Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXV, págs. 435-453 (artículo completo)
Martín I. Idiart, Ana E. Scarabino y Mario A. Storti (Eds.)
La Plata, 7-10 Noviembre 2017

ACELERACIÓN DE UNA HERRAMIENTA DE MECÁNICA DE SÓLIDOS EN ARQUITECTURAS MULTI- Y MANY-CORES

Rodrigo Bayá Crapuchett[a], Pablo Castrillo[b], Jorge Pérez Zerpa[b], Ernesto Dufrechou[a] y Pablo Ezzatti[a]

[a]*Instituto de la Computación (INCO), Universidad de la República, Montevideo, Uruguay.*

[b]*Instituto de Estructuras y Transporte (IET), Universidad de la República, Montevideo, Uruguay.*

Palabras Clave: Sistemas lineales, GPUs, Elasticidad no lineal

Resumen. En el Instituto de Estructuras y Transporte de la Facultad de Ingeniería de la Universidad de la República - Uruguay, se está trabajando en herramientas numéricas que permitan resolver problemas vinculados al desarrollo de nuevos métodos de diagnóstico de enfermedades utilizando la mecánica de sólidos. Para poder simular el comportamiento mecánico de los tejidos biológicos, es necesario contar con herramientas computacionales que permitan determinar cómo un sólido se deforma ante la aplicación de esfuerzos conocidos. En esfuerzos previos se implementaron rutinas capaces de simular el comportamiento no lineal de sólidos, conformando el simulador MecSolENL. Si bien esta herramienta ha mostrado resultados alentadores, la resolución de problemas de grandes dimensiones o la búsqueda de soluciones de alta precisión numérica presentan importantes limitantes desde el punto de vista computacional. En este trabajo, se estudia la herramienta desarrollada anteriormente, MecSolENL, y se constata que los principales requerimientos computacionales están dados por el manejo de operaciones de álgebra lineal numérica, y en particular, por la resolución de sistemas lineales de ecuaciones dispersos. Se estudian diferentes bibliotecas, disponibles públicamente, para explotar plataformas multi-core, como son MUMPS, PARDISO y PETSc, así como desarrollos propios de métodos iterativos de resolución de sistemas lineales sobre GPUs. La evaluación experimental muestra que los nuevos métodos estudiados ofrecen importantes reducciones en los tiempos de ejecución cuando se los compara con la variante original de la herrameinta.

1. INTRODUCCIÓN

La mecánica de sólidos estudia diferentes problemáticas relacionadas con la resistencia, rigidez o la estabilidad de cuerpos sometidos a cargas. De este tipo de análisis se desprenden diversas técnicas, las cuales permiten simular el comportamiento de estos cuerpos.

En el Instituto de Estructuras y Transporte de la Facultad de Ingeniería de la Universidad de la República - Uruguay (IET-FING-UdelaR) se está trabajando en el desarrollo de herramientas numéricas para resolver problemas vinculados al desarrollo de nuevos métodos de diagnóstico precoz de enfermedades. Para poder simular el comportamiento mecánico de los tejidos biológicos es necesario contar con alguna herramienta computacional (solver) que permita determinar cómo un sólido se deforma ante la aplicación de esfuerzos conocidos. En la mayoría de los casos los problemas a resolver son no lineales, los cuales requieren un alto costo computacional. Además, para la resolución de problemas no lineales, se utilizan métodos de tipo Newton-Raphson, en los cuales se debe resolver un sistema lineal en cada iteración.

En Pérez Zerpa (2016); Pérez Zerpa y Canelas (2016) se presentan métodos para resolución de problemas inversos de interés en Biomecánica, entre los cuales suelen ser considerados problemas no lineales. Para realizar simulaciones de comportamiento no lineal de sólidos, en dichos trabajos, se implementó una herramienta que en este trabajo llamaremos MecSolENL. Si bien esta herramienta ha mostrado resultados alentadores, la resolución de problemas de grandes dimensiones o la búsqueda de soluciones de alta precisición numérica presentan importantes limitantes desde el punto de vista computacional. En otras palabras, la resolución de problemas de grandes dimensiones implican tiempos de cómputo prohibitivos.

En este trabajo, primero se evalúa la herramienta desarrollada anteriormente, MecSolENL, donde se constata que los principales requerimientos computacionales están dados por el manejo de operaciones de álgebra lineal numérica, y en particular, por la resolución de sistemas lineales de ecuaciones dispersos. Considerando esta situación, se estudian diferentes bibilotecas disponibles públicamente, como son MUMPS Amestoy et al. (2000), PARDISO (2013) y PETSc Balay et al. (2012), que permiten aplicar técnicas de computación de alto desempeño (del inglés, HPC) para la resolución de dichos sistemas sobre arquitecturas multicore. Además, se estudió el uso de un solver iterativo, desarrollado en el Instituto de Computación de la Universidad de la República - Uruguay (INCO-FING-UdelaR) llamado PCG_{GPU}, el cual utiliza la GPU para su resolución.

La evaluación experimental del uso de los nuevos métodos estudiados, realizada sobre un equipo de arquitectura híbrida (CPU multicore + GPU), muestra que el uso de las bibliotecas que ejecutan en la CPU, junto con técnicas de precondicionamiento en el caso de los métodos iterativos y de ordenamiento en el caso de los métodos directos, permiten alcanzar aceleraciones de hasta $16\times$ cuando se compara con la versión orginal de la herramienta de simulación. En el caso de de la implementación iterativa en GPU, se lograron mejoras de hasta 30 veces para los casos de ejecución de mayor porte cuando se mide el tiempo de la herramienta MecSolENL en su totalidad.

El resto del artículo se estructura de la siguiente manera. En la Sección 2 se describen los principales conceptos matemáticos que sustentan la mecánica de sólidos, así como la aplicación del método de los elementos finitos para discretizar el problema. Posteriormente, se resume el estudio de desempeño, desde el punto de vista empírico, de las herramientas ya desarrolladas en la Sección 3. Luego, en la Sección 4, se presentan las diferentes bibliotecas para la resolución de sistemas lineales en CPU y el solver desarrollado en GPU; También se describen someramente las técnicas de precondicionado (métodos iterativos) y ordenamiento (métodos directos).

Un compendio de los resultados obtenidos más destacados, así como la discusión asociada a los mismos se puede encontrar en la Sección 5. Por último, en la Sección 6 se presentan las principales conclusiones conseguidas durante el trabajo y las posibles líneas de trabajo a futuro para extender las herramientas.

2. PROBLEMA

El número de muertes por cáncer o Enfermedades Cardiovasculares (ECV) ha aumentado en las últimas décadas, por ejemplo en 2012 el 31 % del total de las defunciones fue causado por ECV WHO (2015). Es por esto que el desarrollo de nuevos métodos para la detección precoz de dichas enfermedades se ha convertido en uno de los problemas de investigación de mayor interés en diversas áreas científicas, incluyendo la de la mecánica de sólidos.

En las últimas décadas se ha comenzado a integrar conceptos de la mecánica de sólidos al desarrollo de estos nuevos métodos, aprovechando por ejemplo el hecho de que tejidos cancerígenos son más rígidos que tejidos sanos. Los tejidos biológicos están compuestos por materiales cuyo comportamiento es, de forma general, hiperelástico, viscoelástico, anisótropo y está sujeto a grandes deformaciones (ver por ejemplo Valdez-Jasso et al. (2011); Holzapfel y Ogden (2003); Holzapfel (2000b)).

Existen varios trabajos de mecánica de sólido aplicados a tejidos biológicos. Por ejemplo, en Moireau et al. (2012) se presenta un modelo 3D que integra la mecánica de fluidos y mecánica de sólidos para reproducir el comportamiento de parte del árbol arterial y su flujo sanguíneo. En Goenezen et al. (2012) se aplican modelos de elasticidad no lineal para estimar la deformación de tejidos mamarios con tumores a partir de imágenes ecográficas (sin irradiar a la paciente), logrando distinguir tumores malignos de benignos en un gran número de los casos estudiados. La resolución de los problemas numéricos de la mayoría de estas aplicaciones tiene un elevado costo computacional. Estudios similares se realizan para modelar el comportamiento de tejido de corazón y arterias, para lo cual existen en particular, un intenso desarrollo de metodologías numéricas orientadas a la aplicación de técnicas de HPC (Brinkhues et al. (2013); Pavarino et al. (2015)).

2.1. Conceptos de mecánica de sólidos

En esta sección se describen brevemente algunos conceptos de la mecánica de sólidos no lineal (o elasticidad finita), presentados en diversa literatura del área, como por ejemplo Belytschko et al. (2001).

Dado un sólido que ocupa la región $\mathcal{B}_R \subset \mathcal{E}^3$, siendo $\mathcal{E}^3$ el espacio euclidiano tridimensional, se asumirá que no hay tensiones aplicadas en esa configuración, es decir, $\mathcal{B}_R$ es una configuración de referencia natural. Se considerará que hay aplicada una carga muerta $\mathbf{f}$ en el borde $\Gamma_{\mathbf{f}}$ (incluyendo el borde donde $\mathbf{f} = \mathbf{0}$). Esta carga producirá una deformación $\mathcal{X} : \mathcal{B}_R \to \mathcal{B}$, siendo $\mathcal{B}$ la región ocupada por el cuerpo deformado. La energía total de deformación del cuerpo es definida como:

$$\mathcal{E}_{\text{int}} = \int_{\mathcal{B}_R} \Psi(\mathbf{E}) \, \mathrm{d}V, \tag{1}$$

donde Ψ es la densidad de energía de deformación, la cual define el comportamiento constitutivo del material, y $\mathbf{E}$ es el tensor de deformaciones de Lagrange dado por:

$$\mathbf{E} = \frac{1}{2} \left(\mathbf{F}^{\mathrm{T}} \mathbf{F} - \mathbf{I} \right), \tag{2}$$

donde $\mathbf{I}$ es el tensor identidad y $\mathbf{F}$ el gradiente de la función de deformación:

$$\mathbf{F} = \nabla \mathcal{X}. \tag{3}$$

El trabajo de las fuerzas externas aplicadas puede calcularse también en la configuración de referencia:

$$W_{\text{ext}} = \int_{\Gamma_{\mathbf{f}}} \mathbf{f}_R \cdot (\mathcal{X}(\mathbf{X}) - \mathbf{X}) \, \mathrm{d}A, \tag{4}$$

donde $\mathbf{f}_R$ es la carga definida en la superficie de la configuración de referencia, la cual será considerada constante al deformarse el sólido. La expresión de la energía potencial total es entonces:

$$\mathcal{E}_{\text{tot}} = \mathcal{E}_{\text{int}} - W_{\text{ext}}, \tag{5}$$

es decir

$$\mathcal{E}_{\text{tot}} = \int_{\mathcal{B}_R} \Psi(\mathcal{X}(\mathbf{X})) \, \mathrm{d}V - \int_{\Gamma_{\mathbf{f}}} \mathbf{f}_R \cdot (\mathcal{X}(\mathbf{X}) - \mathbf{X}) \, \mathrm{d}A. \tag{6}$$

El comportamiento constitutivo del material, es decir la relación entre tensiones (o esfuerzos) y deformaciones está dada por la función de densidad de energía de deformación Ψ Holzapfel (2000a). En este trabajo se considerará el comportamiento (o energía) de Curnier Curnier (1994):

$$\Psi(\mathcal{X}(\mathbf{X})) = \lambda \, (J - \log(J) - 1) + \mu \, \mathrm{tr}(\mathbf{E}^2), \tag{7}$$

donde $J = \det(\mathbf{F})$ y λ y μ son los parámetros de Lamé. Para este material el tensor de tensiones de Cosserat es:

$$\mathbf{S} = \lambda \, (J - 1) \, \left(\mathbf{F}^{\mathrm{T}}\mathbf{F}\right)^{-1} + 2 \, \mu \, \mathbf{E}. \tag{8}$$

2.2. Problema de Elasticidad no lineal

Las configuraciones de equilibrio estables en un sólido, se corresponden con los mínimos de la energía total. De lo anterior, el problema de elasticidad no lineal considerado consiste en, dadas las fuerzas aplicadas al sólido, encontrar la función $\mathcal{X}$ que minimiza la energía $\mathcal{E}_{\text{tot}}$, es decir:

$$\mathcal{X}^* = \arg \min_{\mathcal{X}} \{\mathcal{E}_{\text{tot}}\}. \tag{9}$$

Para resolver el problema numéricamente se utiliza el Método de los Elementos Finitos (MEF) Hughes (2000). La metodología de la técnica implica discretizar la estructura en elementos, por ejemplo elementos tetraédricos, formando entre todos la malla de la estructura Belytschko et al. (2001). La posición deformada $\mathcal{X}(\mathbf{X})$ de cada punto dentro de cada tetraedro es interpolada linealmente a partir de posiciones de los vértices. Por tanto, de resolver la Ecuación (9) se obtiene el vector de variables $\mathbf{x}$ que contiene las posiciones de todos los nodos (o vértices) de la malla.

En función de la cantidad de nodos (N) de la estructura se obtiene el número de grados de libertad de ésta, por ejemplo al utilizar elementos tetraédricos la cantidad de grados de libertad es $m \approx 3\,N$. Por lo anterior, al discretizar la estructura, la energía total pasa a ser función de un número finito de grados de libertad, es decir: $\mathcal{E}_{\text{tot}} : \mathbb{R}^m \to \mathbb{R}$, donde m es el número de grados de libertad de la estructura.

El problema es no lineal en $(\mathbf{x})$ y por lo tanto debe resolverse utilizando métodos iterativos como el método de Newton-Raphson descrito en el Algoritmo 1.

Algoritmo 1 Pseudo-código Newton-Raphson para un estado de cargas aplicado

1: $\mathbf{x}_k \leftarrow \mathbf{x}_0$
2: calcular $\nabla\mathcal{E}_{\text{tot}}(\mathbf{x}_k)$, $\nabla^2\mathcal{E}_{\text{tot}}(\mathbf{x}_k)$
3: **while** $\|\nabla\mathcal{E}_{\text{tot}}(\mathbf{x}_k)\| > \text{tol}$ **do**
4: resolver $\nabla^2\mathcal{E}_{\text{tot}}(\mathbf{x}_k)\,\mathbf{d}_k = -\nabla\mathcal{E}_{\text{tot}}(\mathbf{x}_k)$
5: $\mathbf{x}_k \leftarrow \mathbf{x}_k + \mathbf{d}_k$
6: $k \leftarrow k + 1$
7: calcular $\nabla\mathcal{E}_{\text{tot}}(\mathbf{x}_k)$, $\nabla^2\mathcal{E}_{\text{tot}}(\mathbf{x}_k)$
8: **end while**

En la línea 4 del Algoritmo 1 se debe resolver un sistema lineal cuya matriz es usualmente llamada matriz de rigidez. Las entradas de esta matriz dependen de los valores $(\mathbf{x})_k$ aunque en todo caso se conservan las propiedades de simetría y baja densidad de elementos no ceros (es decir, dispersa).

En la mayoría de los casos es necesario encontrar los desplazamientos para varios valores de carga, por lo que se suele aplicar este procedimiento para cada valor de carga creciente tomando como punto inicial el vector resultado de la carga anterior.

3. EVALUACIÓN DE DESEMPEÑO DE MECSOLENL

En esta sección se evalúa, desde el punto de vista de desempeño computacional, la herramienta de simulación de mecánica de sólidos MecSolENL Pérez Zerpa (2016). En este sentido se describen los casos de prueba y la plataforma de hardware que se utilizan en el resto del trabajo. Además, se detallan los tiempos de ejecución de la herramienta y se determinan los cuellos de botella de la misma.

3.1. Casos de prueba

Se definieron tres familias de casos de prueba, una barra sometida a tensión uniaxial (cuya solución analítica fue obtenida), un cilindro con presión interna y un cubo con inclusión. A continuación se detallan los casos de prueba.

3.1.1. Barra con deformación

Supongamos que sobre una barra aplicamos una deformación $\mathcal{X}$ dada por la expresión $\mathcal{X}(X_1, X_2, X_3) = (\alpha X_1, \beta X_2, \beta X_3)$, como se puede ver en la Figura 1, donde α es el alargamiento axial de la barra y β el alargamiento transversal.

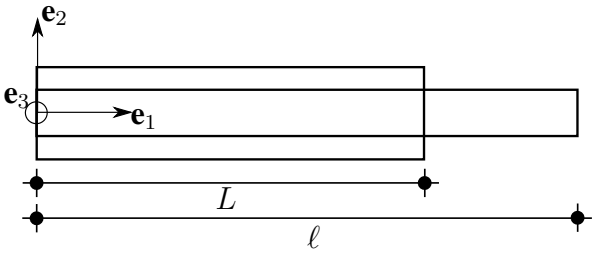

Figura 1: Barra con deformación $\mathcal{X}$.

En este caso se tiene que:

$$\mathbf{F} = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \beta \end{pmatrix}. \tag{10}$$

Para el caso de Curnier se tiene

$$\mathbf{S} = \begin{pmatrix} S_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \; S_1 = \lambda\left(\alpha\beta^2 - 1\right) + \mu\left(\alpha^{-2} - 1\right) \tag{11}$$

donde $\alpha = \frac{\ell}{L}$, $\beta^2 = \frac{\mu - \alpha\lambda + \sqrt{(\mu-\alpha\lambda)^2 + 4\,\mu\,\lambda}}{2\,\mu}$.

Esta familia de casos tiene la particularidad que pueden ser resueltos de manera analítica. Este hecho fue utilizado para validar, desde el punto de vista numérico, los resultados obtenidos.

De este problema se definieron cuatro instancias, $barra_S$, $barra_M$, $barra_L$ y $barra_{XL}$, las cuales se diferencian en la cantidad de elementos utilizados para la discretización del sólido. En particular, se utilizan alrededor de 22 mil elementos para $barra_S$, lo que lleva a un sistema lineal de dimensión 13011 y densidad 0,29 %. Para $barra_M$ se utilizaron aproximadamente 147 mil elementos obteniendo un sistema de dimensión 80182 y densidad 0,05 %. Finalmente para $barra_L$ se utilizaron 476 mil elementos y para $barra_{XL}$ 655 mil elementos aproximadamente, lo que resulto en sistemas de dimensión 249877 con densidad 0,017 % para $barra_L$ y de dimensión 341232 con densidad 0,0126 % para $barra_{XL}$.

3.1.2. Cilindro con presión interna

En Haussy y Ganghoffer (2005) se presenta una solución analítica para el problema del cilindro con presión interna y se observa la dificultad que conlleva obtener una solución analítica general. Por lo anterior es que este ejemplo se resolverá sólo de forma numérica, ver Figura 2.

Para este caso se confeccionaron 2 cilindros, $vaso_S$ y $vaso_M$, discretizados en aproximadamente 7 mil elementos y 50 mil elementos respectivamente. Estas características resultaron en una matriz de dimensión 2296 con una densidad de 0.88 % para $vaso_S$ y otra de dimensión 17558 con densidad 0,158 % para $vaso_M$.

3.1.3. Cubo con inclusión

En la Figura 3 se considera un sólido ocupando una región cúbica con una esfera en el centro constituida por un material de mayor rigidez. En este ejemplo se resuelve una geometría similar al presentado en Banerjee et al. (2013) donde se estudian también estrategias computacionales para hacer más eficiente la identificación de parámetros mecánicos en problemas de muchas variables.

En esta situación se tomaron dos casos de prueba, de nombre $cuboinclu$ y $cuboinclu_{plus}$. El primero está compuesto por 75 mil elementos, resultando en una matriz de dimensión 42338 con una densidad de 0.098 %. Para el segundo caso, se utilizaron cerca de 290 mil elementos, lo que llevo a un sistema lineal de dimensión 155053 con una densidad de 0,028 %.

3.2. Plataforma de hardware

Las ejecuciones se realizaron sobre un equipo con un procesador Intel Core i7-4770 de 4 núcleos y 16GB de RAM. Además el equipo cuenta con una tarjeta NVIDIA K40 de 2880 núcleos a 745 MHz y 12 GB de memoria.

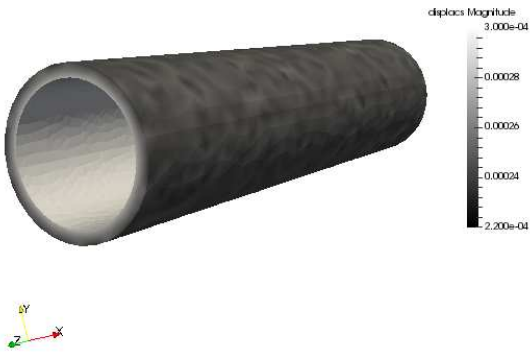

Figura 2: Cilindro con presión interna.

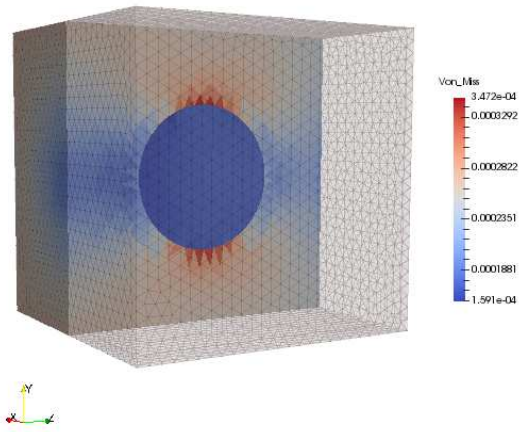

Figura 3: Distribución de tensiones de Von Mises de caso de prueba de cubo con inclusión esférica de mayor rigidez.

Se utilizaron las bibliotecas OpenBLAS versión 0.2.19, INTEL MKL y el compilador para C++ de la suite Intel composer XE 2013 SP1, METIS en su versión 5.1.0, mpich 3.1.3 y el Compilador de GNU para C y FORTRAN en su versión 4.9.2.

En el caso de los solvers en CPU, se utilizó la versión de MUMPS 5.0.2, PARDISO de la suite Intel composer XE 2013 SP1, PETSc en su versión 3.7.5 y la rutina HSL MA57 en su versión 3.1.0. Para el solver en GPU se utilizó la biblioteca CUDA en su versión 6.5.

3.3. Evaluación de MecSolENL

Para evaluar MecSolENL se realizaron simulaciones con diferentes casos de prueba midiendo los tiempos de ejecución que implica el uso de la herramienta. (Por una cuestión de tiempos de ejecución se evaluaron solo los casos chicos o medianos.) Además, se evaluó el tiempo in-

sumido por los solvers de ecuaciones lineales, los cuales utilizan la rutina HSL MA57. Estos valores se resumen en la Tabla 1, incluyendo también el porcentaje que representa el tiempo de ejecución del solver frente al tiempo total.

Tabla 1: Tiempos de ejecución (en segundos) para los distintos casos de prueba, utilizando la herramienta original.

Caso	*Tiempo solver*	*Tiempo total*	*Porcentaje*
$vaso_S$	0,10	3,60	2,87
$barra_S$	16,32	38,78	42,08
$cuboInclu$	142,56	186,03	76,63
$barra_M$	1059,87	1211,60	87,48

Considerando los tiempos de ejecución resumidos en la Tabla 1 se puede decir que, en los casos de dimensiones pequeñas, el tiempo de ejecución del solver no es importante. Sin embargo, a medida que se incrementa la dimensión del problema tratado, el tiempo de resolución de los sistemas lineales aumenta considerablemente. Notar que en los casos de dimensiones intermedias, la proporción de tiempo de ejecución debida a los solvers lineales supera levemente el 40 % del tiempo total, aumentando hasta el 88 % en el caso evaluado más grande. Este comportamiento es lógico si se considera que la resolución de los sistemas lineales tiene un orden computacional mayor al lineal, mientras que las otras etapas del método (lectura de archivo, inicialización de estructuras de datos, actualización de matrices en el método de Newton-Raphson, etc.) o bien se ejecutan en una sola ocasión o son de menor orden.

Entonces, basados en estos resultados, se puede afirmar que para acelerar MecSolENL es necesario mejorar el desempeño de los solvers lineales (es decir, la rutina HSL MA57), al menos cuando se tratan de resolver problemas con dimensiones superiores a un umbral, los cuales son generalmente aquellos que presentan restricciones reales para su abordaje.

4. RESOLUCIÓN DE SISTEMAS LINEALES

Como se vió en la sección anterior, los solver de sistemas de ecuaciones lineales son la etapa más costosa desde el punto de vista computacional de la herramienta de simulación de mecánica de sólidos. Considerando dicha situación en esta sección se profundiza en los conceptos básicos de métodos de resolución de sistemas lineales dispersos.

Dentro del álgebra lineal numérica existen dos grandes familias de métodos para resolver sistemas lineales Datta (2010); Golub y Van Loan (2012); Higham (2002):

- Métodos directos.
- Métodos iterativos.

Cuando se desea resolver sistemas lineales con matrices dispersas, los métodos directos siguen, en general, las siguientes etapas:

* Ordenamiento: esta etapa juega un papel fundamental a la hora de evitar la aparición de nuevos elementos no nulos cuando se factoriza la matriz (problema denominado *fill-in*), lo cual impacta directamente en el desempeño de la resolución. Encontrar el ordenamiento óptimo es un problema NP-difícil Yannakakis (1981), por esta razón se utilizan diferentes heurísticas para definir el ordenamiento (más adelante se profundiza en estas técnicas).

* Factorización simbólica: predice la estructura de las matrices intermedias y reserva la memoria necesaria para la factorización.
* Factorización numérica: transforma la matriz del sistema en las matrices factorizadas (en general triangulares).
* Sustitución: se realiza la sustitución de los sistemas (triangulares) obtenidos.

La factorización simbólica, en muchas ocasiones se realiza al mismo tiempo que el Ordenamiento, y en general, se basa en tratar el patrón de la matriz (las posiciones de la matriz con coeficientes distinto de 0) como un grafo. Para la factorización numérica se dispone de diferentes estrategias, dependiendo del tipo de acceso a los datos. En general, se busca (según el ordenamiento) realizar factorizaciones parciales en forma independiente e impactar esas factorizaciones en la matriz global. Para esto, se puede usar por ejemplo el complemento de Shur para definir el procedimiento de actualización de la matriz global. Estas técnicas permiten definir diferentes métodos, entre los más difundidos se encuentran frontal, multifrontal y supernodo Davis (2006). Otro concepto importante en la factorización numérica es intentar agrandar las dimensiones de las matrices parciales para poder explotar el uso de cómputos basados en BLAS-3 Golub y Van Loan (2012). Adicionalmente, otro punto relevante en todas las etapas de los métodos directos (y en los métodos iterativos también) es el tipo de almacenamiento disperso utilizado por las matrices.

Por otro lado, los métodos iterativos se basan en ir aproximando la solución a partir de un valor inicial y mediante un refinamiento dado. Estos métodos no aseguran la precisión de la solución obtenida ni la cantidad de pasos que el algoritmo requiere para hallarla, sino que es una solución de compromiso. Los principales métodos iterativos actualmente son los basados en los sub-espacios de Krylov Saad (2003). En especial el método del gradiente conjugado para sistemas lineales simétricos y definidos positivos y las diferentes variantes para otro tipo de sistemas (simétricos indefinidos o generales).

Con el objetivo de disminuir la cantidad de pasos necesaria para aproximar la solución, y en muchos casos lograr la convergencia, se pueden utilizar técnicas de precondicionado Benzi (2002). Estas técnicas se basan en llevar el sistema lineal original a otro sistema equivalente pero de resolución más rápida.

Una característica interesante de estos métodos es que son de gran utilidad para resolver sistemas lineales dispersos. Notar que las iteraciones se basan, en general, en la multiplicación matriz-vector, operación conocida como *spmv* (del inglés SParse Matriz Vector) en el caso disperso. Esto implica que durante la iteración no existe el problema de aparición de elementos no nulos.

4.1. Ordenamientos

Uno de los principales problemas a la hora resolver un sistema lineal disperso utilizando un método directo es la aparición de elementos no nulos cuando se realiza la factorización numérica, lo cual impacta de forma directa en el desempeño y el uso de la memoria del método de resolución. Por esta razón, se estila utilizar técnicas de ordenamiento que buscan, mediante permutaciones de filas y columnas de la matriz del sistema, disminuir la cantidad de elementos que aparecen como consecuencia de aplicar dicha factorización.

El objetivo de las técnicas de ordenamiento se puede ver como un problema de optimización, en particular de minimización (minimizar la aparición de elementos no nulos). La resolución de dicho problema es NP-duro y esto hace que sea prohibitivo resolverlo en forma exacta cuando

la dimensión de la matriz crece. Por esta razón, se utilizan técnicas heurísticas al momento de implementar estos métodos. Existen dentro de estas metodologías tres grupos:

- Globales: se basan en ir subdividiendo la matriz hasta llegar a una porción de resolución trivial, para luego ensamblar cada una de ellas. Un ejemplo de estos son el algoritmo Cuthill-McKee Cuthill y McKee (1969) (o la variante inversa, que es la más difundida, Reverse Cuthill-McKee Chan y George (1980)), donde se propone disminuir el ancho de banda de la matriz, o la disección recursiva (*Nested Dissection*) George (1973) donde se busca reordenar la matriz a partir de un algoritmo recusivo.

- Locales: se busca ir ordenando la matriz un pivot a la vez, utilizando información local, luego se impacta en la submatriz y se continua el ordenamiento con los pivotes restantes. Dentro de esta familia se encuentra el método de deficiencia mínima (*Minimum Deficiency*), el cual busca disminuir la cantidad de aristas necesarias para convertir el conjunto de vertices adyacentes a otro en un clique. Otro método es el de grado mínimo (*Minimum Degree*) George y Liu (1989), donde se va elegiendo el pivot con menor grado, o sea que posee menos aristas incidentes.

- Híbridas: utilizan una fusión de técnicas globales y locales. Un ejemplo sería utilizar disección recursiva para realizar un particionamiento grueso de la matriz y para las submatrices de tamaño pequeño utilizar grado mínimo.

En los trabajos de J. Lewis Lewis (1988) y George et al. George et al. (2012) se puede encontrar más información sobre las estrategias de ordenamiento.

4.2. Precondicionadores

En forma conjunta a los métodos iterativos, se pueden utilizar precondicionadores sobre el sistema lineal a trabajar, con el objetivo de disminuir la cantidad de iteraciones necesarias para aproximar la solución a una precisión deseable. Para ello, se premultiplica o posmultiplica el sistema lineal por una matriz M^{-1}, transformando así el sistema lineal en otro equivalente pero con condiciones espectrales más favorables.[1] Existen dentro de estos tipos de precondicionadores dos grandes familias:

- Explícitos: se enfocan en construir la matriz M^{-1} tal que esta se aproxime a la inversa de la matriz del sistema lineal.

- Implícitos: se basan en construir una matriz M de forma que ésta se aproxime a la matriz del sistema lineal y permita la resolución fácil de un sistema lineal (para evitar así el cálculo de la inversa).

Dentro de los precondicionadores implícitos está el precondicionador de Jacobi, donde la matriz M es la diagonal de la matriz del sistema. Otra forma de este tipo de precondicionadores son aquellos obtenidos por factorizaciones incompletas, donde se busca utilizar una versión de la matriz del sistema construida a partir de una factorizacion parcial de la misma. Dentro de estos métodos se encuentran ILU e ICC que emplean una factorización incompleta LU e incompleta de Cholesky como matriz M respectivamente.

[1] Notar que si M^{-1} es igual a A^{-1} la resolución del sistema es trivial.

4.3. Herramientas estudiadas

Dado que el objetivo de este trabajo es acelerar un simulador de mecánica de sólidos limitado en tiempo por la resolución de sistemas lineales dispersos de grandes dimensiones, se buscó utilizar ambas familias de solvers (directos e iterativos). En particular, se decidió utilizar tres implementaciones de solvers de ALN incluidos en distintas bibliotecas altamente difundidas en la comunidad:

1. MUMPS: MUltifrontal Massively Parallel sparse direct Solver Amestoy et al. (2000).
2. PARDISO: Parallel Direct Sparse Solver PARDISO (2013).
3. PETSc: Portable, Extensible Toolkit for Scientific Computation Balay et al. (2012).

Los primeros dos solvers son de la familia de los métodos directos mientras que el tercero es del tipo iterativo. En particular, el solver presente en MUMPS es del tipo multifrontal que utiliza tanto la descompisición LU como de Cholesky (según el tipo de sistema lineal), mientras que el solver presente en PARDISO utiliza un método de super-nodo también para ambas descomposiciones. En el caso de PETSc, la resolución itertativa utilizada para los sistemas simétricos y definidos positivos es el método del gradiente conjugado. Sin embargo, incluye una larga lista de otros solvers, como GMRES, Bi-CG-stab o transpose free QMR, para otros tipos de sistemas lineales.

Como se dijo anteriormente, las técnicas de ordenamiento y precondicionado son escenciales para los métodos directos e iterativos respectivamente, por lo cual a continuación se describen las opciones disponibles en este sentido en las herramientas abordadas.

En el caso del solver de MUMPS se dispone de los ordenamientos:

- AMD (Approximate Minimum Degree): método local basado en grado mínimo (minimum degree), que en lugar de utilizar los grados de los vértices, estima un límite superior del grado, el cual es más sencillo de calcular Amestoy et al. (1996).
- AMF (Approximate Minimum Fill-in): método local basado en deficiencia mínima (minimum deficiency), que busca aproximarse al fill-in óptimo utilizando un algotitmo basado en KTS de orden polinómico Natanzon et al. (2000).
- QAMD (AMD with automatic quasi-dense row detection): método local basado AMD, con la particularidad que considera y evita problemas ocacionados por filas parcialmente densas.
- METIS: método global basado en disección recursiva (nested dissection) que utiliza técnicas de partición de grafos.
- PORD: método híbrido basado en AMD y disección recursiva (nested dissection).

Para el caso de PARDISO, los métodos de ordenamiento son:

- AMD
- METIS
- PARMETIS: implemetación paralela del método METIS.

Para el solver PETSc, se estudiaron los siguientes precondicionadores:

- Jacobi: método implícito que utiliza la diagonal de la matriz del sistema lineal.
- ICC de nivel 0: método implícito que utiliza una descomposición parcial de Cholesky sin ningun nivel de llenado. Es decir, realiza la factorización de Cholesky de la matriz pero no permite la aparición de elementos no nulos.
- ICC de nivel 1: método implícito que utiliza una descomposición parcial de Cholesky con un nivel de llenado. Es decir, permite la aparición de elementos no nulos cuando se deben a operaciones con elementos que no eran nulos en la matriz original.

4.4. Uso de plataformas masivamente paralelas

Además de las bibliotecas mencionadas en el apartado anterior, en este trabajo evaluamos en forma preliminar el uso de procesadores masivamente paralelos, específicamente GPUs, para acelerar la herramienta MecSolENL. Este experimento resulta de interés para evaluar el desempeño de estas arquitecturas y en particular observar las instancias donde éstas pueden ser de utilidad para la aplicación de mecánica de sólidos.

En este sentido, usamos una implementación propia del método del gradiente conjugado precondicionado (PCG por su sigla en inglés), siendo ILU(0) el precundicionador seleccionado en esta instancia. La implementación utilizada (por más detalles ver Dufrechou et al. (2013)) utiliza la GPU para realizar las operaciones más costosas computacionalmente, sacando partido del uso de las bibliotecas CUSPARSE y CUBLAS de NVIDIA y siguiendo la propuesta de Naumov (2011).

Es de importancia mencionar que el alcance de la evaluación de esta clase de dispositivos es meramente exploratorio y se aborda únicamente el paradigma de los métodos iterativos sobre estas plataformas, debido a que las soluciones directas para el caso disperso sobre GPUs no han tenido un amplio desarrollo.

5. EVALUACIÓN EXPERIMENTAL

Para la evaluación experimental de la propuesta se utilizan los mismos casos de prueba y plataforma de hardware ya descriptos en la Sección 3.

En primera instancia, es relevante mencionar que los diferentes solvers se validaron desde un punto de vista numérico (calidad de simulación) y que los resultados obtenidos no difieren más allá de las diferencias explicables por el uso de la representación en punto flotante. Recordar que solamente el cambio en el orden de ejecución, y por lo tanto en el orden en que se computan ciertas operaciones (por ejemplo al aplicar técnicas de paralelismo), puede introducir pequeñas diferencias numéricas.

5.1. Resultados Experimentales

Antes de decidir cual de los solvers ofrecidos por las bibliotecas en CPU es el que mejor se comporta para este tipo de problemas, es necesario alcanzar la mejor configuración para cada una de las herramientas. En este sentido, el primer estudio realizado consiste en comparar los distintos ordenamientos o precondicionadores disponibles en los paquetes de métodos directos o iterativos respectivamente. Para llevar a cabo este experimento, se evaluaron los métodos descriptos en la Sección 4.3 utilizando el primer sistema lineal del caso de prueba $cuboinclu$. Los

resultados se presentan en la Tabla 2, siendo estos el promedio de 10 ejecuciones independientes.

Tabla 2: Tiempo de ejecución (en segundos) que requiere cada solver para resolver un sistema lineal del caso $cuboinclu$, variando el método de ordenamiento o precondicionado.

(a) MUMPS

Ordenamiento	***Tiempo***
AMD	2,18
AMF	1,46
PORD	1,25
METIS	1,21
QAMD	2,09

(b) PARDISO

Ordenamiento	***Tiempo***
AMD	1,51
METIS	0,73
PARMETIS	0,70

(c) PETSc

Ordenamiento	***Tiempo***
JACOBI	7,74
ICC(0)	3,50
ICC(1)	3,34

Como se puede observar en la Tabla 2, la mejor configuración para el solver MUMPS es el ordenamiento METIS, aunque es importante mencionar que el ordenamieto PORD presenta resultados muy similares. Por otro lado, para el solver PARDISO el mejor ordenamiento es PARMETIS, sin embargo no presenta una predominancia significativa contra METIS, el cual presenta el segundo mejor tiempo. Para el solver iterativo PETSc, el mejor precondicionador es ICC(1) pero nuevamente la segunda opción en cuanto a tiempos de ejecución, ICC(0), alcanza un desempeño muy similar. Si bien las bondades ofrecidas por las estrategias de ordenamiento y precondicionado pueden variar al cambiar la dimensión o patrón de los sistemas lineales resueltos (y en nuestro caso se utilizó únicamente un caso de prueba por cuestiones de tiempos), en el contexto del tipo de problemas abordados, estimamos que estos resultados son extrapolables a otros sistemas lineales. Además, considerando este planteo, estimamos que si bien todos los casos presentan dos opciones muy próximas, es correcto elegir la mejor versión vista en este estudio, ya que las diferencias pueden ser significativas en otros problemas.

Luego de obtener la mejor configuración para cada uno de los solvers incluidos en las bibliotecas, se procedió a evaluar los mismos para cada uno de los casos de pruebas antes descriptos. Para ello, se tomó el tiempo de resolución del primer sistema lineal de cada uno de los casos de prueba, utilizando la mejor configuración de cada solver. El promedio del tiempo de 10 ejecuciones independientes para la resolución de un sistema lineal se ofrece en la Tabla 3.

Tabla 3: Tiempos de ejecución (en segundos) para los distintos casos de prueba, utilizando la mejor configuración para cada solver presente en las bibliotecas de CPU.

Caso	***MA57***	***MUMPS***	***PARDISO***	***PETSC***
$vaso_S$	0,01	0,01	0,01	0,03
$vaso_M$	0,09	0,14	0,04	0,20
$barra_S$	0,38	0,23	0,10	0,77
$cuboInclu$	5,08	1,20	0,69	3,33
$barra_M$	24,41	3,47	2,33	9,73
$cuboInclu_{plus}$	145,53	10,69	8,77	28,07
$barra_L$	305,36	22,85	21,45	47,55
$barra_{XL}$	944,27	41,46	45,50	73,76

Como se puede notar en la Tabla 3, cuando el caso de prueba es de dimensiones moderadas, la mejor opción es el solver MA57. Sin embargo, a medida que las dimensiones del problema aumentan, es notoria la mejora obtenida por todos los solvers evaluados y en especial que, para este estudio, los solvers de tipo directo son mas adecuados que el solver iterativo. Entre los nuevos métodos, PARDISO logra superar en tiempos de ejecución a los demás solvers a excepción del caso de mayor porte, $barra_{XL}$, donde MUMPS muestra una leve ventaja. Si se observa la tabla con detenimiento, se puede observar que este resultado se debe a que MUMPS presenta, para los casos estudiados, una mejor escalabilidad frente a la dimensión del problema. Debido a las conclusiones obtenidas de esta ejecución, y con el fin de tomar la mejor solución para cada caso de estudio, se utilizará en adelante como mejor opción la familia de los solvers directos, utilizando según convenga el solver PARDISO o MUMPS.

Una vez establecido los mejores tiempos ofrecidos por las bibliotecas para resolver un sistema lineal, se procedió a estudiar su desempeño como parte del simulador de mecánica de sólidos en su totalidad. Es decir, se utiliza para resolver todos los sistemas lineales presentes en el modelo de mecánica de sólidos abordado. Además, se comparan dichos resultados con el desempeño alcanzado por la implementación en GPU (PCG_{GPU}). En la Tabla 4 se muestran los tiempos de ejecución para resolver todos los sistemas lineales, para los distintos casos de prueba. Además, se incluye en el mismo estudio los tiempos de ejecución del presente experimento para la contraparte original (basado en el solver MA57), con el fin de evaluar la mejora obtenida por los nuevos solvers. También se ofrece la aceleración obtenida, calculada como el ratio entre el tiempo original y el tiempo de la propuesta, para los solvers directos en CPU y para el solver en GPU.

Tabla 4: Tiempos de ejecución (en segundos) y aceleración obtenidos para la resolución de todos los sistemas lineales utilizados por MA57, solvers directos y PCG$_{GPU}$, para los distintos casos de prueba trabajados.

Casos	***MA57***	***Solver de Bibliotecas***	***Aceleración***	***GPU***	***Aceleración***
vaso_S	0,10	0,33	0,32	2,98	0,03
vaso_M	1,31	0,61	2,14	2,24	0,58
barra_S	16,52	3,79	4,36	24,13	0,68
cuboinclu	142,63	18,64	7,65	191,91	0,74
barra_M	1079,14	90,45	11,93	86,18	12,52
cuboinclu_plus	3467,88	218,46	15,87	276,41	12,55
barra_L	13259,46	988,80	13,41	277,19	47,83
barra_XL	39874,31	1773,16	22,49	375,17	106,28

Observando la Tabla 4 se puede percibir que, si bien los solvers directos (PARDISO-MUMPS) no obtiene mejoras para el caso de dimensión más pequeña ($vaso_S$), para el caso $vaso_M$ y $barra_S$ el mismo ya alcanza una disminución en el tiempo de ejecución de más de dos veces la del MA57. A partir de estos casos, el solver compuesto por ambos métodos directos ofrece mejoras de 7 veces para el caso $cuboinclu$, logrando ser hasta 22 veces mejor para la $barra_{XL}$ aproximadamente. Es importante hacer notar entonces que, a medida que la matriz de prueba aumenta su tamaño, también lo hace la diferencia en tiempo de ejecución entre ambos métodos, independientemente del caso de prueba utilizado. Esta propiedad, de escalabilidad, resulta de sumo interés, ya que utilizar matrices de dimensiones mayores permite modelar cuerpos más complejos o modelar cuerpos simples con mayor precisión. Por otro lado, cuando se compara la variante en GPU (PCG$_{GPU}$), se puede ver que las mejoras comienzan en tamaños de problema de mayor porte, en particular, en el caso $barra_M$ donde se alcanza un incremento de 12 $\times$ frente al solver MA57. En los siguientes casos de mayor dimensión, la solución en GPU presenta mejoras sustanciales, logrando superar los tiempos de ejecución por más de 100$\times$ para $barra_{XL}$. Este comportamiento se debe a que los casos pequeños no logran aprovechar la característica masivamente paralela de la arquitectura, lo que no compensa los tiempos de envío de los datos para su procesamiento, mientras que el gran volumen de los restantes casos explota esta propiedad satisfactoriamente. Es importante notar además que, si bien las mejoras de la solución en GPU son alentadoras, en los casos de tamaño intermedio los solvers directos son mejores que esta propuesta y solamente en los casos $barra_L$ y $barra_{XL}$ ésta logra sobrepasarlos, alcanzando una mejora de 3.5$\times$ y casi 5 veces respectivamente. Esto refuerza el concepto mencionado que es necesario un gran volumen de datos y cómputo para lograr mejoras sustanciales al usar tarjetas gráficas para la resolución de los presentes sistemas lineales.

Finalmente, y con el objetivo de analizar el impacto de utilizar estos solvers sobre la herramienta de simulación de mecánica de sólidos en su conjunto, se evaluó el tiempo de ejecución total de la herramienta utilizando el mejor solver directo, la versión en GPU y su comparación con la versión original.

Tabla 5: Tiempos de ejecución (en segundos) y aceleración obtenidos para el solver de mecánica de sólidos para los distintos casos de prueba, utilizando los solvers lineales MA57, métodos directos y PCG$_{GPU}$

Casos	*MA57*	*Solver de Bibliotecas*	*Aceleración*	*GPU*	*Aceleración*
vaso_S	3,50	4,81	0,73	6,54	0,54
vaso_M	66,23	66,47	1,00	69,71	0,95
barra_S	40,24	30,62	1,31	55,38	0,73
cuboinclu	186,67	68,59	2,72	267,43	0,70
barra_M	1224,60	242,81	5,04	294,85	4,15
cuboinclu_plus	3646,79	384,11	9,49	514,61	7,09
barra_L	13764,07	1496,67	9,20	972,46	14,15
barra_XL	41882,52	2465,65	16,99	1375,17	30,46

En la Tabla 5 se puede notar que las mejoras obtenidas a nivel del solver lineal impactan directamente en el desempeño del modelo en su totalidad. De esta forma, como se notó en el estudio anterior, si bien para los casos de dimensiones reducidas los tiempos de ejecucion de la versión original son los menores, a partir del caso $barra_S$ se empiezan a observar mejoras de 1.3 veces a favor de las variantes directas (PARDISO en estos casos). Por otro lado, para los casos de dimensiones superiores a éste, se observa mejoras aun más significativas, alcanzando tiempos de casi 3 veces mejores para el caso $cuboinclu$ y hasta 16 veces para el caso $barra_{XL}$.

Para la propuesta en GPU, hay que hacer notar que presenta ciertos overheads relacionados con la construcción de las estructuras necesarias para ejecutar los casos en dicha arquitectura (principalmente la construcción de la matriz), lo que disminuye su desempeño cuando es evaluada en la herramienta completa. A pesar de ésto, la versión PCG$_{GPU}$ logro mejoras de 1,5$\times$ y 1,8$\times$ para los casos $barra_L$ y $barra_{XL}$, respectivamente, frente a la solución que utiliza los solvers directos. Estos desempeños se traducen en 14 veces y 30 veces mejor respectivamente, si se los compara contra la solución original.

6. CONCLUSIONES Y TRABAJO FUTURO

En el trabajo se abordó el estudio y mejora del desempeño computacional de una herramienta para la simulación de mecánida de sólidos, MecSolENL. En primera instancia se identificaron los cuellos de botella de la herramienta, desde el punto de vista computacional. Se determinó que, en problemas de talla mediana o grande, la resolución de sistemas lineales de ecuaciones dispersos es la principal limitante. En este contexto se estudiaron y evaluaron diferentes métodos para la resolución de dichos sistemas, así como estrategias para optimizar el desempeño los métodos (es decir, técnicas de ordenamiento y precondicionado).

De esta evaluación se desprendió que, en los casos chicos la versión original ofrece un desempeño razonable, mientras que para los casos de dimensiones media los solvers directos (PARDISO-MUMPS) alcanzan mayores reducciones en el tiempo de ejecución, por ejemplo en el caso $cuboinclu$ la reducción de tiempos totales cuando se lo compara con la variante original es de 7,65$\times$. Por último, en los casos que presentan dimensiones realmente importantes, la GPU alcanza rendimientos superiores a las otras variantes. Dicha implementación, para el caso $barra_{XL}$, consigue una aceleración de 106,28 $\times$ y 4,73$\times$ cuando se lo compara con la versión original y el mejor método directo respectivamente. Cuando se aplican estas mejoras al solver MecSolENL, se obtinen disminuciones en el tiempo total de ejecución de la herramienta de

hasta 30 veces frente a la implementación que utiliza el solver MA57 y de casi 1,8× frente a la versión que utiliza el mejor solver dirécto. Además, las mejoras obtenidas considerando todos los solvers estudiados, crecen al aumentar la dimensión de los problemas abordados, siendo esta una propiedad deseable cuando se trabaja con herramientas computacionales.

Diferentes aspectos que no fueron cubiertos en el presente trabajo merecen un abordaje en mayor profundidad. Entre los más destacados, parece importante completar el estudio computacional de la herramienta, incluyendo una evaluación desde el punto de vista de uso de memoria y/o consumo energético, así como disminuir los overhead en el caso de la variante en GPU.

En otro sentido, y considerando los resultados auspiciosos que se obtuvieron parece interesante utilizar la nueva variante de la herramienta en casos de mecánica de sólidos que requieran mayor poder de cómputo, como son los casos directamente relacionados con la medicina.

REFERENCIAS

Amestoy P.R., Davis T.A., y Duff I.S. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

Amestoy P.R., Duff I.S., L'Excellent J.Y., y Koster J. Mumps: a general purpose distributed memory sparse solver. En *International Workshop on Applied Parallel Computing*, páginas 121–130. Springer, 2000.

Balay S., Brown J., Buschelman K., Eijkhout V., Gropp W., Kaushik D., Knepley M., McInnes L.C., Smith B., y Zhang H. Petsc users manual revision 3.3. *Computer Science Division, Argonne National Laboratory, Argonne, IL*, 2012.

Banerjee B., Walsh T.F., Aquino W., y Bonnet M. Large scale parameter estimation problems in frequency-domain elastodynamics using an error in constitutive equation functional. *Computer Methods in Applied Mechanics and Engineering*, 253:60–72, 2013. ISSN 00457825. doi:10.1016/j.cma.2012.08.023.

Belytschko T., Liu W.K., y Moran B. *Nonlinear finite elements for continua and structures*. Wiley, 1st edición, 2001. ISBN 978-0471-98773-4.

Benzi M. Preconditioning techniques for large linear systems: a survey. *Journal of computational Physics*, 182(2):418–477, 2002.

Brinkhues S., Klawonn A., Rheinbach O., y Schröder J. Augmented Lagrange methods for quasi-incompressible materials-Applications to soft biological tissue. *International Journal for Numerical Methods in Biomedical Engineering*, 29(3):332–350, 2013. ISSN 20407939. doi:10.1002/cnm.2504.

Chan W.M. y George A. A linear time implementation of the reverse cuthill-mckee algorithm. *BIT Numerical Mathematics*, 20(1):8–14, 1980.

Curnier A. *Computational Methods in Solid Mechanics*, volumen 29 de *Solid Mechanics and Its Applications*. Springer Netherlands, Dordrecht, 1994. ISBN 978-94-010-4486-8. doi: 10.1007/978-94-011-1112-6.

Cuthill E. y McKee J. Reducing the bandwidth of sparse symmetric matrices. En *Proceedings of the 1969 24th National Conference*, ACM '69, páginas 157–172. ACM, New York, NY, USA, 1969.

Datta B.N. *Numerical linear algebra and applications*. Siam, 2010.

Davis T.A. *Direct methods for sparse linear systems*. SIAM, 2006.

Dufrechou E., Ezzatti P., Quintana-Ortí E.S., y Remón A. Accelerating the lyapack library using gpus. *The Journal of Supercomputing*, 65(3):1114–1124, 2013. ISSN 1573-0484. doi: 10.1007/s11227-013-0889-8.

George A. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

George A., Gilbert J.R., y Liu J.W. *Graph theory and sparse matrix computation*, volumen 56. Springer Science & Business Media, 2012.

George A. y Liu J.W. The evolution of the minimum degree ordering algorithm. *Siam review*, 31(1):1–19, 1989.

Goenezen S., Dord J.F., Sink Z., Barbone P.E., Jiang J., Hall T.J., y Oberai A.a. Linear and Nonlinear Elastic Modulus Imaging: An Application to Breast Cancer Diagnosis. *IEEE Transactions on Medical Imaging*, 31(8):1628–1637, 2012. ISSN 0278-0062. doi: 10.1109/TMI.2012.2201497.

Golub G.H. y Van Loan C.F. *Matrix computations*, volumen 3. JHU Press, 2012.

Haussy B. y Ganghoffer J.F. Instability analysis in pressurized transversely isotropic Saint-Venant-Kirchhoff and neo-Hookean cylindrical thick shells. *Archive of Applied Mechanics*, 74(9):600–617, 2005. ISSN 09391533. doi:10.1007/s00419-005-0376-7.

Higham N.J. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edición, 2002. ISBN 0898715210.

Holzapfel G. *Nonlinear solid mechanics: A continuum approach for engineering*, volumen First Edit. Wiley, 2000a. ISBN 0471823198. doi:10.1023/A:1020843529530.

Holzapfel G.A. Biomechanics of soft tissue. *Handbook of Material Behavior*, 1(7):1–15, 2000b. ISSN 00219290. doi:10.1109/CA.1999.781200.

Holzapfel G.A. y Ogden R.W., editores. *Biomechanics of Soft Tissue in Cardiovascular Systems*. Springer Vienna, Vienna, 2003. ISBN 978-3-211-00455-5. doi:10.1007/978-3-7091-2736-0.

Hughes T.J.R. *The finite element method : linear static and dynamic finite element analysis*. Dover Publications, 2000. ISBN 0486411818.

Lewis J.G. Ordering methods for sparse matrices and vector computers. Informe Técnico, DTIC Document, 1988.

Moireau P., Xiao N., Astorino M., Figueroa C.A., Chapelle D., Taylor C.A., y Gerbeau J.F. External tissue support and fluid-structure simulation in blood flows. *Biomechanics and Modeling in Mechanobiology*, 11(1-2):1–18, 2012. ISSN 16177959. doi:10.1007/s10237-011-0289-z.

Natanzon A., Shamir R., y Sharan R. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM Journal on Computing*, 30(4):1067–1079, 2000.

Naumov M. Incomplete-lu and cholesky preconditioned iterative methods using cusparse and cublas. *Nvidia white paper*, 2011.

PARDISO I.M. Parallel direct sparse solver interface. 2013.

Pavarino L., Scacchi S., y Zampini S. Newton–Krylov-BDDC solvers for nonlinear cardiac mechanics. *Computer Methods in Applied Mechanics and Engineering*, 295:562–580, 2015. ISSN 00457825. doi:10.1016/j.cma.2015.07.009.

Pérez Zerpa J.M. y Canelas A. Efficient formulations of the material identification problem using full-field measurements. *Computational Mechanics*, 58(2):235–255, 2016. ISSN 1432-0924. doi:10.1007/s00466-016-1291-1.

Pérez Zerpa J.M. *Resolución de Problemas Inversos en Mecánica de Sólidos con aplicación al modelado e identificación de propiedades mecánicas de tejidos biológicos*. Tesis de Doctorado, Universidad de la República, Uruguay, 2016.

Saad Y. *Iterative methods for sparse linear systems*. SIAM, 2003.

Valdez-Jasso D., Bia D., Zócalo Y., Armentano R.L., Haider M.A., y Olufsen M.S. Linear and nonlinear viscoelastic modeling of aorta and carotid pressure-area dynamics under in vivo

and ex vivo conditions. *Annals of Biomedical Engineering*, 39(5):1438–1456, 2011. ISSN 00906964. doi:10.1007/s10439-010-0236-7.

WHO. Enfermedades cardiovasculares. Informe Técnico, World Health Organization Media Center, 2015.

Yannakakis M. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.