forma de lograr una alta performance al momento de resolver problemas es mediante el uso de operaciones de Nivel 3 de BLAS,

pero el enfoque de transformaciones de Householder básico usa productos matriz-vector (Nivel 2). Por esta razón, en [56], Bischof y Van Loan crearon una nueva forma de representar una serie de transformaciones de Householder, que permite utilizar *bloques de transformaciones* y mejorar la eficiencia de la descomposición QR.

Para ilustrar el procedimiento, se toma la descomposición QR de una matriz A:

$$A = [A_0, A_1, A_2] = [Q_0, Q_1, Q_2] \begin{bmatrix} R_{00} & R_{01} & R_{02} \\ 0 & R_{11} & R_{12} \\ 0 & 0 & R_{22} \end{bmatrix},$$

donde A_i y Q_i son conjuntos de columnas de A y Q respectivamente. Cada Q_i es una secuencia de k transformaciones de Householder

$$Q_i = P_i^0 \dots P_i^{k-1}, \text{ tal que } P_i^t = I + u_i^t (v_i^t)^T.$$

Siguiendo el algoritmo básico, las transformaciones se aplicarían una detrás de la otra, resultando en una gran cantidad de productos matriz-vector. Los autores proponen representarlo de otra manera:

$$Q_i = I + W_i Y_i^T,$$

y teniendo en cuenta que la primera iteración puede representarse como $Q_i^0 = P_i^0 = I + u_i^0 (v_i^0)^T$, se define $W_i^0 = u_i^0$ y $Y_i^0 = v_i^0$. Esto permite agrupar k transformaciones y generar la representación de Q_i^k:

$$\begin{aligned} Q_i^k = Q_i^{k-1} P_i^k &= \left(I + W_i^{k-1} (Y_i^{k-1})^T\right) \left(I + u_i^k (v_i^k)^T\right) && 3.7 \\ &= I + \left[W_i^{k-1}, Q_i^{k-1} u_i^k\right] \left[Y_i^{k-1}, v_i^k\right]^T && 3.8 \\ &= I + \left[W_i^{k-1}, u_i^k\right] \left[P_i^k Y_i^{k-1}, v_i^k\right]^T. && 3.9 \end{aligned}$$

De esta manera, pueden tomarse cualquiera de las dos formulaciones para W e Y, correspondientes a la fila 2 y 3 de la Ec. (3.9) y proponer dos algoritmos Algoritmo 8 para generar la representación WY de un conjunto de reflexiones de Householder, que puede aplicarse por bloques para obtener la descomposición QR de la matriz A. El proceso de uno de ellos (la primera opción) queda plasmado en el Algoritmo 9.

Algoritmo 8: Algoritmo para obtener la representación WY de un conjunto de reflexiones de Householder.

Definir $A \in \mathbb{R}^{m \times n}$ en términos de sus columnas, $A = [a_0, \dots, a_{n-1}]$;
for $k = 0, \dots, n-1$ **do**
  $a_k \leftarrow \left(I + WY^T\right)^T a_k$;
  $u, v \leftarrow$ vectores de la reflexión de Householder de a_k;
  $W \leftarrow \left[W, (I + WY^T)u\right]$;
  $Y \leftarrow [Y, v]$;
end

Algoritmo 9: Algoritmo para obtener la descomposición QR, mediante representaciones WY por bloques.

Definir $A \in \mathbb{R}^{m \times n}$ en términos de N bloques de columnas de ancho k, $A = [A_0, \ldots, A_{N-1}]$;
for $k = 0, \ldots, N-1$ **do**
  Definir los índices $\mathcal{I} = kN, \ldots, m$ y $\mathcal{J} = kN, \ldots, n$;
  $W, Y \leftarrow$ matrices obtenidas en el Algoritmo 8 tales que $(I + WY^T)(A_k)_{\mathcal{I},\cdot}$ es triangular superior;
  $(A_k)_{\mathcal{I},\mathcal{J}} \leftarrow \left(I + WY^T\right)^T (A_k)_{\mathcal{I},\mathcal{J}}$;
end

Este algoritmo puede sobreescribir la matriz A con R en su porción triangular superior y los vectores para generar las matrices Y debajo de la diagonal, requiriendo una cantidad de espacio extra igual a una matriz $m \times k$ para almacenar los datos de las N matrices W. La cantidad de operaciones de este algoritmo es la de la descomposición QR multiplicada por un factor de $(1 + \frac{2}{N})$, pero es rica en operaciones matriz-matriz, logrando un mejor rendimiento en arquitecturas actuales.

Representación WY Compacta para Almacenamiento Eficiente

Si bien la representación WY provee la capacidad de resolver la descomposición QR utilizando más multiplicaciones matriz-matriz, su rendimiento, si quiere conservarse Q y R, se ve afectado por el tamaño de k: si es muy grande, entonces el almacenaje extra se convierte casi en el mismo que la matriz A (normalmente este método se aplica para la solución de problemas de cuadrados mínimos, donde se tiene muchas mas filas que columnas), pero si es muy pequeño no se puede conseguir una verdadera aceleración del proceso, ya que sería más similar a la aplicación de transformaciones de Householder.

Por estas razones, en [57], Schreiber y Van Loan proponen una manera más eficiente de almacenar la representación WY de una matriz, llamada la Representación Compacta WY. Esta representación es la que actualmente se encuentra implementada en las librerías que realizan la descomposición QR y necesitan almacenar lo necesario para calcular ambas matrices. Consiste en aprovechar la relación entre las matrices W y Y para llevar $Q = I + WY^T$ a la forma $Q = I + YTY^T$, proveyendo un teorema cuya demostración da lugar al procedimiento para calcular la nueva representación. Aquí se presenta una ligera variación del mismo, que toma en cuenta matrices en el cuerpo de los complejos, puesto que los sistemas estudiados en esta tesis se encuentran allí.

Teorema 2 (Actualización WY Compacta)**.** Sea $Q = I + YTY^H \in \mathbb{C}^{m \times m}$ una matriz hermitiana con $Y \in \mathbb{C}^{m \times j}$ (m>j) y $T \in \mathbb{C}^{j \times j}$ triangular superior. Suponga que $P = I - 2vv^H$ es una matriz de reflexión de Householder con $v \in \mathbb{C}^m$ con $\|v\|_2 = 1$.

Si $Q_+ = QP$, entonces

$$Q_+ = I + Y_+ T_+ Y_+^H,$$

donde $Y_+ = [Y, v] \in \mathbb{C}^{m \times (j+1)}$ y

$$T_+ = \begin{bmatrix} T & z \\ 0 & \rho \end{bmatrix},$$

con $\rho = -2$ y $z = -2TY^H v$.

Demostración. Notar que

$$\begin{aligned} I + Y_+ T_+ Y_+^H &= I + [Y, v] \begin{bmatrix} T & z \\ 0 & \rho \end{bmatrix} \begin{bmatrix} Y^H \\ v^H \end{bmatrix} \\ &= I + [Y, v] \begin{bmatrix} TY^H + zv^H \\ \rho v^H \end{bmatrix} \\ &= I + YTY^H + Yzv^H + \rho vv^H. \end{aligned}$$

Entonces, esto lleva a

$$Q_+ = QP = \left(I + YTY^H\right)\left(I - 2vv^H\right) = I + YTY^H + 2YTY^H vv^H - 2vv^H,$$

obteniendo la igualdad al definir los valores $\rho = -2$ y $z = -2YT^H v$. □

Algo importante que se desprende de esta representación es que el espacio de almacenamiento requerido es de $\frac{nk}{2}$ elementos de doble precisión compleja, dependiendo ahora de las columnas de la matriz y siendo la mitad del almacenamiento de la representación WY original en el peor de los casos, lo que convierte esta representación en la más eficiente en términos de almacenamiento y, finalmente, en rendimiento. En [58], Elmroth y Gustavson realizan un estudio exhaustivo de las *flops* de este algoritmo y fueron los responsables de la última implementación presente en `LAPACK`, que utiliza una estrategia recursiva para conseguir la mejor performance actual para la descomposición QR.

La implementación utilizada en esta tesis para la representación WY compacta sigue los lineamientos de la función `ZGEQRT2` de `LAPACK` [1].

Con las mejoras en rendimiento que ofrece la representación WY compacta, es posible utilizarla para resolver el problema de la actualización de la descomposición QR de la matriz

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

Complejidad y Estabilidad Numérica

Al completar el problema de actualización de la descomposición QR por bloques de M, se incurre en una cantidad de operaciones igual a $\frac{4}{3}n^3 + bn_A(\frac{n_A}{2} + n_D)$, donde n_A y n_D son las mayores dimensiones de A y D, respectivamente. Eligiendo b pequeño, la cantidad final de operaciones es muy cercana a la de la descomposición original de la matriz M ($\frac{4}{3}n^3$).

Si se utiliza el coeficiente $\rho = \frac{\|Q\|\|R\|}{\|A\|}$ es sencillo ver que la estabilidad numérica de este algoritmo es mejor que la de la descomposición LU, ya que la norma de una matriz Q ortogonal es siempre igual a 1, ya que cada paso de actualización sigue utilizando matrices ortogonales. Wilkinson demuestra que el error dado por la actualización de una matriz mediante transformaciones de Householder está acotado por una función $\phi(m, n)$ que es *lentamente creciente* con respecto a las dimensiones del problema [59], garantizando que la estabilidad numérica de QR es superior a la de LU.

[1] El código original está disponible en `https://www.netlib.org/lapack/explore-html/d3/d01/group__complex16_g_ecomputational_gaf3cc243f2912f6bf569c4ec61e071413.html`

3.2.3. Continuación de RLLS

Debido a la potencial inestabilidad en la que se incurre al utilizar la técnica de pivoteo incremental durante la actualización de la descomposición LU, incluso si la cantidad de operaciones es menor, se decidió utilizar la descomposición QR para resolver el problema de pocos cuerpos cuántico, ya que la estabilidad numérica del algoritmo tiene una mayor importancia, sobre todo cuando es muy probable que la solución satisfactoria del problema requiera múltiples actualizaciones (más de 2 niveles en RLLS).

Continuando con el proceso descripto en el Paso 3.2 del Algoritmo RLLS (Figura 3.1), se revisará el algoritmo para reutilizar una factorización ya conocida de $A^{(s-1)} = A_{00}^{(s)} = (Q\backslash R)^{s-1}$ en la descomposición de la matriz de coeficientes y la solución del sistema lineal correspondiente que aparece en el Paso 5.

A partir del comienzo del nivel s, asumiendo que ya se calculó la factorización $A^{(s-1)} = Q^{(s-1)}\backslash R^{(s-1)}$ y revisando la estructura en bloques del sistema completo, se puede considerar la siguiente partición del sistema lineal:

$$\left[\begin{array}{c|c} A^{(s-1)} & B^{(s-1)} \\ \hline C^{(s-1)} & D^{(s-1)} \end{array}\right] \left[\begin{array}{c} b_0^{(s)} \\ \hline b_1^{(s)} \end{array}\right] = \left[\begin{array}{c|c} Q\backslash R^{(s-1)} & B^{(s-1)} \\ \hline C^{(s-1)} & D^{(s-1)} \end{array}\right] \left[\begin{array}{c} b_0^{(s)} \\ \hline b_1^{(s)} \end{array}\right] =$$

$$= \left[\begin{array}{cccc|c} Q_{00}\backslash R_{00} & R_{01} & \dots & R_{0,s-1} & B_0 \\ Q_{01} & Q_{11}\backslash R_{11} & \dots & R_{1,s-1} & B_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ Q_{s-1,0} & & \dots & Q\backslash R_{s-1,s-1} & B_{s-1} \\ \hline C_0 & C_1 & \dots & C_{s-1} & D \end{array}\right] \left[\begin{array}{c} c_0 \\ c_1 \\ \vdots \\ c_{s-1} \\ \hline e \end{array}\right], \tag{3.10}$$

donde los bloques en la diagonal $R_{k,k}$, de dimensión $m_k \times m_k$, $k = 1, 2, \dots, s-1$, son todos triangulares superiores, $Q^{(0)} = Q_{0,0}$ denota a la matriz ortogonal que reduce $A^{(0)}$ a la forma triangular superior $R^{(0)}$ (por simplicidad, se puede asumir que n_{s-1}, la dimensión de fila/columna de $A^{(s-1)}$ y $R^{(s-1)}$, es un número entero múltiplo de t_d).

El procedimiento *consciente de la estructura* (structure-aware) para actualizar esta factorización matricial, el cual se denominará UMF por sus siglas en inglés (updating matrix factorization), se puede ilustrar en la Figura 3.2. Luego de completar el proceso, la secuencia de transformaciones definida por $Q_1, Q_2, \dots, Q_{t_n-1}$ habrá reducido la matriz de coeficientes en la Ec. (3.10)

$$\left[\begin{array}{c|c} Q^T A_{00}^{(s)} & A_{01}^{(s)} \\ \hline A_{10}^{(s)} & A_{11}^{(s)} \end{array}\right] = \left[\begin{array}{c|c} R^{(s-1)} & B \\ \hline C & D \end{array}\right] \tag{3.11}$$

a una forma triangular superior $R^{(s)}$ por una secuencia de transformaciones ortogonales, que son representadas por $Q^{(s)}$. Por esta razón, aplicando la misma secuencia de transformaciones al vector $b^{(s)}$, se puede utilizar el método de sustitución hacia atrás para resolver $R^{(s)}x^{(s)} = ((Q^{(s)})^T b^{(s)})$ para $x^{(s)}$.

La integración del proceso UMF para descomponer matrices densas en el algoritmo RLLS produce una variante *lazy* de la factorización QR donde, en cada paso de la descomposición (nivel del algoritmo RLLS), sólo se actualiza los elementos de la s-va columna y la s-va fila de la matriz por encima y a la izquierda del elemento (s, s), respectivamente. Con el mejor rendimiento en mente, esto puede formularse como un algoritmo en bloques.

Es importante remarcar algunos conceptos en el Algoritmo 3.2:

1.	Aplicar transformaciones previas a $B^{(s-1)}$: **para** $j = 0, 1, \ldots, s-1$	Costo (flops):
1.1	$B_j := Q_{j,j}^T B_j$	$2m_j^3$
	para $i = j+1, j+2, \ldots, s-1$	
1.2	$\begin{bmatrix} B_j \\ \hline B_i \end{bmatrix} := Q_{i,j}^T \begin{bmatrix} B_j \\ \hline B_i \end{bmatrix}$,	$4((m_j + m_i)m_j^2 - m_j^3/2)$
	fin bucle	
	fin bucle	
2.	Factorizar el nuevo bloque fila en $[C \mid D]$: **para** $j = 0, 1, \ldots, s-1$	
2.1	Calcular la desc. QR $\begin{bmatrix} R_{j,j} \\ \hline C_j \end{bmatrix} = Q_{s,j} \begin{bmatrix} \bar{R}_{s,j} \\ \hline 0 \end{bmatrix}$ y fijar $R_{j,j} := \bar{R}_{s,j}$.	$2(m_s m_j^2 + 4m_j^3/3)$
	para $p = j+1, j+2, \ldots, s-1$	
2.2	$\begin{bmatrix} R_{j,p} \\ \hline C_p \end{bmatrix} := Q_{s,j}^T \begin{bmatrix} R_{j,p} \\ \hline C_p \end{bmatrix}$.	$4((m_j + m_s)m_j^2 - m_j^3/2)$
	end for	
2.3	$\begin{bmatrix} B_j \\ \hline D \end{bmatrix} := Q_{s,j}^T \begin{bmatrix} B_j \\ \hline D \end{bmatrix}$.	$4((m_j + m_s)m_j^2 - m_j^3/2)$
	fin bucle	
3.	Calcular la fact. QR $E = Q_{s,s}\bar{R}_{s,s}$, y fijar $E := \bar{R}_{s,s}$.	$4m_s^3/3$

FIGURA 3.2: Procedimiento consciente de la estructura UMF (aplicación en el paso s del algoritmo RLLS). Aquí, $Q_{00} = Q^{(0)}$ denota el factor ortogonal resultante de la descomposición inicial de $A^{(0)}$ calculada como parte del Paso 2 del algoritmo RLLS.

- La factorización del bloque C es una operación independiente con un costo de $C_{\mathsf{U}} = 4((t_d + m_s)t_d^2 - t_d^3/2)$ flops. Se referirá a cada una de ellas como $\mathsf{U}_{k,j}$. De manera análoga, la aplicación de la submatriz constituida por los bloques C_k/E, al igual que los subvectores c_k/e, son operaciones independientes. La primera presenta un costo de $C_{\mathsf{C}} = 4((t_d+m_s)m_s^2 - m_s^3/2)$ flops mientras que la segunda requiere de una cantidad despreciable de operaciones. Se referirá a estas últimas dos aplicaciones como C_{k,t_n} y c_k, respectivamente.

- La última factorización de la matriz E tiene un costo de $C_{\mathsf{E}} = 4m_s^3/3$ flops y puede denotarse como E_{t_n}.

- Elegir t_d relativamente pequeño en relación a la dimensión del sistema y explotando la estructura triangular de $R^{(s-1)}$, es posible obtener una reducción considerable en los costos con respecto al enfoque simple; ver la columna de costos en la Figura 3.2.

Concretamente, teniendo en cuenta el recuento de operaciones de punto flotante de las operaciones enunciadas durante la formulación del procedimiento UMF en la Figura 3.2, el costo es

$$\sum_{k=0}^{t_n-1} \left(C_{\mathsf{F}} + C_{\mathsf{C}} + \sum_{j=k+1}^{t_n-1} C_{\mathsf{U}} \right) + C_{\mathsf{E}} \text{ flops.} \tag{3.12}$$

Más aún, debido al uso de transformaciones ortogonales, el procedimiento consciente de la estructura es numéricamente estable [38].

A partir de todo lo exhibido en este Capítulo, es posible comenzar con la implementación del algoritmo y evaluar su desempeño, en términos de uso de recursos computacionales y correctitud de la solución de los sistemas lineales.

Capítulo 4

Implementación, Resultados Numéricos y Aplicación a la Física de Colisiones

4.1. QuickSched

El primer experimento realizado para el trabajo de esta tesis consistió en la recreación del procedimiento de la descomposición LU por bloques de la Apartado 2.5 y de manera similar a la descomposición QR de [47], utilizando la librería `QuickSched`. Construir este ejemplo requiere definir 5 tareas diferentes en la implementación:

- `Build`: Construcción de bloque al leer de disco.
- `SolveLU`: Resolución LU en bloque diagonal.
- `SolveLeft`: Actualización a derecha de bloque diagonal (Sistema lineal a izquierda).
- `SolveRight`: Actualización debajo de bloque diagonal (Sistema lineal a derecha).
- `Update`: Actualización en resto de la matriz.

El objetivo de este experimento fue el de comparar la performance de la descomposición LU de una matriz en bloques mediante dos diferentes procedimientos, con múltiples hilos:

1. Descomposición LU por bloques utilizando `QuickSched`.
2. Descomposición LU de `LAPACK`.

La única tarea común a ambos procedimientos es la construcción de la matriz por bloques: `Build`.

4.1.1. Detalles de Implementación y Hardware

Como se explicó al introducir la librería, uno de sus requerimientos más importantes es la creación explícita del DAG que debe seguir la aplicación para resolver un problema. En la Figura 4.1 se muestra un ejemplo en código para agregar la tarea `SolveLU` de cada nivel de la descomposición LU.

Se utilizaron las implementaciones de `LAPACK` disponibles en la librería `OpenBLAS` para las operaciones matriciales.

Se utiliza como ejemplo una matriz de tamaño 36864×36864, separada en 16×16 y 9×9 bloques cuadrados de igual tamaño. Los programas creados fueron ejecutados para 1, 4, 6 y 12 hilos en la siguiente configuración de hardware:

CPU	Intel(R) Xeon(R) CPU E5-2620
Frecuencia	2.40 GHz
Cores/Hilos	12
Memoria RAM	128 GiB

```
for (i = 0; i < m && i < n; i++){
  /* Tarea diagonal */
  data[0] = i;
  data[1] = i;
  data[2] = i;
  tid_new = qsched_addtask(&s, task_blockLU, task_flag_none, data,
                           sizeof(int) * 3, 3);
  qsched_addlock(&s, tid_new, rid[i * m + i]);
  if (tid[i * m + i] != -1) qsched_addunlock(&s, tid[i * m + i], tid_new);
  tid[i * m + i] = tid_new;
```

FIGURA 4.1: Ejemplo de agregado de tareas al DAG

4.1.2. Resultados y Discusión

La Figura 4.2 muestra los tiempos de corrida de los diferentes experimentos numéricos realizados, para el procedimiento 1 (QuickSched 9x9 y 16x16) y el procedimiento 2 (Matriz entera 9x9 y 16x16), para distintas cantidades de hilos de cómputo, lo que permite comparar la performance del algoritmo de descomposición LU para la matriz.

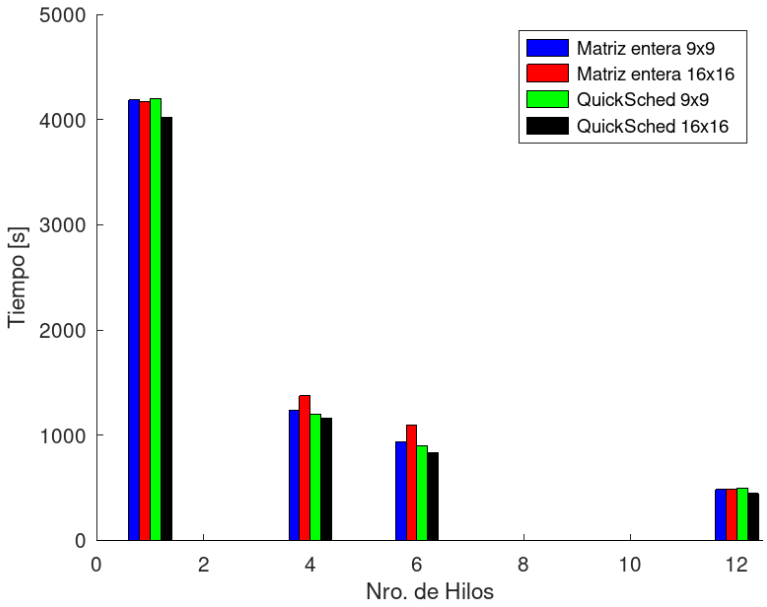

FIGURA 4.2: Cantidad de Hilos vs. Tiempo de Corrida (escala logarítmica) en la descomposición LU.

En la Figura 4.3 se muestra un gráfico de torta con el tiempo total por tarea para la corrida de 16×16 bloques.

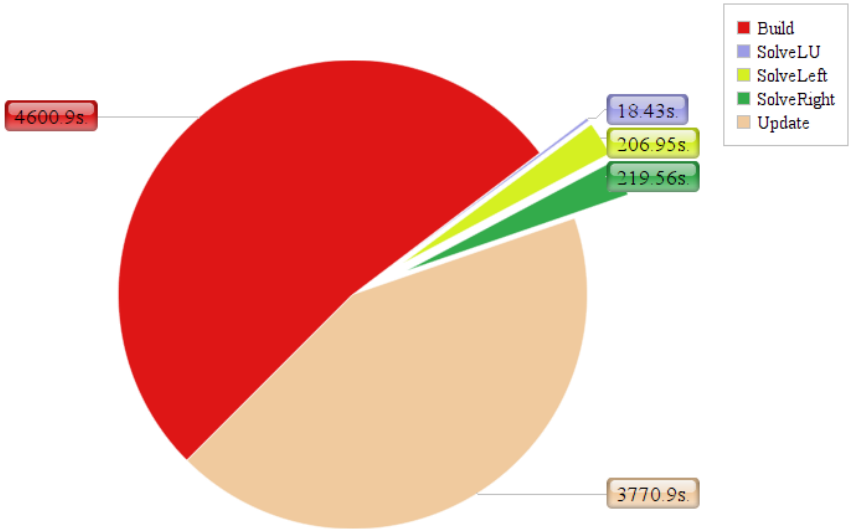

FIGURA 4.3: Diagrama de Tiempo por Tarea en la descomposición LU con QuickSched.

Si bien las mejoras en performance aplicando `QuickSched` no parecen ser sustanciales, la mayoría del tiempo en estos algoritmos estaba ocupado por las tareas `Build`, que tomaban el mismo tiempo en ambos experimentos. Eliminando esos tiempos, la mejora en performance relativa es mejor, teniendo en cuenta que la tarea `Update` también se desarrollaba de manera poco optimizada, sin aprovechar las capacidades de los procesadores disponibles (vectorización de operaciones). Estos resultados son buenos si se considera que se está compitiendo con una de las librerías más optimizada para el álgebra lineal numérica.

A pesar de que estos resultados son aceptables y pueden mejorar a medida que los tamaños de matrices sean cada vez más grandes, existen múltiples razones para no continuar utilizando `QuickSched` en este trabajo:

- el trabajo extra que supone crear explícitamente el DAG de ejecución de tareas, el cual se vuelve mucho más complejo al momento de implementar el algoritmo RLLS;
- la susceptibilidad a errores humanos en la programación del DAG, algo que puede disminuirse si se depende de otras librerías que realizan la planificación de tareas a partir del código, como `OpenMP` y `OmpSs`;
- una de las mayores fortalezas de `QuickSched` reside en la capacidad de generar condiciones de bloqueos para la no ejecución de algunas tareas, pero la misma no resulta de utilidad para implementar RLLS.

4.2. Herramientas Paralelas para la Solución de Sistemas Latentes

La solución de sistemas lineales densos convencionales (por ejemplo, los *no latentes*), sobre procesadores multinúcleo de próposito general, puede desarrollarse usando rutinas computacionales disponibles en librerías de álgebra lineal numérica como `MKL`, `OpenBLAS` o `FLAME`. De forma concreta, las rutinas paralelas alojadas en estas librerías pueden también ser aplicadas a la solución del sistema lineal inicial en la Ec. (3.5).

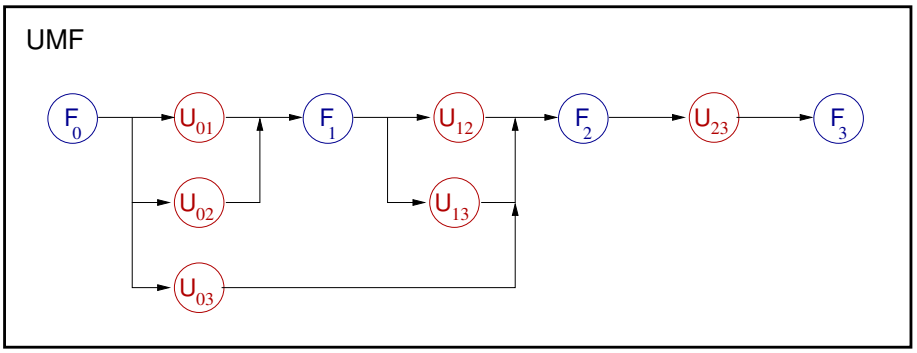

FIGURA 4.4: GDT (simplificado) para el procedimiento consciente de la estructura UMF aplicado a la matriz $A^{(s)}$ en el nivel $s = 4$.

Para resolver la recurrencia definida en los sistemas latentes, en el Paso 3.2 de la Figura 3.1, también es posible utilizar esas rutinas con el objetivo de calcular en paralelo los kernels que componen al procedimiento UMF consciente de la estructura y su solución subsecuente (ver Figura 3.2). De cualquier manera, este enfoque está restringido en la cantidad de paralelismo que puede ser explotado, lo que lleva a una solución que es difícil de escalar.

A continuación, se presenta una estrategia alternativa que expone un mayor paralelismo de tareas en la recurrencia de los sistemas lineales latentes, mejorando la escalabilidad final del proceso, con un análisis detallado de las capacidades del mismo. Luego, se muestra el modelo de programación de `OmpSs` que subyace a la solución paralela del problema.

4.2.1. Solución en paralelismo de tareas para sistemas lineales latentes

Considerar el procedimiento consciente de la estructura matricial UMF en la Figura 3.2 aplicado al nivel $s = 4$. Se puede denotar cada una de las factorizaciones en el Paso 2.1 de la figura como F_j, $j = 0, \ldots, 3$; y a la aplicación de las transformaciones ortogonales resumidas por $Q_{s,j}$ a cada una de las submatrices

$$\left[\frac{R_{j,p}}{C_p}\right], \quad p = j+1, j+2, \ldots, 3, \tag{4.1}$$

en el Paso 2.2, como $\mathsf{U}_{j,p}$, para $p = j+1, j+2, \ldots, 3$. Más aún, por simplicidad, para la discusión siguiente se puede pasar de la aplicación de las transformaciones en el Paso 2.3 a

$$\left[\frac{B_j}{D}\right] \tag{4.2}$$

al igual que se puede evitar pensar en las operaciones en el Paso 1.

La Figura 4.4 representa las dependencias de datos que surgen en la actualización de la descomposición matricial en la como un grafo de dependencias de tareas (GDT), donde los vértices identifican a las operaciones (tareas o núcleos de cómputo) y las aristas especifican las dependencias.

Este gráfico reporta la concurrencia disponible en este paso del algoritmo como aquel presente solamente en las actualizaciones $\mathsf{U}_{j,p}$ para cada valor independiente de p. Se puede exponer paralelismo adicional dividiendo en bloques de columnas/filas más pequeños de ancho/altura $t_d < m_j$ al bloque $C^{(s-1)}/B^{(s-1)}$ en la Ec. (3.10) (Notar

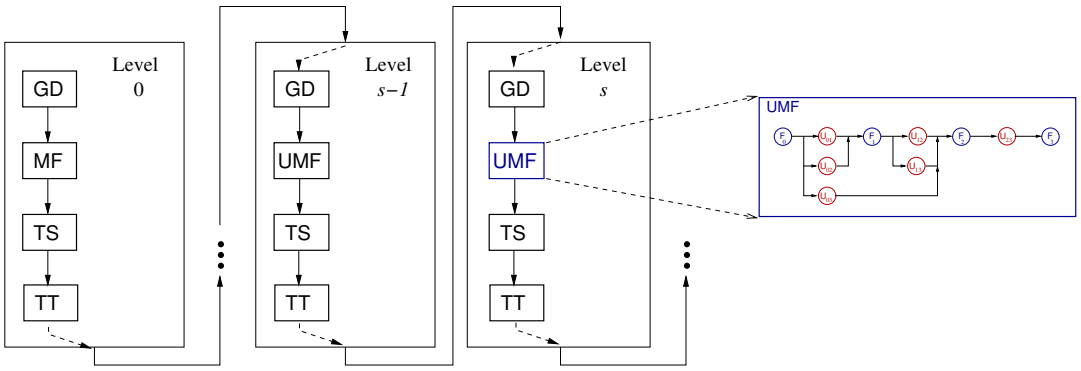

FIGURA 4.5: GDT (Simplificado) para el algoritmo RLLS.

que esto introduce una partición de $R^{(s-1)}$ que tiene bloques diagonales de dimensión $t_d \times t_d$.)

Ahora, la cantidad de estos bloques depende del cociente entre n_{s-1} y t_d (es decir, $t_n = n_{s-1}/t_d$) y, por lo tanto, haciendo decrecer t_d se puede exponer un paralelismo de tareas mayor. De cualquier manera, a medida que t_d se hace más pequeño, la actualización de estos bloques se convierte en una tarea que requiere mucha memoria, reduciendo dramáticamente el rendimiento.

Una mejor oportunidad de exponer el paralelismo de tareas surge considerando la solución de la recurrencia completa de los sistemas lineales latentes. En particular, la Figura 4.5 muestra los niveles y pasos que aparecen en el proceso de solución (ver la Figura 3.1), usando la siguiente notación:

- GD: Generación de Datos (Pasos 1 y 3.1);
- MF: Factorización Matricial (Paso 2, sólo en el nivel 0) y UMF para la versión de actualización (Paso 3.2);
- TS: Aplicación de las transformaciones ortogonales al lado derecho y solución del sistema triangular superior subsecuente (siguiendo MF en el nivel 0 o UMF en los niveles 1,2,...); y
- TT: test de terminación (criterio de parada).

Para la tarea GD es posible particionar los datos del problema en múltiples bloques independientes, logrando una ejecución trivialmente paralelizable de este paso. En contraste con GD, TS necesita operaciones con un costo menor y ofrecen una baja performance en términos de paralelismo.

Desafortunadamente, esta organización del algoritmo RLLS presenta dependencias de datos entre cada par de pasos consecutivos y una dependencia de control (para el criterio de parada) entre cada par consecutivo de niveles, que introducen cada uno un punto de sincronización:

$$\overbrace{\mathsf{GD} \to \mathsf{MF} \to \mathsf{TS} \to \mathsf{TT}}^{\text{Nivel } 0} \to \cdots \to \overbrace{\mathsf{GD} \to \mathsf{UMF} \to \mathsf{TS} \to \mathsf{TT}}^{\text{Nivel } s-1} \to \overbrace{\mathsf{GD} \to \mathsf{UMF} \to \mathsf{TS} \to \mathsf{TT}}^{\text{Nivel } s} \to \cdots$$

El enfoque propuesto aquí es fusionar GD con MF (en el nivel 0) o UMF (en los niveles 1, ...), para que la primera tarea en acceder un bloque de $C^{(s-1)}$ durante la actualización de la factorización matricial sea también responsable de generar los datos para ese bloque. Para el grafo simplificado en la Figura 4.4, esto significa que la tarea F_0 genera los datos para C_0; y las tareas $\mathsf{U}_{0,p}$, $p = 1, 2, 3$, generan los de C_p.

Otro paso aún más importante es romper el punto de sincronización entre niveles consecutivos, permitiendo que el nivel s comience su ejecución incluso antes de que el nivel $s-1$ haya sido completado. El propósito de esto es aprovechar los hilos de cómputo que no estén trabajando, realizando una ejecución especulativa de las tareas pertenecientes a niveles futuros, que pueden o no ser necesarias de acuerdo al test del criterio de parada.

Con este enfoque, el test en el nivel s se convierte en una señal asíncrona que puede parar la ejecución de tareas en niveles futuros $s+1, s+2, \ldots$. Para favorecer una solución temprana al problema, es importante aún priorizar las ejecuciones de las tareas en los niveles más bajos.

A modo de cierre de esta propuesta, es importante remarcar que una solución basada en librerías existentes para paralelismo de tareas en álgebra lineal numérica densa, como `libflame` o PLASMA, no es práctica debido a dos puntos:

1. la necesidad de explotar la estructura especial de $A^{(s)}$ para poder reducir el costo de la actualización de la factorización;
2. la necesidad de un manejo explícito del proceso de solución para exponer un mayor paralelismo de tareas.

4.2.2. Bloques Computacionales Básicos

Los pasos MF y UMF están subdivididos en pequeñas tareas, denominadas como Bloques Computacionales Básicos, a partir de la factorización QR *lazy*:

- GEQRF: Cálculo de la factorización QR de un bloque en la diagonal;
- APQTR: Aplicación de la transpuesta de un bloque diagonal ortogonal a un bloque a su derecha;
- UPQRD: Actualización de la transformación ortogonal en un bloque diagonal por medio de un bloque debajo de la diagonal (siguiendo el esquema de Householder - WY); y
- UPDSE: Actualización de un bloque de la matriz.

El procedimiento TS puede ser subdividido en 3 tareas, que describen los pasos usuales para resolver un sistema triangular a partir de la factorización QR:

- APQTR: Esta tarea es la misma que la de UMF, aplicada al vector del lado derecho del sistema lineal;
- GEMV: Actualización del sistema triangular superior que resulta en cada bloque de la matriz; y
- TRSV: Solución del sistema triangular que queda en cada bloque de la matriz para obtener la solución final.

El problema de sincronización existente entre niveles de MF o UMF está dado por la tarea TS: el sistema lineal correspondiente al nivel $s-1$ necesita ser resuelto antes de actualizar la factorización de la matriz en el nivel s para los bloques $A^{(s-1)}$ y $B^{(s-1)}$. Aunque esta porción de la matriz no puede ser modificada mientras se resuelve el sistema para el nivel $s-1$, la factorización matricial puede ser actualizada en los bloques de $C^{(s-1)}$ porque no afecta a $A^{(s-1)}$ en ninguna manera. Por eso también

es posible fusionar las tareas de TS y UMF relacionadas con la porción $C^{(s-1)}$ de la matriz en el nivel s del procedimiento (tareas APQTR y UPDSE).

A modo informativo, en la Figura 4.6 se muestran los tiempos de corrida de las tareas relacionadas con la factorización QR por bloques de la matriz, para distintos tamaños de bloques. Es importante notar que la mayor parte del esfuerzo computacional es hecha por GEQRF (Factorización de un bloque diagonal) y UPQRD (actualización de un bloque diagonal), por requerir operaciones BLAS de nivel 3.

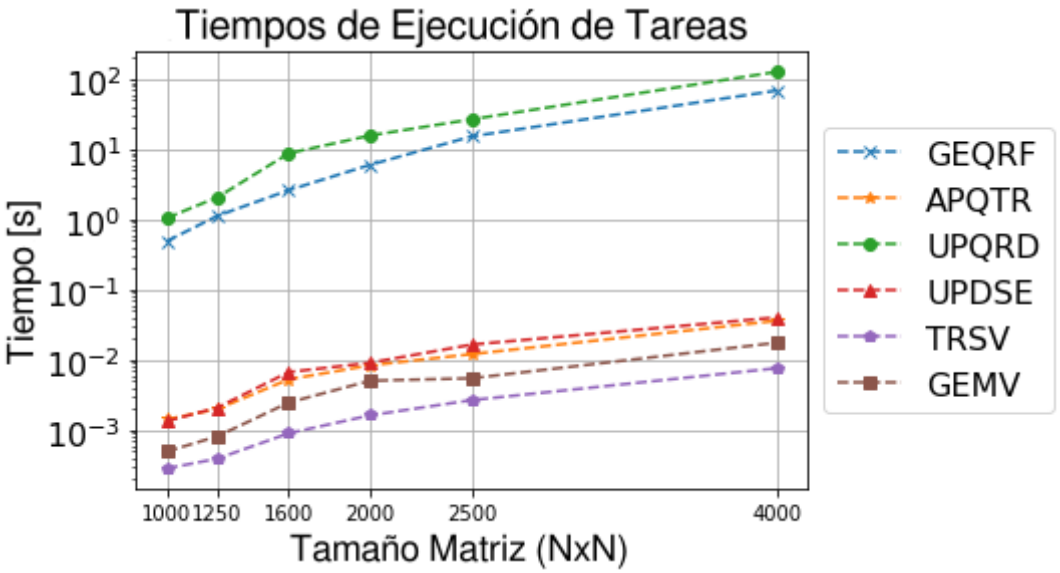

FIGURA 4.6: Análisis de tiempos de ejecución de tareas computacionales. Las descripciones de las mismas se encuentran en la Subsección 4.2.2.

4.2.3. Detalles de implementación en `OmpSs` del algoritmo RLLS

En tiempo de ejecución, el sistema de `OmpSs` detecta las dependencias de datos entre tareas, con la ayuda de la persona que programa, via directivas de compilador como las de `OpenMP` (pragmas), anotadas con cláusulas que indican la direccionalidad (entrada, salida o entrada/salida) de los operandos de las tareas. `OmpSs` luego genera el grafo de tareas durante su ejecución, el cual es utilizado para administrar las tareas entre los hilos, explotando el paralelismo al nivel de tareas inherente, al mismo tiempo que se satisfacen las dependencias embebidas en el grafo.

Ahora puede discutirse el aprovechamiento del modelo de programación paralela por tareas definido por `OmpSs` en la solución de los sistemas lineales latentes que forman parte de los problemas de Coulomb de pocos cuerpos.

El solver paralelo por tareas depende de una implementación secuencial del algoritmo RLLS y el procedimiento consciente de la estructura UMF en el lenguaje `C`, con rutinas para las tareas computacionales con interfaces limpias.

A partir de ese punto inicial, se agregan pragmas de `OmpSs` (`#pragma omp task`) para identificar las tareas, cláusulas de direccionalidad (`in`, `out` and `inout`) para indicar el carácter de los argumentos de las rutinas matriciales y posiblemente prioridades para guiar el orden de ejecución de las tareas. Utilizando la información que provee el código, el scheduler genera el grafo de dependencias y puede proceder a ejecutar el código en el orden correcto. Se muestra un ejemplo del uso de estas cláusulas en una tarea en la Figura 4.7.

```
// Factorize block on the diagonal, level k
#pragma omp task depend(inout: Ah[nt*k+k])\
                 depend(out: Th[nt*k+k], Sh[nt*k+k])\
                 firstprivate(k, ldim_A, ldim_S, ibs, its, nt)
Compute_dense_QR_var31a_complex(ibs, its, its,
                                &Ah[ nt*k+k ], ldim_A,
                                &Th[ nt*k+k ],
                                &Sh[ nt*k+k ], ldim_S,
                                nt, k, stop);
```

FIGURA 4.7: Ejemplo del uso de pragmas en `OmpSs`/`OpenMP`.

Otro detalle de implementación importante es que, si bien el tamaño final del sistema lineal a resolver es desconocido, es necesario generar una estructura que permita manejar los bloques de la matriz de manera sencilla. Se optó por generar una matriz de punteros que alojarán la dirección del puntero que representa a cada bloque, que puede tener tamaño más grande del necesario (por ejemplo 50×50), pero cuyo tamaño en memoria es despreciable en relación al de cada bloque, ya que en cada elemento de la estructura sólo se aloja una dirección de memoria (las posibles locaciones en memoria de los bloques matriciales). Lo mismo debe hacerse para los bloques de cada transformación WY, que se pueden denotar con T y S, al igual que para el lado derecho b. El código que genera estas estructuras, conocidas como *matrices jerárquicas*, se presenta en la Figura 4.8.

```
/* Creation of hierarchical matrices. */
Ah = (double **) malloc( nt*nt*sizeof(double *) );
Th = (double **) malloc( nt*nt*sizeof(double *) );
Sh = (double **) malloc( nt*nt*sizeof(double *) );
bh = (double **) malloc( nt*nt*sizeof(double *) );
```

FIGURA 4.8: Creación de las matrices jerárquicas.

4.2.4. Análisis de Rendimiento

El objetivo de esta subsección es analizar la velocidad y uso de recursos del algoritmo de descomposición descripto en la subsección anterior, para exponer los beneficios de utilizar un modelo de programación con paralelización de tareas, como el que ofrece `OmpSs`, para calcular la factorización QR *lazy* al operar sobre matrices latentes.

Por esta razón, se diseñaron experimentos que permitan evaluar la escalabilidad del código de factorización desarrollado en `OmpSs`, en un escenario simple pero práctico. En particular, para la evaluación de la factorización QR *lazy* paralela por tareas, se generaron matrices cuadradas latentes, cuyas dimensiones se detallan en la Tabla 4.1.

Por simplicidad, en cada problema la dimensión de todos los niveles será la misma: $m = m_0 = m_1 = m_2 = \ldots$, con valores que crezcan en proporción con n. Los mismos están en la columna ts de la Tabla 4.1. En estas pruebas no habrá criterio de parada, si no que se realizará la factorización QR por bloques de la matriz completa. Esto implica que el número de niveles requerido para convergencia está fijo en 24 para todas las

Hilos	n	ts	bs
1	14400	600	100
2	20160	840	140
4	28800	1200	200
6	34560	1440	240
8	40320	1680	280
10	44640	1860	310
12	48960	2040	340

CUADRO 4.1: Dimensiones para el experimento de escalabilidad.

instancias de prueba. La evaluación es desarrollada en términos de GFLOPS (miles de millones de flops por segundo), usando el costo de la factorización QR estándar.

Todos los experimentos de esta subsección fueron desarrollados en aritmética y con datos de doble precisión, en un servidor equipado con el siguiente hardware:

CPU	2x Intel(R) Xeon(R) CPU E5645
Frecuencia	2.40 GHz
Cores	12
Memoria RAM	48 GiB

Los códigos fueron vinculados con kernels de BLAS de `Intel MKL composer_xe` 2011 y la versión 16.06 de `OmpSs`.

4.2.5. Escalabilidad de la Factorización QR Lazy

Las Figuras 4.9 y 4.10 muestran la tasa de GFLOPS obtenida por la versión en `OmpSs` de las rutinas para la factorización QR *lazy*, ejecutadas en el servidor Intel de 12 *cores* para dos configuraciones de experimentos diferentes, correspondiendo a pruebas de escalabilidad fuerte y débil, respectivamente.

Para la evaluación de la escalabilidad fuerte, el tamaño del problema se deja fijo en $n = 49{,}152$ (con $m = 2{,}048$), variando luego la cantidad de núcleos que realizan la factorización desde 1 a 12. En este escenario debería observarse un decrecimiento de los GFLOPS obtenidos a medida que el número de hilos se incrementa, reflejando una situación en la que la dimensión del problema ($n \times n$) que toma cada hilo se va reduciendo gradualmente, donde el escenario ideal sería tener el mismo valor sin importar la cantidad de hilos utilizada. La Figura 4.9 muestra que los GFLOPS por núcleo decrecen a medida que se utilizan más núcleos. De cualquier manera, el decrecimiento es pequeño y ocurre mayormente para ejecuciones de 4, 6 y 12 hilos. En general, la reducción de 1 hilo y 8 GFLOPS a 12 hilos y un poco menos de 7 GFLOPS es de alrededor del 15 %, lo que muestra una escalabilidad razonablemente fuerte para la factorización QR *lazy* paralela por tareas.

Para evaluar la escalabilidad débil de la implementación (Figura 4.10), se corre una instancia más pequeña del problema (n=14,400, m=600) en un único hilo, luego se incrementa gradualmente el número de hilos mientras se mantiene constante el cociente entre la dimensión del problema y la cantidad de hilos. Las dimensiones son muy similares a las que se enumeran en la Tabla 4.1, con n=14.400, 20.160, ..., 49.152 ejecutados usando 1, 2, ..., 12 hilos/cores. Para este experimento, se observa un incremento rápido de la tasa de GFLOPS para hasta 10 hilos, pero se ve un estancamiento para la ejecución del problema más grande con 12 hilos. Aunque

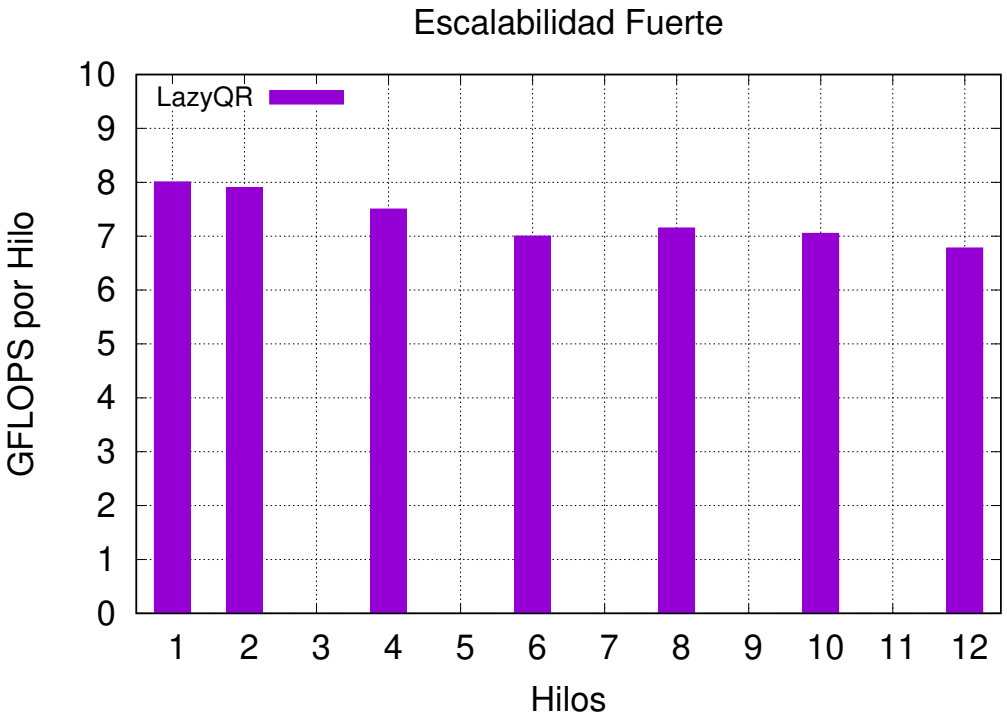

FIGURA 4.9: Escalabilidad Fuerte de la Factorización QR paralela por tareas usando `OmpSs` en el servidor equipado con Intel E5-5645.

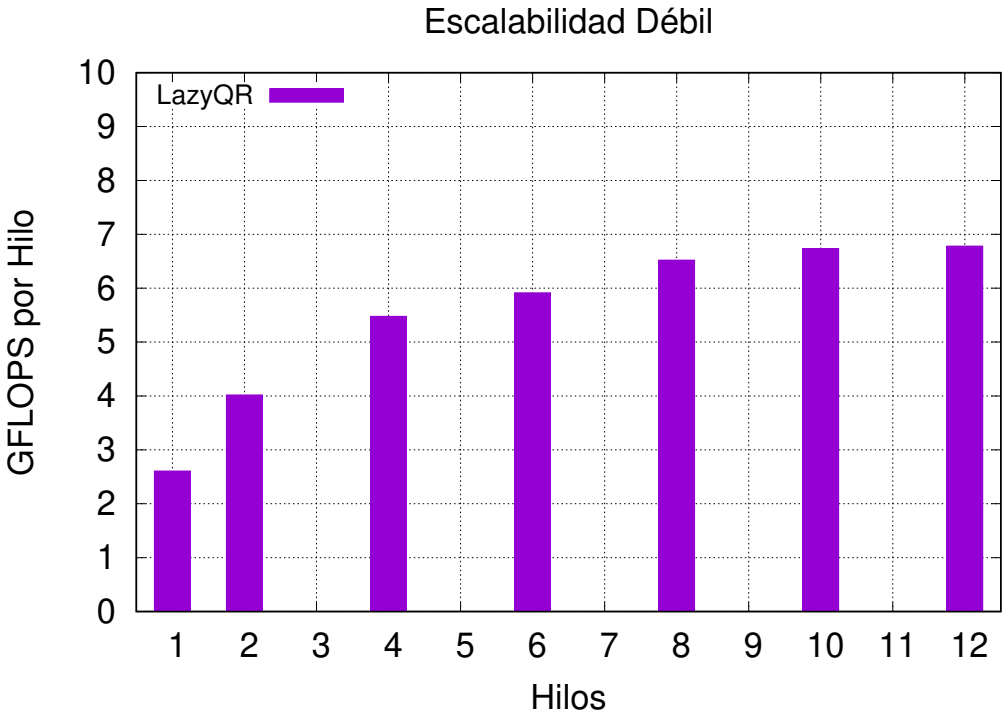

FIGURA 4.10: Escalabilidad Débil de la Factorización QR paralela por tareas usando `OmpSs` en el servidor equipado con Intel E5-5645.

puede ser un resultado poco alentador (una escalabilidad débil ideal resultaría en una correspondencia lineal creciente entre la cantidad de núcleos y la tasa de operaciones realizadas), una optimización dependiente del problema (e.g., ajustar los parámetros de bloques para uso interno en los bloques principales del procedimiento `UMF`) puede resolver este inconveniente.

4.3. OpenMP y Modelado de Prioridades

Como se describió en la Subsección 2.5.1, `OpenMP` es el estándar más utilizado para paralelismo de tareas. Sumado al hecho de que su instalación es relativamente simple, se decidió explorar el desarrollo de la herramienta y realizar los experimentos con este estándar.

Para ello se realizó un estudio de performance y comparar esta nueva solución con la implementada en `OmpSs`, sumando el estudio de las instrucciones de *prioridad*.

A continuación, se presentarán las diferentes soluciones programadas y se las compara, se mide el rendimiento de cada uno de los bloques computacionales del algoritmo RLLS y se compara el rendimiento de la solución con respecto al flujo normal de trabajo, que requiere generar la matriz por bloques y luego aplicar la factorización QR al final de la generación.

Aunque `OmpSs` y `OpenMP` son muy similares, el primero poseía un sistema de prioridades más simple y completo al momento de desarrollar estos experimentos. Este sistema permite guiar en cierta manera la ejecución del algoritmo, haciéndolo más rápido al determinar el orden en el cual las tareas deben ser realizadas.

Esto puede asignarse a cada tarea con la cláusula `priority(p)`, donde p puede ser una número o una función de los inputs de la tarea correspondiente. La capacidad de utilizar una función como valor de prioridad es muy útil en esta aplicación, porque permite representar la sincronización entre algunas de las tareas aplicadas en el algoritmo, por ejemplo para la ejecución de GD y TS en momentos similares. Esto se logra asignando una mayor prioridad a las tareas TS en el nivel $s-1$ que al valor de las tareas UMF en el nivel s.

Para comparar los modelos, se implementó el algoritmo *LatentQR* de tres maneras diferentes:

- Una implementación con `OpenMP`;
- Otra utilizando `OmpSs` sin prioridades;
- Una última utilizando cláusulas priority en `OmpSs`, con un valor de prioridad más alto a las tareas de TS que a las de UMF en cada nivel.

Se utilizó el siguiente esquema: se crea una matriz $40,000 \times 40,000$ subdividida en bloques de tamaños iguales, creando experimentos diferentes para las configuraciones de bloques 10×10, 16×16, 20×20, 25×25, 32×32 y 40×40. Los elementos de la matriz serán números flotantes aleatorios de precisión doble distribuidos uniformemente en el intervalo $[0, 1]$. Para acercarse a una configuración similar a la de algunos problemas de Coulomb de pocos cuerpos, donde el costo de generar la matriz crece a medida que se utilizan más momentos angulares (es decir, se avanza en los bloques fila/columna), se optó por agregar una pequeña latencia a la creación de los bloques aleatorios utilizando la función `usleep` en `C`. De cualquier manera, la cantidad de latencia agregada no supera 0.1 segundos al bloque de mayor latencia, para poder medir el rendimiento del nuevo enfoque en términos de las operaciones de la factorización. Los resultados se muestran en las Figuras 4.11 y 4.12, revelando los siguientes puntos:

- En general, las tres implementaciones paralelas por tareas ofrecen una buena eficiencia de cálculo, comparadas con los tiempos de ejecución previamente observados (para `WholeQR`).
- Comparando las tres versiones, es claro que `OmpSs` siempre muestra un mejor rendimiento cuando se utiliza el esquema de prioridades. Más aún, cuando se utilizan 6 hilos, `OmpSs` es ligeramente más lento que `OpenMP`, pero su comportamiento es opuesto a medida que se utilizan más hilos. Esto puede estar justificado por el hecho que `OmpSs` incurre en cierto costo extra al comienzo de la ejecución, cuando debe generar el cronograma de tareas de acuerdo a las prioridades, que va ganando más importancia a medida que el grado de concurrencia (cantidad de hilos) se incrementa.
- Por último, con respecto a la división de la matriz inicial por bloques, todas las implementaciones revelan que, utilizando 6 hilos, se logra la mejor eficiencia paralela cuando se particiona la matriz en bloques 16×16 o 20×20 y, cuando se utilizan 12 hilos, es mejor dividir la matriz original en bloques 20×20 o 25×25 blocks. En general, se puede concluir que, para un número de hilos entre 6 y 12, sería recomendable dividir la matriz $40,000 \times 40,000$ en bloques de tamaño 20×20, aunque para aplicaciones como el problema de Coulomb, es muy probable que sea imposible decidir esta división de antemano.

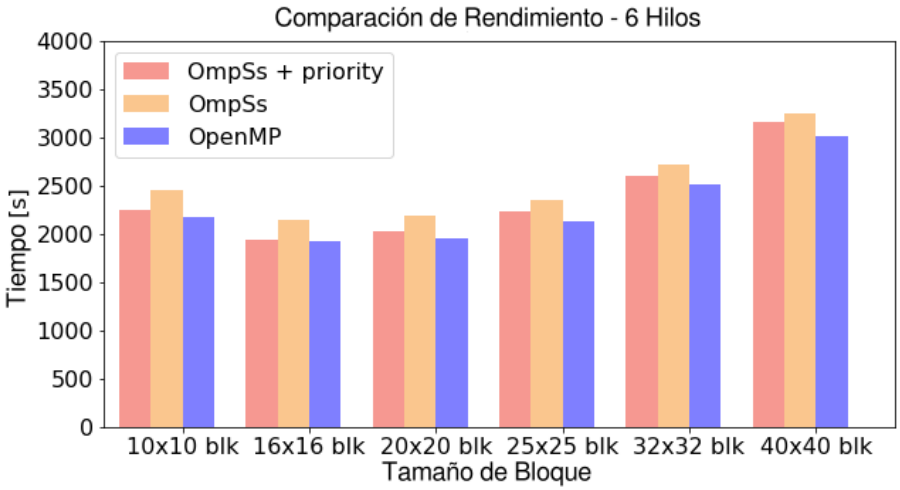

FIGURA 4.11: Tiempo de Ejecución por Tamaño de Bloque (6 Hilos).

Todos los experimentos en esta subsección fueron desarrollados en aritmética flotante de doble precisión, sobre un servidor con las siguientes especificaciones de hardware:

CPU	2x Intel(R) Xeon(R) CPU E5645
Frecuencia	2.40 GHz
Cores/Hilos	12
Memoria RAM	48 GiB

Los códigos fueron vinculados con kernels de BLAS de `Intel MKL composer_xe` 2011 y la versión 16.06 de `OmpSs`. La aplicación de `OpenMP` fue compilada utilizando `GCC 5.3.0`.

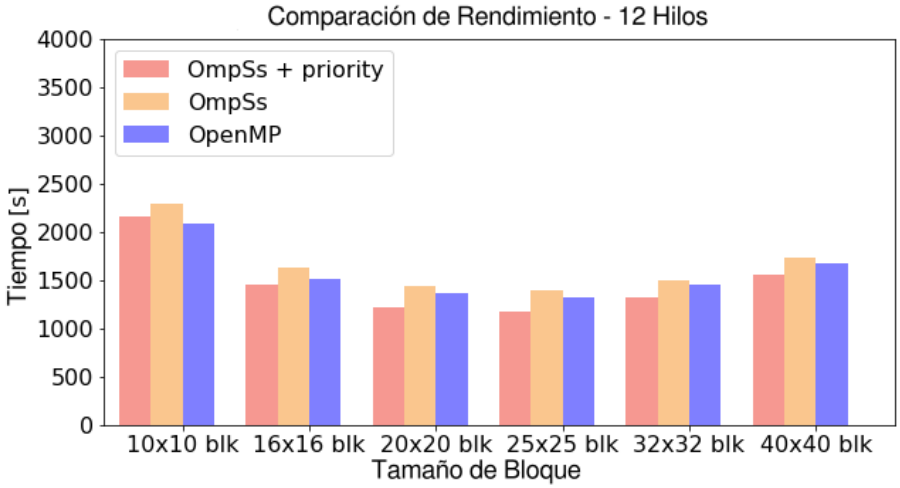

FIGURA 4.12: Tiempo de Ejecución por Tamaño de Bloque (12 Hilos)

4.3.1. Análisis de Rendimiento General y Conclusiones

A continuación se compara el rendimiento de la solución representante del algoritmo RLLS con el método más usado frecuentemente: generar la matriz completa por bloques y aplicar un método de factorización QR a la misma. Es importante notar, además, que el enfoque RLLS (denominado *LatentQR*) no sólo genera la descomposición QR; además, resuelve los sistemas lineales intermedios contra vectores b_s (tarea TS en el grafo de dependencias). Todo esto es comparado con la generación y factorización de la matriz completa (referenciada como *WholeQR*).

Incluso teniendo en cuenta que la cantidad de tareas de *LatentQR* es mayor, la aplicación paralela por tareas exhibe un rendimiento mucho mejor utilizando 6 y 12 hilos, como se muestra en la Figura 4.13 y la Figura 4.14.

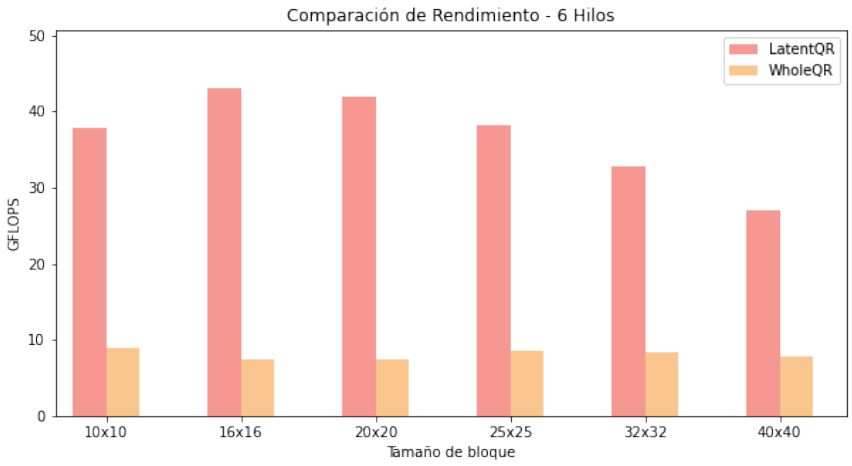

FIGURA 4.13: GFLOPS para Descomposición QR con 6 hilos para diferentes tamaños de bloque.

Esto permite asegurar que el enfoque RLLS realmente ofrece una mejora en rendimiento, que permite resolver las ecuaciones diferenciales relacionadas con simulaciones

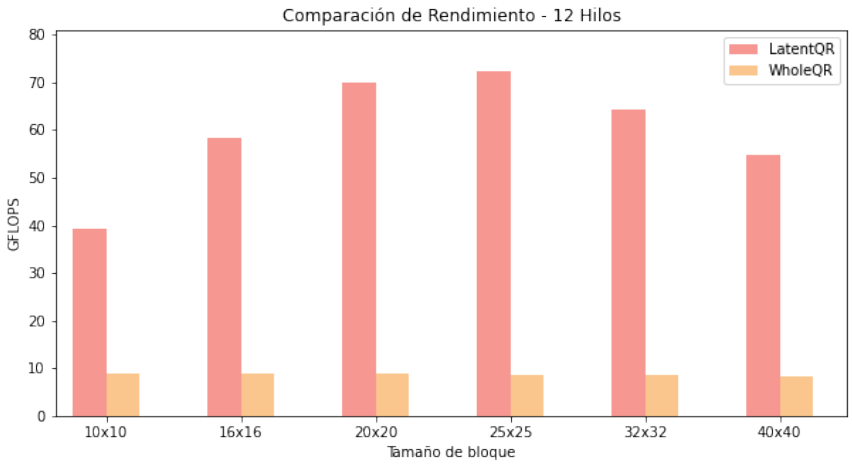

FIGURA 4.14: GFLOPS para Descomposición QR con 12 hilos para diferentes tamaños de bloque.

atómicas con mayor precisión y velocidad, permitiendo obtener resultados para problemas que no han podido ser atacados en el pasado.

Debido a que la solución programada en `OpenMP` tiene un rendimiento bueno incluso sin aplicar *prioridades*, su alta disponibilidad y simpleza de instalación con comandos simples (lo cual es de importancia al momento de reproducir experimentos en otras piezas de hardware) y el hecho de que las mejoras en prioridades ofrecidas por `OmpSs` serán agregadas en el futuro, se decidió desarrollar el resto de la solución teniendo en cuenta este estándar.

4.4. Simulaciones Atómicas

Esta sección se enfoca en experimentos numéricos relacionados con problemas de Coulomb de pocos cuerpos; muestra las primeras pruebas del algoritmo RLLS aplicado a simulaciones atómicas. En particular, se aplicó el algoritmo al problema de **Doble Fotoionización**, descripto en el Capítulo 1. En las próximas subsecciones, se mostrarán simulaciones numéricas en moléculas de Helio (He) y Agua (H_2O), junto con experimentos para medir el rendimiento de las implementaciones del algoritmo RLLS para obtener las secciones eficaces, realizando comparaciones con métodos utilizados anteriormente por otros autores.

4.4.1. Detalles de Implementación

Las simulaciones atómicas utilizan números complejos, por lo que fue necesario adaptar el código implementado para resolver el problema RLLS a aritmética compleja de doble precisión. Esto puede hacerse de manera directa reemplazando las instrucciones D (*double precision*) por Z (*double complex precision*), de acuerdo a la documentación de `LAPACK` [43], teniendo en cuenta que las transposiciones deben además conjugar los valores de la matriz, para poder realizar la factorización QR por bloques de manera correcta, de acuerdo al Teorema 2.

Para las siguientes simulaciones, la matriz fue generada a partir de los bloques calculados con la herramienta desarrollada en [60]. Esta implementación fue hecha en el lenguaje `Fortran` y da como resultado matrices complejas en formato binario, por

lo que fue necesario generar una rutina en `C` que pudiera leer este formato binario de manera correcta para realizar los cálculos (ambos lenguajes tienen formas distintas de generar salidas en binario). Este tipo de rutinas es conocida como *wrapper*.

4.4.2. Doble Fotoionización del Helio

La doble fotoionización del Helio (He) por absorción de un fotón es un problema de gran interés para la disciplina de las simulaciones atómicas, dado que es el proceso más simple donde puede ocurrir la emisión al continuo de dos electrones: es un problema de tres cuerpos puro sin perturbaciones antes y después de que el fotón sea absorbido, lo que lo convierte en la mejor prueba para estudiar una transición de tres cuerpos entre los estados ligado y continuo. Además, gracias a la existencia de múltiples mediciones experimentales realizadas en la literatura, este problema es de gran utilidad para evaluar la correctitud del método de simulación implementado. La Figura 4.15 muestra las secciones eficaces obtenidas aplicando el método RLLS para resolver el sistema lineal resultante. Puede observarse que la misma coincide con los resultados obtenidos mediante otros métodos en el pasado.

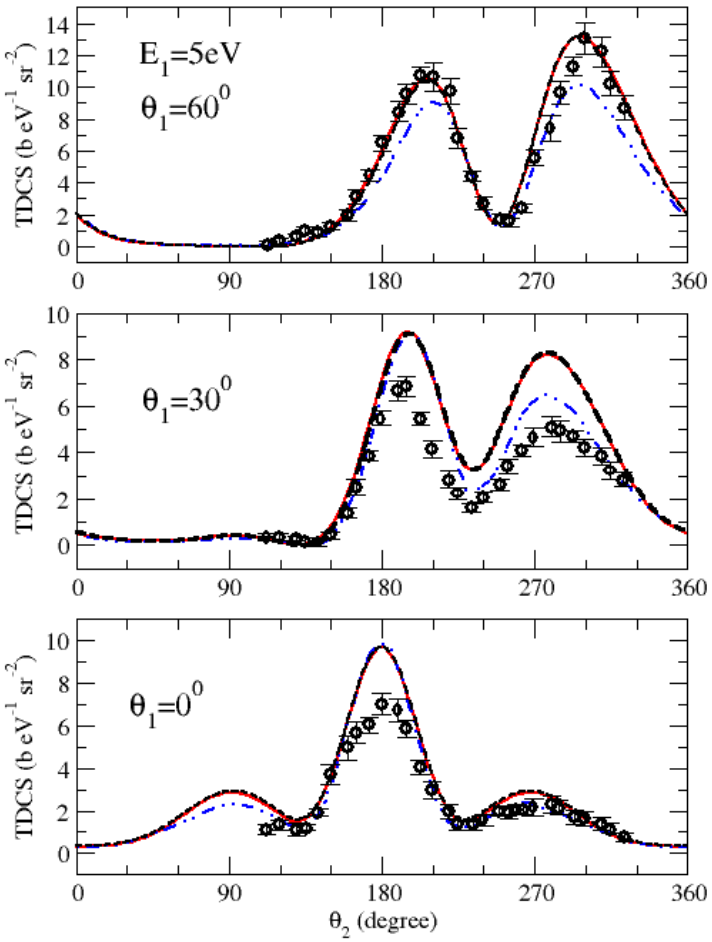

FIGURA 4.15: Secciones eficaces obtenidas de forma experimental, mediante el método de gradiente conjugado cuadrado (línea roja) y utilizando el algoritmo RLLS (línea negra cortada) [35]

La Figura 4.16 muestra el rendimiento (en forma de GFLOPS promedio por problema) que la aplicación de RLLS en `OpenMP` consigue al resolver la doble fotoionización del helio, comparado con el método utilizado en [61], el Gradiente Conjugado Cuadrado Precondicionado.

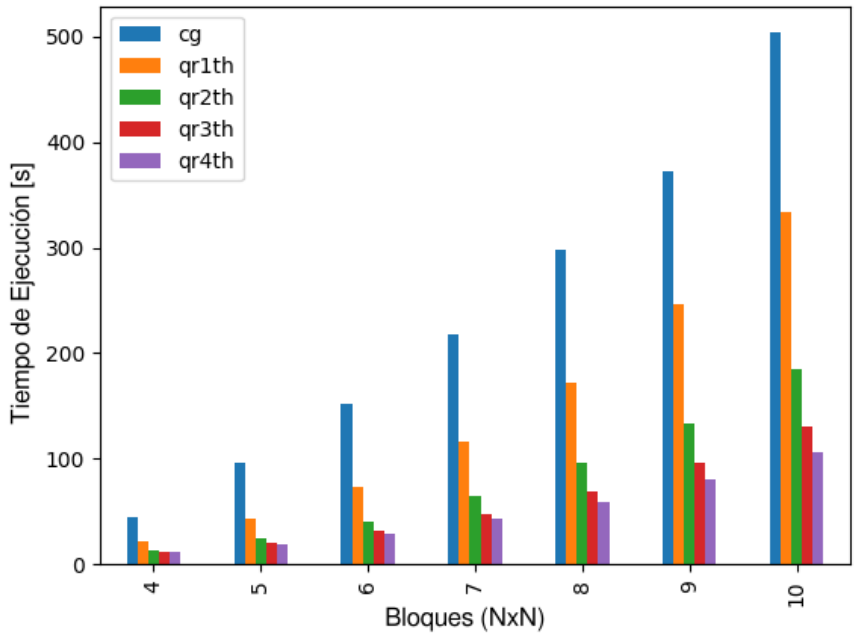

FIGURA 4.16: Comparación de tiempos de ejecución de algoritmos: Gradiente Conjugado (no paralelizado - cg) y la implementación del algoritmo RLLS, usando 1-4 hilos de cómputo (qr1-4th).

4.4.3. Doble Fotoionización del Agua

El sistema físico bajo consideración en esta subsección es el de la Doble Fotoionización (DPI) de la molécula de agua (H_2O) mediante la absorción de un único fotón, siguiendo los experimentos e investigaciones teóricas en [60]: se busca obtener la sección eficaz triple-diferencial de una partícula de agua en dos energías fotónicas $\hbar\omega$, detectando dos fotoelectrones con la misma energía cinética. Usando un enfoque de CI-Singles sobre el estado fundamental del sistema, se construye un Hamiltoniano de dos electrones, para los electrones en el continuo que corresponden a cada par de orbitales del estado inicial. Se utilizaron las distribuciones electrónicas congeladas, junto con los centros de Coulomb, para construir una función que represente al potencial asintótico que modele de forma aproximada al problema. La solución numérica para la función de onda de scattering de 2 electrones es obtenida resolviendo sistemas lineales, que surgen de una descomposición en ondas parciales, utilizando el método de Funciones Sturmianas Generalizadas, de los estados inicial y final [62].

Encontrar la sección eficaz de esta molécula requiere calcular la solución numérica de 18 sistemas lineales diferentes, cuyas matrices asociadas poseen dimensiones (finales) entre 15100×15100 y 21325×21325, utilizando aritmética doble flotante compleja. La Figura 4.17 muestra cómo evoluciona la sección eficaz a medida que se agregan más combinaciones de momentos angulares a la simulación, ie. se agregan

más bloques de filas y columnas a los sistemas lineales, recalculando las soluciones utilizando el algoritmo RLLS. Se agregan nuevas combinaciones hasta que la sección eficaz es la misma que la obtenida en [60].

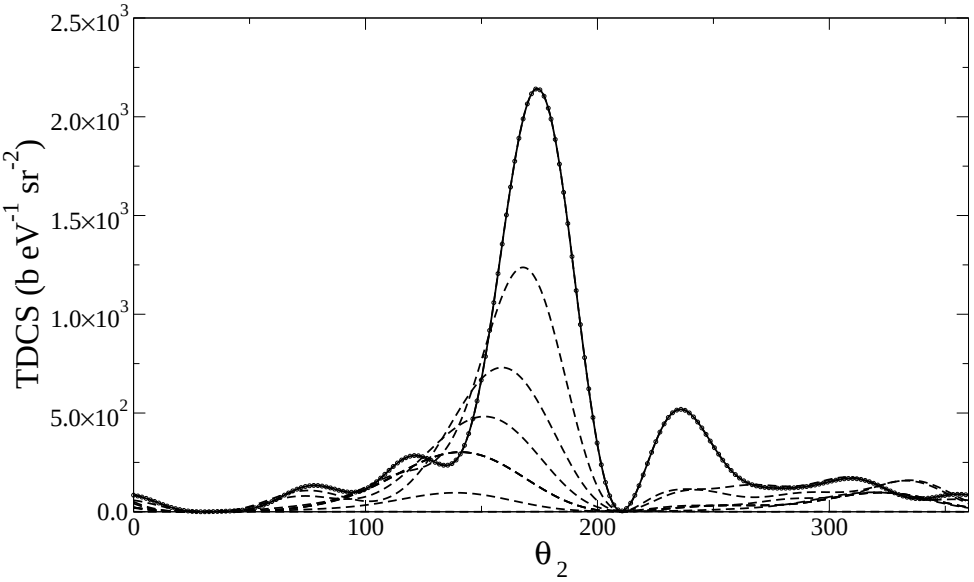

FIGURA 4.17: Convergencia de la sección eficaz de la doble fotoionización del agua, agregando bloques de 1250-2500 elementos por iteración. La línea continua corresponde al cálculo previo utilizando gradiente conjugado en [60].

Todos los experimentos en esta subsección fueron desarrollados en aritmética flotante de doble precisión compleja, sobre un servidor con las siguientes especificaciones de hardware:

CPU	Intel Xeon CPU E5-2680 v4
Frecuencia	2.40 GHz
Cores/Hilos	14
Memoria RAM	128 GiB DDR4

Los códigos fueron vinculados con kernels de BLAS de `Intel MKL composer_xe` 2020.1.217 y la versión 4.5 de `OpenMP`, compilada utilizando `GCC 10.2.1`.

La Figura 4.18 muestra la escalabilidad de la solución multi-hilo del algoritmo `RLLS` aplicada a este problema, promediando los GFLOPS realizados para solucionar los 18 sistemas lineales, utilizando 1-8 hilos de cómputo. Se muestra además la desviación estándar del rendimiento en los 18 problemas, que tienen una diferencia importante con respecto a las configuraciones de pruebas de las secciones anteriores: todos los bloques tienen dimensiones variables, las cuales además pueden ser rectangulares y podrían, en principio, comprometer la eficiencia de cómputo de la solución final. Un ejemplo de la estructura en bloques de las matrices se muestra en la Figura 4.19.

Los resultados de escalabilidad y la evolución de la sección eficaz permiten concluir que la estabilidad del algoritmo es buena y prueban que es posible realizar experimentos numéricos en simulaciones atómicas que coinciden con el estado del arte, en tiempos mucho menores a los obtenidos anteriormente, aprovechando eficientemente los recursos de cómputo que se encuentren disponibles.

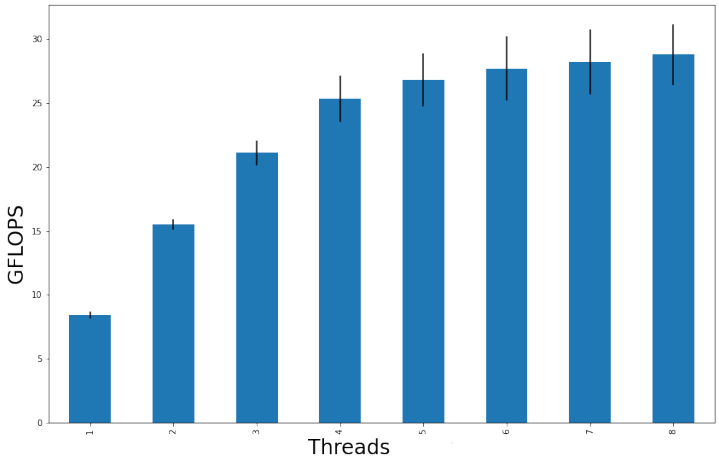

FIGURA 4.18: GFLOPS (promedio y desviación estándar) obtenidos por número de hilos utilizados.

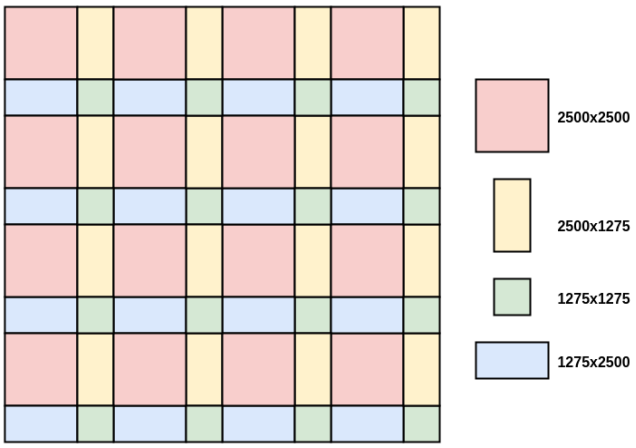

FIGURA 4.19: Ejemplo de matriz de la ecuación de Schrödinger y dimensiones de bloques para la doble fotoionización de H_2O.

En resumen, en este capítulo se mostraron las distintas implementaciones realizadas para resolver sistemas lineales con matrices latentes utilizando paralelismo por tareas, con dos de los frameworks más utilizados para simulaciones numéricas de alto rendimiento en el lenguaje `C`: `OpenMP` y `OmpSs`. Ambas librerías permitieron generar soluciones con buena escalabilidad.

Se realizaron simulaciones numéricas para comparar estas soluciones con la manera tradicional de resolver la decomposición QR, demostrando que el enfoque propuesto no sólo realiza la descomposición de manera más rápida, si no que es capaz de además resolver sistemas lineales intermedios en la búsqueda del tamaño del sistema lineal final que permita obtener una buena solución para cualquier estructura atómica que se pretenda simular, utilizando de mejor manera los recursos computacionales disponibles y resultando en tiempos de ejecución mucho menores.

Además, estas herramientas fueron adaptadas al entorno de las simulaciones atómicas, realizando experimentos para los problemas de Doble Fotoionización del Helio y Agua, mostrando resultados comparables con el estado del arte con tiempos de ejecución menores a los vistos en la literatura.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

Mediante la presentación del concepto de los *Sistemas Lineales Latentes*, el presente trabajo logra dar una formulación y un marco de trabajo exitosos para la solución de sistemas lineales cuya dimensión no es conocida a priori.

En los primeros capítulos de este trabajo, se presentaron la motivación (basada en la simulación de sistemas atómicos de pocos cuerpos) y se introdujeron los conceptos del álgebra lineal numérica de alto desempeño necesarios para enunciar el concepto de *sistemas lineales latentes*.

Se exploraron dos descomposiciones matriciales (LU y QR), con sus respectivas reformulaciones para problemas de actualización y diversas librerías de programación que permiten resolver la formulación del problema. Se presentaron reportes de rendimiento y argumentos para la elección de las herramientas (`OpenMP + Intel MKL`) que se utilizaron en la implementación final del algoritmo RLLS.

La formulación del algoritmo RLLS se muestra como un problema de actualización de factorizaciones matriciales, el cual es resuelto mediante la aplicación de un algoritmo de factorización QR en su forma *lazy*. La solución del mismo puede explotar el paralelismo de tareas y soluciona las posibles reducciones en el grado de paralelismo de este problema, proponiendo una ejecución asíncrona de múltiples sistemas lineales en niveles intermedios, incrementando la concurrencia y, de esa forma, evitando la posibilidad de tener recursos sin utilizar (por ejemplo, mediante la no utilización de algunos núcleos computacionales).

Este enfoque permite producir una ejecución de código más rápida, a expensas de realizar algunos cálculos especulativos extra, que pueden no ser parte de la solución final. El algoritmo RLLS permite resolver algunos de los problemas más importantes que existían al aplicar enfoques anteriores para la solución de sistemas derivados del problema de Coulomb de pocos cuerpos:

- El desconocimiento de la dimensión que permite resolver satisfactoriamente el problema, que derivaba en el ciclo ineficiente de *prueba y error*.
- La posibilidad de no encontrar solución del sistema lineal mediante un método iterativo (fuertemente dependiente de la estructura de la matriz, en principio desconocida).
- La estabilidad numérica del algoritmo a medida que el sistema latente incrementa su tamaño (resuelto por el uso de la descomposición QR).

El proceso fue exitosamente implementado en el lenguaje de programación `C`, con la creación de rutinas de manejo de datos binarios de `Fortran`, utilizando las librerías

`OpenMP` e `Intel MKL`, disponibles de manera abierta. La implementación permitió simular procesos físicos, como la doble fotoionización del Helio, con eficiencias mayores a las reportadas hasta el momento con herramientas anteriores.

Esta herramienta permitirá simular procedimientos físicos que no podían ser atacados en el pasado debido a su complejidad y tamaño, de manera eficiente y con mejoras sustanciales en tiempos de ejecución, para arquitecturas multinúcleo de propósito general, dando la posibilidad de conseguir resultados más allá del actual estado del arte.

5.2. Trabajo Futuro

Como se menciona en las conclusiones, la herramienta permite resolver problemas más complejos que los que se intentó resolver en el pasado. Por ejemplo, se podrán abordar, en el futuro, procesos de fotoionización múltiple, sistemas atómicos más complejos que requieran la expansión de la función de onda en numerosos momentos angulares (que resultan en una matriz con muchos bloques), e incluso aquellas situaciones dinámicas con electrones de mayor energía, para los que sería necesario contar con una mayor cantidad de funciones base para una descripción precisa del sistema (resultando en mayores tamaños de matriz).

Una de las actuales debilidades de la herramienta es la necesidad de generar los datos en herramientas desarrolladas con anterioridad en otro lenguaje de programación (`Fortran`), para luego ser ingeridos y resolver los sistemas lineales necesarios. Desarrollar una solución que permita vincular la generación de los datos y la posterior solución del sistema latente de manera automática mejorará la eficiencia del proceso general. Un lenguaje de programación de *scripting* como `Python` puede ser una opción interesante para codificar esta solución.

Si bien este trabajo se centró en obtener la mayor eficiencia posible usando CPUs, es posible extrapolar lo hecho al uso de GPUs como un dispositivo auxiliar, tanto para la generación de datos como para la solución del sistema lineal. `OpenMP` permite vincular placas de procesamiento gráfico a la ejecución del código y su *scheduler* puede utilizarlas como un recurso extra de cómputo, codificando previamente las rutinas que permitan realizar las tareas en ese tipo de hardware. Aún así, existen otras técnicas para intentar utilizar mejor aún los recursos existentes, como el concepto de *maleabilidad*: la posibilidad de asignar una cantidad de hilos de cómputo variable a una tarea a medida que van quedando inactivos o se necesite acelerar su ejecución.

Finalmente, la presente herramienta funciona de manera correcta para matrices que entran en la memoria RAM disponible y, si bien la disponibilidad de este recurso es cada vez más grande, algunos problemas de interés pueden requerir una cantidad de almacenamiento muy grande, incluso para estándares actuales. Para esos casos, es necesario implementar una estrategia de descomposición Out Of Core, pero el trabajo hecho en la reformulación del problema y la bibliografía disponible apuntan a que la estrategia puede implementarse, con algunas modificaciones a la base de código actual.

Bibliografía

[1] Wilhelm Wien. «XXX. On the division of energy in the emission-spectrum of a black body». En: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 43.262 (1897), págs. 214-220. DOI: 10.1080/14786449708620983. eprint: https://doi.org/10.1080/14786449708620983. URL: https://doi.org/10.1080/14786449708620983.

[2] Elain M Marzluff. *Quantum Chemistry Notes*. 2020. URL: https://chem.libretexts.org/Sandboxes/marzluff_at_grinnell.edu/Username%5C%3A_marzluff@grinnell.edu/Unit_1%5C%3A_Quantum_Chemistry%5C%2C_Spectroscopy_and_Bonding.

[3] Theresa Julia Zielinski et al. «Quantum States of Atoms and Molecules». En: *Journal of Chemical Education* 82.12 (2005), pág. 1880. DOI: 10.1021/ed082p1880.2. URL: https://doi.org/10.1021/ed082p1880.2.

[4] J. N. Das, K. Chakrabarti y S. Paul. «Hyperspherical partial wave calculation for double photoionization of the helium atom at 20 eV excess energy». en. En: *Journal of Physics B: Atomic, Molecular and Optical Physics* 36.13 (jul. de 2003), pág. 2707. ISSN: 0953-4075. DOI: 10.1088/0953-4075/36/13/302. URL: http://iopscience.iop.org/0953-4075/36/13/302 (visitado 04-06-2014).

[5] P. Bolognesi et al. «Complementary TDCS for the photo-double ionization of He at 40 eV above the threshold in unequal energy-sharing conditions». en. En: *Journal of Physics B: Atomic, Molecular and Optical Physics* 34.15 (ago. de 2001), pág. 3193. ISSN: 0953-4075. DOI: 10.1088/0953-4075/34/15/321. URL: http://iopscience.iop.org/0953-4075/34/15/321 (visitado 04-06-2014).

[6] J P Wightman, S Cvejanovic y T J Reddish. «(y, 2e) cross section measurements of D2 and He». En: *Journal of Physics B: Atomic, Molecular and Optical Physics* 31.8 (1998), pág. 1753. URL: http://stacks.iop.org/0953-4075/31/i=8/a=024.

[7] K. Soejima et al. «Linear and Circular Dichroism in the Double Photoionization of Helium». En: *Phys. Rev. Lett.* 83 (8 ago. de 1999), págs. 1546-1549. DOI: 10.1103/PhysRevLett.83.1546. URL: http://link.aps.org/doi/10.1103/PhysRevLett.83.1546.

[8] R. Wehlitz et al. «Electron-energy and -angular distributions in the double photoionization of helium». En: *Phys. Rev. Lett.* 67 (27 dic. de 1991), págs. 3764-3767. DOI: 10.1103/PhysRevLett.67.3764. URL: http://link.aps.org/doi/10.1103/PhysRevLett.67.3764.

[9] R. Dörner et al. «Ratio of Cross Sections for Double to Single Ionization of He by 85-400 eV Photons». En: *Phys. Rev. Lett.* 76 (15 abr. de 1996), págs. 2654-2657. DOI: 10.1103/PhysRevLett.76.2654. URL: http://link.aps.org/doi/10.1103/PhysRevLett.76.2654.

[10] R Wehlitz et al. «Photon energy dependence of ionization-excitation in helium at medium energies». En: *Journal of Physics B: Atomic, Molecular and Optical Physics* 30.2 (1997), pág. L51. URL: http://stacks.iop.org/0953-4075/30/i=2/a=004.

[11] J. A. R. Samson et al. «Double photoionization of helium». En: *Phys. Rev. A* 57 (3 mar. de 1998), págs. 1906-1911. DOI: 10.1103/PhysRevA.57.1906. URL: http://link.aps.org/doi/10.1103/PhysRevA.57.1906.

[12] V. Mergel et al. «Helicity Dependence of the Photon-Induced Three-Body Coulomb Fragmentation of Helium Investigated by Cold Target Recoil Ion Momentum Spectroscopy». En: *Phys. Rev. Lett.* 80 (24 jun. de 1998), págs. 5301-5304. DOI: 10.1103/PhysRevLett.80.5301. URL: http://link.aps.org/doi/10.1103/PhysRevLett.80.5301.

[13] H. Bräuning et al. «Absolute triple differential cross sections for photo-double ionization of helium-experiment and theory». en. En: *Journal of Physics B: Atomic, Molecular and Optical Physics* 31.23 (dic. de 1998), pág. 5149. ISSN: 0953-4075. DOI: 10.1088/0953-4075/31/23/012. URL: http://iopscience.iop.org/0953-4075/31/23/012 (visitado 04-06-2014).

[14] S. Cvejanovic et al. «Photodouble ionization of helium at an excess energy of 40 eV». en. En: *Journal of Physics B: Atomic, Molecular and Optical Physics* 33.2 (ene. de 2000), pág. 265. ISSN: 0953-4075. DOI: 10.1088/0953-4075/33/2/311. URL: http://iopscience.iop.org/0953-4075/33/2/311 (visitado 04-06-2014).

[15] J. Mazeau et al. «Angular correlations in near-threshold double photoionization of krypton». En: *Phys. Rev. Lett.* 67 (7 ago. de 1991), págs. 820-823. DOI: 10.1103/PhysRevLett.67.820. URL: https://link.aps.org/doi/10.1103/PhysRevLett.67.820.

[16] J Ullrich et al. «Recoil-ion and electron momentum spectroscopy: reaction-microscopes». En: *Reports on Progress in Physics* 66.9 (ago. de 2003), pág. 1463. DOI: 10.1088/0034-4885/66/9/203. URL: https://dx.doi.org/10.1088/0034-4885/66/9/203.

[17] C. W. McCurdy et al. «Theoretical treatment of double photoionization of helium using a B -spline implementation of exterior complex scaling». En: *Physical Review A* 69.3 (mar. de 2004), pág. 032707. DOI: 10.1103/PhysRevA.69.032707. URL: http://link.aps.org/abstract/PRA/v69/e032707.

[18] F Maulbetsch y J S Briggs. «Angular distribution of electrons following double photoionization». En: *Journal of Physics B: Atomic, Molecular and Optical Physics* 26.11 (1993), pág. 1679. URL: http://stacks.iop.org/0953-4075/26/i=11/a=005.

[19] P. Selles, L. Malegat y A. K. Kazansky. «Ab initio calculation of the whole set of He double-photoionization cross sections». En: *Phys. Rev. A* 65 (3 feb. de 2002), pág. 032711. DOI: 10.1103/PhysRevA.65.032711. URL: http://link.aps.org/doi/10.1103/PhysRevA.65.032711.

[20] Marcel Pont y Robin Shakeshaft. «Theory of double photoionization of a two-electron atom: Circumventing the boundary conditions». En: *Phys. Rev. A* 51 (1 ene. de 1995), págs. 494-499. DOI: 10.1103/PhysRevA.51.494. URL: http://link.aps.org/doi/10.1103/PhysRevA.51.494.

[21] G. Gasaneo et al. «Chapter 7 - Three-Body Coulomb Problems with Generalized Sturmian Functions». En: *Advances in Quantum Chemistry*. Ed. por Philip E. Hoggan. Vol. Volume 67. Proceedings of MEST 2012: Exponential Type Orbitals for Molecular Electronic Structure Theory. Academic Press, 2013, págs. 153-216. URL: http://www.sciencedirect.com/science/article/pii/B9780124115446000078 (visitado 03-07-2014).

[22] J. M. Randazzo et al. «Generating optimal Sturmian basis functions for atomic problems». En: *Physical Review A* 81.4 (abr. de 2010), pág. 042520. DOI: 10.1103/PhysRevA.81.042520. URL: http://link.aps.org/doi/10.1103/PhysRevA.81.042520.

[23] John Scales Avery y James Emil Avery. «Coulomb Sturmians as a basis for molecular calculations». En: *Molecular Physics* 110.15-16 (2012), págs. 1593-1608. DOI: 10.1080/00268976.2012.658876. eprint: https://doi.org/10.1080/00268976.2012.658876. URL: https://doi.org/10.1080/00268976.2012.658876.

[24] Emmanuel Foumouo et al. «Theory of multiphoton single and double ionization of two-electron atomic systems driven by short-wavelength electric fields: An ab initio treatment». En: *Phys. Rev. A* 74 (6 dic. de 2006), pág. 063409. DOI: 10.1103/PhysRevA.74.063409. URL: https://link.aps.org/doi/10.1103/PhysRevA.74.063409.

[25] H. Shull y P. O. Löwdin. «Role of the continuum in superposition of configuration». En: *J. Chem. Phys.* 30 (1959), pág. 617.

[26] A. L. Frapiccini et al. «Fast Track Communication: A boundary adapted spectral approach for breakup problems». En: *Journal of Physics B: Atomic, Molecular and Optical Physics* 43.10 (2010), pág. 101001. ISSN: 0953-4075. DOI: 10.1088/0953-4075/43/10/101001. URL: http://iopscience.iop.org/0953-4075/43/10/101001?fromSearchPage=true.

[27] J. M. Randazzo et al. «Solving three-body-breakup problems with outgoing-flux asymptotic conditions». En: *Phys. Rev. A* 84 (5 nov. de 2011), pág. 052715. DOI: 10.1103/PhysRevA.84.052715. URL: http://link.aps.org/doi/10.1103/PhysRevA.84.052715.

[28] G. Gasaneo et al. «S-model calculations for high-energy-electron-impact double ionization of helium». En: *Phys. Rev. A* 87 (4 abr. de 2013), pág. 042707. DOI: 10.1103/PhysRevA.87.042707. URL: http://link.aps.org/doi/10.1103/PhysRevA.87.042707.

[29] Raimonds Peterkops. *Theory of ionization of atoms by electron impact / R. K. Peterkop ; translation edited by D. G. Hummer from a draft translation by Elliot Aronson*. English. Colorado Associated University Press, Boulder : 1977, vi, 263 p. : ISBN: 0870811053.

[30] Dmitrii Aleksandrovich Varshalovich, Anatolij Nikolaevic Moskalev y Valerii Kelmanovich Khersonskii. *Quantum theory of angular momentum*. World Scientific, 1988.

[31] Darío M. Mitnik et al. «Computational methods for Generalized Sturmians basis». En: *Computer Physics Communications* 182.5 (2011), págs. 1145-1155. ISSN: 0010-4655. DOI: DOI:10.1016/j.cpc.2011.01.016.

[32] SP Goldman. «Uncoupling correlated calculations in atomic physics: Very high accuracy and ease». En: *Physical Review A* 57.2 (1998), R677.

[33] Turco F. et al. «Optimización de Bases Sturmianas para el Problema de Tres Cuerpos Cuántico utilizando Procesadores Gráficos.» En: *101a Reunión de la Asociación Física Argentina* (2016). URL: https://www.fisica.org.ar/wp-content/blogs.dir/33/files/sites/33/2019/02/resumen_rafa_2016.pdf.

[34] Luis Biedma, Flavio Colavecchia y Enrique S Quintana-Ortí. «Solution of few-body Coulomb problems with latent matrices on multicore processors». En: *Procedia Computer Science* 108 (2017), págs. 1743-1752.

[35] Randazzo J. Biedma L. Colavecchia F. «Task-Parallelized Numerical linear algebra methods to solve the Few Body Coulomb Problem». En: *Proceedings of COPIAMC 2019* (2019). URL: https://copiamc.event.univ-lorraine.fr/data/pages/Proceedings_of_COPIAMC_2019.pdf.

[36] Randazzo J. Biedma L. Colavecchia F. «Numerical methods for few body problems in atomic physics». En: *Proceedings of 32nd International Conference on Photonic, Electronic and Atomic Collisions (VICPEAC21)* (2021). URL: https://icpeac2021.ca/abstracts/ViCPEAC2021_Abstract_Book-PUBLIC.pdf.

[37] Jean-Claude Martzloff. *A History of Chinese Mathematics*. Ene. de 2006, págs. 1-485. ISBN: 978-3-540-33782-9. DOI: 10.1007/978-3-540-33783-6.

[38] Gene H. Golub y Charles F. Van Loan. *Matrix Computations*. Third. The Johns Hopkins University Press, 1996.

[39] Victor Eijkhout, Robert van de Geijn y Edmond Chow. *Introduction to High Performance Scientific Computing*. Ene. de 2016. DOI: 10.5281/zenodo.49897.

[40] Jack J Dongarra et al. *Numerical linear algebra for high-performance computers*. Vol. 7. Siam, 1998.

[41] Instrucciones por Segundo (EN). *Instrucciones por Segundo (EN) — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Instructions_per_second.

[42] BLAS. *Basic Linear Algebra Subprograms (BLAS)*. 1979. URL: http://www.netlib.org/blas/.

[43] E. Anderson et al. *LAPACK Users' Guide*. Third. Philadelphia, PA: Society for Industrial y Applied Mathematics, 1999. ISBN: 0-89871-447-8 (paperback).

[44] Zhang Xianyi, Wang Qian y Werner Saar. «OpenBLAS: An optimized BLAS library». En: *Accedido: Agosto* (2016).

[45] Robert D. Blumofe et al. «Cilk: An Efficient Multithreaded Runtime System». En: *SIGPLAN Not.* 30.8 (ago. de 1995), págs. 207-216. ISSN: 0362-1340. DOI: 10.1145/209937.209958. URL: https://doi.org/10.1145/209937.209958.

[46] Alfredo Buttari et al. «A Class of Parallel Tiled Linear Algebra Algorithms for Multicore Architectures». En: *CoRR* abs/0709.1272 (2007). arXiv: 0709.1272. URL: http://arxiv.org/abs/0709.1272.

[47] Pedro Gonnet, Aidan B. G. Chalk y Matthieu Schaller. «QuickSched: Task-based parallelism with dependencies and conflicts». En: *CoRR* abs/1601.05384 (2016). arXiv: 1601.05384. URL: http://arxiv.org/abs/1601.05384.

[48] Alejandro Duran et al. «OmpSs: a Proposal for Programming Heterogeneous Multi-Core Architectures.» En: *Parallel Processing Letters* 21 (jun. de 2011), págs. 173-193. DOI: 10.1142/S0129626411000151.

[49] Rohit Chandra et al. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[50] Flavio Colavecchia. «Accelerating spectral atomic and molecular collisions methods with graphics processing units». En: *Computer Physics Communications* 185 (jul. de 2014). DOI: 10.1016/j.cpc.2014.03.026.

[51] *PLASMA project home page*. http://icl.cs.utk.edu/plasma.

[52] Field G. Van Zee. libflame*: The Complete Reference*. www.lulu.com, 2009.

[53] E. S. Quintana-Ortí y R. A. van de Geijn. «Updating an LU factorization with Pivoting». En: *ACM Trans. Math. Soft.* 35.2 (jul. de 2008), 11:1-11:16.

[54] Gilbert W Stewart. *Matrix algorithms: volume 1: basic decompositions*. SIAM, 1998.

[55] Brian C. Gunter y Robert A. van de Geijn. «Parallel Out-of-Core Computation and Updating the QR Factorization». En: *ACM Trans. Math. Soft.* 31.1 (mar. de 2005), págs. 60-78.

[56] Christian Bischof y Charles Van Loan. «The WY representation for products of Householder matrices». En: *SIAM Journal on Scientific and Statistical Computing* 8.1 (1987), s2-s13.

[57] Robert Schreiber y Charles Van Loan. «A storage-efficient WY representation for products of Householder transformations». En: *SIAM Journal on Scientific and Statistical Computing* 10.1 (1989), págs. 53-57.

[58] Erik Elmroth y Fred Gustavson. «Applying recursion to serial and parallel QR factorization». En: *IBM Journal of Research and Development* 44 (ago. de 2000), págs. 605-624. DOI: 10.1147/rd.444.0605.

[59] A. R. Mitchell. «J. H. Wilkinson, The Algebraic Eigenvalue Problem (Clarendon Press, Oxford, 1965), 662pp., 110s.» En: *Proceedings of the Edinburgh Mathematical Society* 15.4 (1967), págs. 328-328. DOI: 10.1017/S0013091500012104.

[60] J. M. Randazzo et al. «Photo-double-ionization of water at 20 eV above threshold». En: *Phys. Rev. A* 101 (3 mar. de 2020), pág. 033407. DOI: 10.1103/PhysRevA.101.033407. URL: https://link.aps.org/doi/10.1103/PhysRevA.101.033407.

[61] Juan M Randazzo et al. «Double photoionization of helium: a generalized Sturmian approach». En: *The European Physical Journal D* 69 (2015), págs. 1-10.

[62] P Bolognesi et al. «A combined experimental and theoretical study of photodouble ionization of water at 32 eV excess energy and unequal energy sharing». En: *Journal of Physics B: Atomic, Molecular and Optical Physics* 54.3 (feb. de 2021), pág. 034002. DOI: 10.1088/1361-6455/abd647. URL: https://doi.org/10.1088/1361-6455/abd647.