UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Aplicación de la computación paralela mediante MPI en la formulación finita

Autor: Adrián de Pablo Sánchez
Tutor(es): Dr. José Miguel Monzón Verona
Dr. Leopoldo Simón Rodríguez
Fecha: Junio de 2017

IUMA
t +34 928 451 086
f +34 928 451 083
iuma@iuma.ulpgc.es
www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Aplicación de la computación paralela mediante MPI en la formulación finita

HOJA DE FIRMAS

Alumno/a:	Adrián de Pablo Sánchez	Fdo.:
Tutor/a:	Dr. José Miguel Monzón Verona	Fdo.:
Tutor/a:	Dr. Leopoldo Simón Rodríguez	Fdo.:

Fecha: Junio de 2017

IUMA
t +34 928 451 086 | iuma@iuma.ulpgc.es | Campus Universitario de Tafira
f +34 928 451 083 | www.iuma.ulpgc.es | 35017 Las Palmas de Gran Canaria

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Aplicación de la computación paralela mediante MPI en la formulación finita

HOJA DE EVALUACIÓN

Calificación: .....

Presidente Fdo.:

Secretario Fdo.:

Vocal Fdo.:

Fecha: Junio de 2017

IUMA
t +34 928 451 086
f +34 928 451 083
iuma@iuma.ulpgc.es
www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Acrónimos

CGS	Conjugate Gradient Squared
CUDA	Compute Unified Device Architecture
DGF-FDTD	Discrete Green's Function, Finite-Difference Time-Domain Method
GetDP	General environment for the Treatment of Discrete Problems
GPU	Graphics Processing Unit
ISEF	International Symposium on Electromagnetic Fields
LME	Laboratorio de Máquinas Eléctricas
MEMs	Microelectromechanical systems
MIMD	Multiple Instruction Multiple Data
MPI	Message Passing Interface
PETSc	Portable, Extensible Toolkit for Scientific Computation
SPMD	Single Program Multiple Data

Lista de símbolos y unidades

$\mathbb{G}$	Matriz de incidencia aristas – nudos del primal	[-]
$\mathbb{C}$	Matriz de incidencia caras – aristas del primal	[-]
$\mathbb{D}$	Matriz de incidencia volúmenes – caras del primal	[-]
$\widetilde{\mathbb{G}}$	Matriz de incidencia aristas – nudos del dual	[-]
$\widetilde{\mathbb{C}}$	Matriz de incidencia caras – aristas del dual	[-]
$\widetilde{\mathbb{D}}$	Matriz de incidencia volúmenes – caras del dual	[-]
$\mathbb{P}$	Conjunto de elementos de dimensión 0 del primal	[-]
p	Punto, nudo o vértice del primal	[-]
$\mathbb{L}$	Conjunto de elementos de dimensión 1 del primal	[-]
l	Línea o arista del primal	[-]
$\mathbb{S}$	Conjunto de elementos de dimensión 2 del primal	[-]
s	Superficie o cara del primal	[-]
$\mathbb{V}$	Conjunto de elementos de dimensión 3 del primal	[-]
v	Volumen o tetraedro del primal	[-]
$\widetilde{\mathbb{P}}$	Conjunto de elementos de dimensión 0 del dual	[-]
$\tilde{p}$	Punto, nudo o vértice del dual	[-]
$\tilde{\mathbb{L}}$	Conjunto de elementos de dimensión 1 del dual	[-]
$\tilde{l}$	Línea o arista del dual	[-]
$\tilde{\mathbb{S}}$	Conjunto de elementos de dimensión 2 del dual	[-]
$\tilde{s}$	Superficie o cara del dual	[-]
$\widetilde{\mathbb{V}}$	Conjunto de elementos de dimensión 3 del dual	[-]
$\tilde{v}$	Volumen o tetraedro del dual	[-]
t	Instante de tiempo del primal	[-]
$\tilde{t}$	Instante de tiempo del dual	[-]
$\mathbb{T}$	Intervalo de tiempo del primal	[-]
$\widetilde{\mathbb{T}}$	Intervalo de tiempo del dual	[-]
Ω	Dominio	[-]
χ	Característica de Euler	[-]

g	Género o número de huecos topológicos de una figura	[-]
j	Unidad imaginaria	[-]
ω	Frecuencia angular	[rad/s]
U	Tensión eléctrica	[V]
Φ	Flujo magnético	[Wb]
$\mathcal{F}_m$	Fuerza magnetomotriz	[A]
I	Corriente	[A]
Ψ	Flujo eléctrico	[Wb]
$\mathcal{V}$	Reluctancia magnética	[Wb]
σ	Conductividad eléctrica	[S/m]
a	Escalar potencial magnético	[Wb]
$M_{\mathcal{V}}$	Matriz constitutiva magnética	[-]
M_{σ}	Matriz constitutiva eléctrica	[-]
Q^c	Contenido de carga eléctrica	[Q]
E	Fuerza electromotriz	[V]
ε	Impulso de fuerza electromotriz	[Wb]
$\vec{A}$	Vector potencial magnético	[Wb/m]
$\vec{B}$	Densidad de flujo magnético	[Wb/m^2]
$\vec{H}$	Intensidad de campo magnético	[A/m]
J	Densidad volumétrica de corriente eléctrica	[A/m^2]

Índice

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1 Antecedentes

Este Trabajo de Fin de Máster sigue con las líneas de trabajo de otros estudios y publicaciones en las que se emplea el Método de la Celda para resolver problemas electromagnéticos mediante computación secuencial.

En la Tesis Doctoral de L. Simón Rodríguez, "Aportaciones al Método de la Celda en el Diseño y Análisis de un Modelo de Máquina Rotativa Trifásica de Inducción Magnética", de Marzo de 2015, se estudia un modelo de máquina trifásica de inducción magnética desde la Formulación Finita, concretamente mediante el Método de la Celda [1].

En primer lugar, se implementan las ecuaciones de Maxwell planteadas desde la Formulación Finita a un modelo de corrientes inducidas y, seguidamente, se plantea una matriz global de las ecuaciones circuitales en las que se incluyen estas ecuaciones de Maxwell basadas la Formulación Finita y el Método Nodal Modificado.

Adicionalmente, en esta tesis se aplica el Método de la Celda al análisis de una máquina de inducción magnética. Distintos aspectos de este análisis, como el análisis de fallos o el ajuste paramétrico, se realizan mediante la implementación de programas secuenciales.

La Tesis Doctoral de P. I. González Domínguez, "Aportaciones al Diseño de Máquinas Eléctricas de Inducción mediante el Método de la Celda. Análisis Térmico y Electromagnético", de Noviembre de 2015, se centra en explicar numéricamente mediante el Método de la Celda el acoplamiento electromagnético y térmico existente en las máquinas asíncronas [2].

Al igual que en la tesis anterior, inicialmente se plantean las ecuaciones de Maxwell, las ecuaciones circuitales y el Método Nodal Modificado desde la perspectiva de la Formulación Finita, añadiendo en este caso las ecuaciones térmicas correspondientes.

Posteriormente, se diseñan y ejecutan una serie de experimentos numéricos para comprobar la validez de las ecuaciones anteriores en el Método de la Celda. Esto se lleva a cabo mediante el uso de aplicaciones informáticas existentes y programas creados específicamente para este propósito, siempre de forma secuencial.

En las tesis anteriores se propone la posibilidad de implementar mediante técnicas de computación paralela los programas empleados en los análisis y experimentos. De esta forma, se conseguiría mejorar el detalle de los resultados obtenidos y reducir los tiempos de ejecución de estos programas. Por ello, este Trabajo de Fin de Máster surge de la idea de paralelizar los programas secuenciales del pre-procesado más relevantes y comprobar la viabilidad de esta técnica computacional.

1.2 Objetivos

El objetivo principal de este Trabajo de Fin de Máster es la paralelización de los programas del pre-procesado del análisis de sistemas basado en el Método de la Celda para mejorar los tiempos de ejecución de dichos programas.

Para ello, en primer lugar se estudiará el Método de la Celda y se reescribirán, desde la perspectiva de este método numérico, las leyes electromagnéticas utilizadas en este trabajo para comprender así la teoría detrás del análisis de sistemas electromagnéticos.

También será conveniente la comprensión de la programación paralela genérica y de las herramientas a utilizar en este trabajo como paso previo a la paralelización de los algoritmos del pre-procesado.

Como conclusión de este trabajo, se estudiarán los resultados obtenidos, tanto de los programas secuenciales originales y los optimizados, como de los programas ya paralelizados. Se deberá determinar si el paralelizar estos programas proporciona una reducción de los tiempos de ejecución que compense el utilizar ordenadores con procesadores en los que implementar esta filosofía de programación paralela.

Finalmente, al ser un trabajo de investigación, otro de los objetivos es la difusión de sus contenidos para que éstos sean recibidos por la comunidad científica internacional. Actualmente, se han aceptado dos artículos relacionados con este trabajo que serán expuestos en ISEF 2017, el 18º Simposio Internacional de Campos Electromagnéticos en Ingeniería Mecatrónica, Eléctrica y Electrónica. Estos dos artículos son los titulados “Finite Formulation in Parallel Computation. Application to Electromagnetic Field in 3-D” y “Thermal Constitutive Matrix Applied to Asynchronous Electrical Machine Using the Cell Method”, y serán publicados y presentados para el congreso de ISEF 2017 que se celebrará en Septiembre de 2017 en Lodz, Polonia.

1.3 Estructura de la memoria

Esta memoria está estructurada en diez capítulos, una bibliografía y un anexo, cuyos contenidos se detallarán a continuación.

En este primer capítulo se describen los antecedentes de este Trabajo de Fin de Máster, donde se exponen los motivos para su realización; se detallan los objetivos propuestos para este trabajo, y se explica el contenido de la memoria.

En el segundo capítulo se expone el estado del arte de este proyecto. Se trata el Método de la Celda y sus aplicaciones en computación secuencial, se habla sobre las aplicaciones de otros métodos numéricos en computación paralela y se comentan las diversas herramientas software empleadas en este trabajo.

En el tercer capítulo se trata la teoría del Método de la Celda. Se explican las celdas complejas, tanto del primal como del dual, con sus orientaciones internas o externas; las matrices de incidencia; la fórmula de Euler-Poincaré y se compara este método numérico con otros métodos similares.

En el cuarto capítulo se reescriben las principales leyes electromagnéticas desde la formulación finita, específicamente desde la perspectiva del Método de la Celda. Para ello, se tratan las leyes físicas que se emplean en este proyecto, como las ecuaciones de Maxwell y las ecuaciones circuitales.

En el quinto capítulo se explica el mallado de un sistema y el software empleado para construirlo. Se describen los principales módulos de dicha herramienta y se muestran ejemplos de mallados de diversos sistemas.

En el sexto capítulo se introduce el concepto de programación paralela y se explican los tipos de paralelización y sus principales objetivos. Además, se describe el interfaz de paso de mensajes empleado en este trabajo y se muestran ejemplos de programas paralelos básicos.

En el séptimo capítulo se describe PETSc, la herramienta software empleada en la programación de los algoritmos del análisis de sistemas físicos en este proyecto, se compara con el interfaz básico de paso de mensajes estudiado en el capítulo anterior, y se explican unos programas de ejemplo basados en esta herramienta.

En el octavo capítulo se estudian los programas empleados en el pre-procesado y se muestran los resultados obtenidos por estos programas. Además, algunos de estos programas iniciales son optimizados, comentando también la mejoría en los resultados de esta optimización.

En el noveno capítulo se explica el proceso de paralelización de los programas secuenciales, tanto genérico como en concreto para cada algoritmo, y se describen los resultados obtenidos en este trabajo.

En el décimo capítulo se muestran los resultados del post-procesado de este mismo proyecto, se comparan los objetivos iniciales con los resultados finales obtenidos, y se comentan las futuras líneas de trabajo.

Finalmente, se incluye la bibliografía utilizada en la documentación de este Trabajo de Fin de Máster además de dos anexos, uno con los artículos aceptados pendientes de publicar, y otro con el código de los programas empleados en el pre-procesado de este proyecto.

Capítulo 2. Estado del arte

2.1 Introducción

A continuación, se describirá el estado del conocimiento sobre el método numérico conocido como Método de la Celda. Se incluirán las principales aplicaciones computacionales, tanto serie como paralelas, basadas en dicho método numérico u otros métodos numéricos similares. Finalmente, se revisarán las herramientas softwares e interfaces más relevantes en esta aplicación.

2.2 Método de la celda

El método de la celda como método numérico basado en la formulación discreta fue propuesto a finales de la década de los 90. En 1998, el físico y matemático italiano Enzo Tonti comienza a presentar artículos donde propone el uso del método de la celda para la solución de problemas físicos sin recurrir a la formulación diferencial.

En 1999 se publicó "On the Geometrical Structure of Electromagnetism" [3] y "A direct discrete formulation for the wave equation" [4], donde se explica el uso de la formulación discreta en el electromagnetismo y en la acústica, respectivamente, utilizando topología algebraica. Para ello, se analizan propiedades físicas y ecuaciones de forma discreta en ambos campos de la física, recurriendo a un complejo espaciotemporal y a su dual.

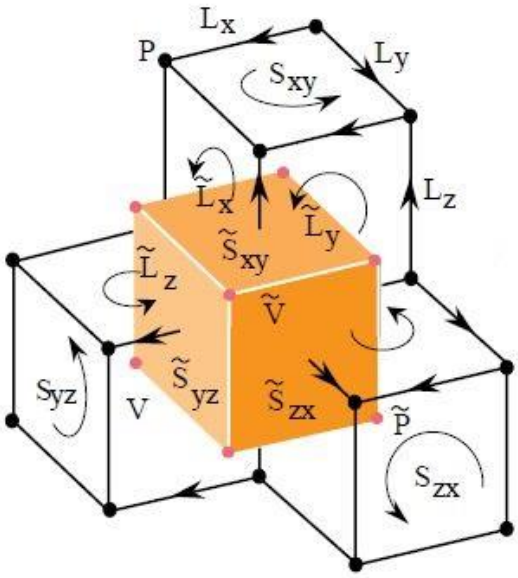

Figura 1. Esquema del complejo espacial primal (blanco) y su dual (naranja).

Además, se introdujeron los distintos tipos de celdas espaciotemporales y se relacionaron con magnitudes físicas. Se explican los dos tipos de orientación, interna y externa, y cómo se asocian a tipos de variables globales de la física y a sus distintos tipos, específicamente en el electromagnetismo y en la acústica. También se describen los tipos de complejos dependiendo de su número de aristas o caras (simpliciales o celdas complejas) y se explican los

duales de los complejos frente a los primales. Igualmente, los artículos se adentran en la formulación discreta al tratar los números y matrices de incidencia, las cadenas de células, las co-caras, las co-cadenas y más leyes topológicas. De esta forma, se presenta el método de la celda como un método numérico sencillo, que es consistente con los resultados experimentales y de fácil implementación en la computación numérica.

A principio de la siguiente década, Enzo Tonti presenta artículos del método de la celda más generales, sin centrarse únicamente en un solo ámbito de la física. "A discrete Formulation of Field Laws: The Cell Method" [5], publicado en 2001, explica la importancia de las variables globales y sus tipos para este método en mayor profundidad además de describir cómo el método de la celda se adapta a diversos campos de la física, como la termodinámica o el electromagnetismo. "A Classification Diagram for Physical Variables" [6], presentado en 2003, muestra y explica diferentes diagramas de clasificación, como el de la mecánica de partículas o de fluidos, aunque en el que más se centra es en el diagrama de clasificación del electromagnetismo.

2.3 Método de la celda en computación serie

Ya en el año 2000, M. Repetto y F. Trevisan presentaron su artículo "Global formulation of 3D magnetostatics using flux and gauged potentials" [7], donde se lleva a cabo una revisión del método numérico; se analizan diversas leyes del electromagnetismo formuladas mediante este método, como la ley magnética de Gauss, la ley de Ampère o la ley de conservación de carga; y se muestran los resultados de la implementación de una serie de ejemplos básicos en MATLAB y en Fortran.

En esa misma década, ingenieros de la Universidad de Padova, en Italia, utilizan el método de la celda para solucionar problemas termo-electromagnéticos acoplados en aplicaciones médicas que utilizan un tratamiento térmico mediante radiofrecuencia. En 2004 presentan "Isotropic and Anisotropic Electrostatic Field Computation by Means of the Cell Method" [8], artículo que explica cómo utilizar este método numérico para resolver problemas electrostáticos mediante aplicación computacional.

En 2006 publican "Nonlinear Coupled Thermo-Electromagnetic Problems With the Cell Method" [9] y "Coupled Electrical and Thermal Transient Conduction Problems With a Quadratic Interpolation Cell Method Approach" [10], donde se emplea el método de la celda para la solución de problemas termo-electromagnéticos acoplados y se contrastan los resultados obtenidos experimentalmente, con los obtenidos mediante un algoritmo basado en el método de la celda y con los de otro algoritmo basado en el método de los elementos finitos. También se plantea el uso de interpolación cuadrática en este tipo de problemas.

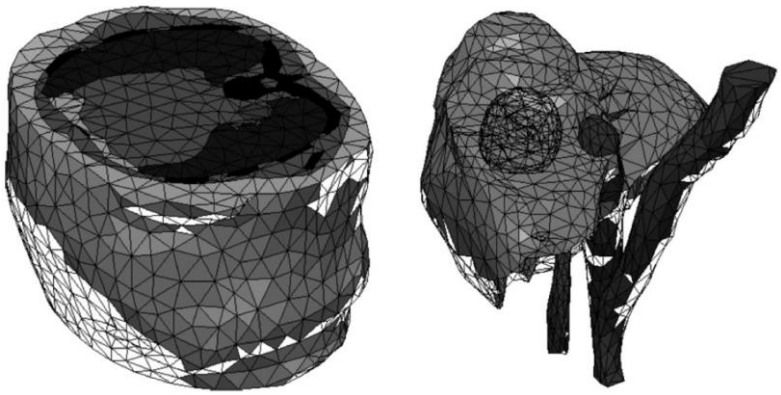

Figura 2. Mallado en 3D de la simulación de la zona abdominal humana y del hígado.

Finalmente, en 2007 presentan "A 3-D Cell Method Formulation for Coupled Electric and Thermal Problems" [11], donde se introduce el estudio de la conducción eléctrica y térmica transitoria del problema y se explica el método de la celda en el dominio espacial y temporal conjuntamente.

2.4 Formulación finita en computación paralela

Existen diversos equipos de trabajo independientes que han desarrollado aplicaciones basadas en formulación finita con arquitectura paralela. Sin embargo, no utilizan el método de la celda, sino otros métodos numéricos como el método de los elementos finitos o el método de las diferencias finitas.

En 1990 se presentó un artículo que revisaba diversas variantes del método de los elementos finitos y de los parámetros que mejorarían los resultados de su implementación en paralelo para la solución de problemas de electromagnetismo, con el título "An Examination of Finite Element Formulations and Parameters for Accurate Parallel Solution of Electromagnetic Scattering Problems" [12].

Durante esa década se publicaron diversos trabajos que proponían la implementación en paralelo del método de los elementos finitos. En 1994 se plantea la utilización de una arquitectura paralela basada en dicho método para la solución de problemas de electromagnetismo en "Parallel Computation of 3D Electromagnetic Scattering using Finite Elements and Conformal ABCs" [13]; y para la solución de problemas en la simulación 3D de formación de metales en “A Clustered Reduced Communication Element by Element Preconditioned Conjugate Gradient Algorithm for Finite Element Computations" [14].

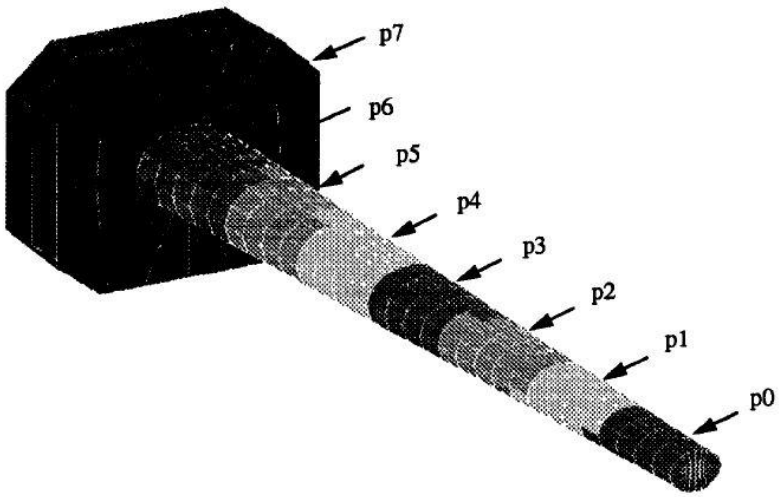

Figura 3. Mallado de una pieza metálica en ocho procesadores.

En 1995 se publicó un artículo, "Parallel Solution of Unstructured Sparse Finite Element Equations" [15], que describe el uso de programación paralela para la solución de ecuaciones basadas también en el método de los elementos finitos. Se propuso el uso de máquinas MIMD (Multiple Instruction Multiple Data) y de un algoritmo CGS (conjugate gradient squared) para la resolución de problemas de este método en campos como el electromagnético. El año siguiente se presentó un algoritmo para la solución de problemas basados en el método de los elementos finitos en el dominio del tiempo, implementado en ordenadores con paso de mensajes en paralelo en el artículo "An Unconditionally Parallel Finite Element Time Domain Algorithm" [16].

En los trabajos presentados en 1997, "Finite Elements and Absorbing Boundary Conditions for Scattering Problems on a Parallel Distributed Memory Computer" [17] y en 1998, "Implementation of a Finite Element and Absorbing Boundary Conditions Package on a Parallel Shared Memory Computer" [18] se mostraron resultados de simulaciones de sistemas reales basadas en el método de los elementos finitos y la mejoría en los tiempos de ejecución gracias a la paralelización de los algoritmos.

En 2004 se publicó el artículo "Generalized h-p Triangles and Tetrahedra for Adaptive Finite Element Analysis in Parallel Processing Environments" [19] que propone nuevas familias de triángulos y tetraedros para la utilización del método de los elementos finitos en el procesamiento en paralelo.

El trabajo presentado en 2007, "Parallel Implementation of the Matrix Formulation of the FDTD Scheme" [20], explica la implementación de un algoritmo para formulación matricial del método de las diferencias finitas en el dominio del tiempo, en arquitecturas paralelas mediante la interfaz de paso de mensajes MPI. Esta interfaz paralela también se empleó en el trabajo que describe el artículo "Parallel Simulation of Two-phase Flow Problems Using the Finite Element Method" [21] en el que se utiliza una arquitectura paralela del método de los elementos finitos para la solución de problemas de fluidos.

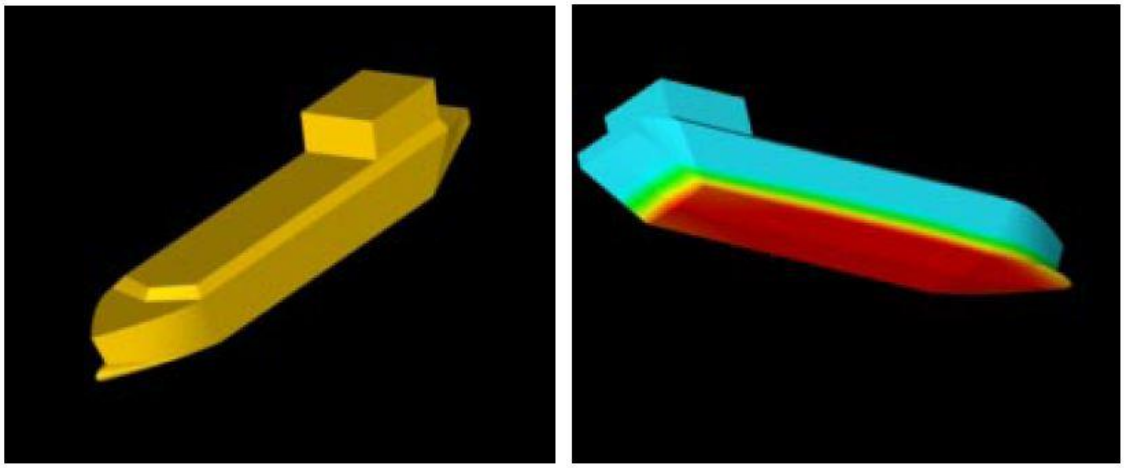

Figura 4. Modelo de un barco y distribución de presiones.

En 2012, un grupo investigador de la universidad Gdansk, en Polonia, desarrolló un algoritmo paralelo basado en la formulación de la función discreta de Green del método de las diferencias finitas. Este algoritmo se implementó en una unidad de procesamiento gráfico GPU siguiendo una arquitectura CUDA, según se puede leer en los artículos "Acceleration of the DGF-FDTD Method on GPU Using the CUDA Technology" [22] y "Parallel Implementation of the DGF-FDTD Method on GPU Using the CUDA Technology" [23].

Finalmente, en 2013 se presentó un artículo, "Time-Domain Parallel Finite-Element Method for Fast Magnetic Field Analysis of Induction Motors" [24], donde se propone un procedimiento para paralelizar análisis estáticos y transitorios basados en el método de los elementos finitos en el dominio del tiempo.

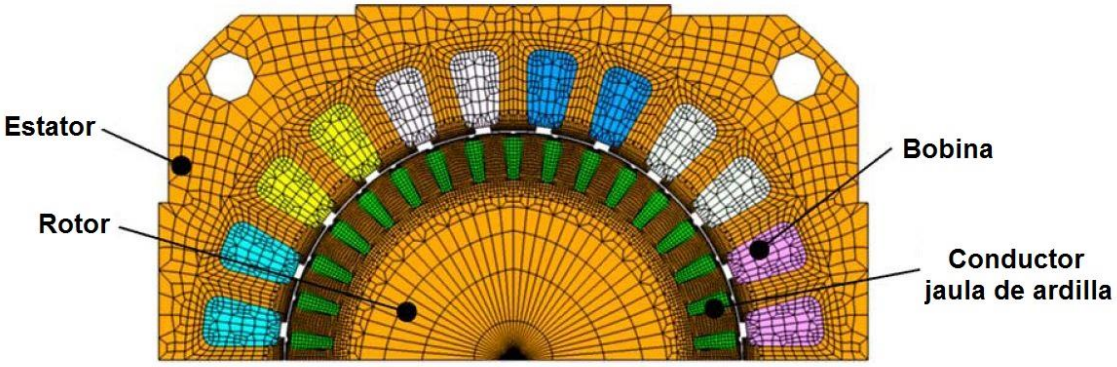

Figura 5. Mallado de un modelo de motor de inducción.

2.5 Herramientas software e interfaces útiles

Para la realización de este trabajo se han empleado diversas herramientas que facilitan la paralelización de programas de pre-procesado basados en el método de la celda. Por ejemplo, Gmsh es un software que genera mallados tridimensionales de elementos finitos que empezó a desarrollarse en 1996. Dos años más tarde se publicó una primera versión del software, hasta que, en 2007, la segunda versión de Gmsh comenzó a ser utilizada, según indica el artículo

"Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities" [25]. Esta publicación, además de revisar la historia de Gmsh, explica el diseño del software y los distintos módulos con los que cuenta mostrando diversos ejemplos de su utilización.

Por otro lado, MPI, o Message Passing Interface en inglés, es un interfaz estandarizado desarrollado por el MPI Forum que define la sintaxis y la semántica de funciones para la realización de aplicaciones paralelas basadas en paso de mensajes. Este interfaz se comenzó a desarrollar en 1992 hasta que en 1994 se presentó la primera versión de MPI. Actualmente, la última versión de MPI implementada es MPI 3.1, según se recoge en el documento "MPI: A Message-Passing Interface Standard, Version 3.1" [26] publicado por el MPI Forum, donde se explican todos los detalles del interfaz MPI en esta última versión.

Por último, PETSc (Portable, Extensible Toolkit for Scientific Computation) es una herramienta desarrollada por el Laboratorio Nacional Argonne de Chicago para la solución de problemas mediante ecuaciones diferenciales parciales además de permitir la paralelización de algoritmos. La versión actual de PETSc es la versión 3.7, según se explica en el documento "PETSc Users Manual, Revision 3.7" [27], donde se describe el uso de PETSc además de su diseño y elementos principales.

2.6 Conclusiones

El método de la celda es un método numérico que empezó a ser utilizado en distintas aplicaciones en la década de 1990. En el año 2000 comenzó a empleare para resolver problemas físicos mediante computación secuencial. Sin embargo, los métodos numéricos utilizados en computación paralela son otros, como el método de los elementos finitos o el método de las diferencias finitas. Por lo tanto, al no haberse podido encontrar ninguna referencia al uso del método de la celda en computación paralela, se puede llegar a la conclusión de que esta línea de investigación es innovadora y podría llegar a ser pionera como campo de estudio.

Capítulo 3. Método de la celda

3.1. Introducción

El método de la celda es un método numérico que hace uso de la formulación discreta de las ecuaciones de campo. Se basa en el uso de variables globales y de un par de complejos, el primal y el dual, donde cada elemento cuenta con una orientación que puede ser interna o externa.

Este método permite asociar propiedades físicas del electromagnetismo con elementos del espacio-tiempo perteneciente a sus celdas, como puntos, líneas, superficies, volúmenes, instantes o intervalos, y así poder resolver problemas físicos.

3.2. Variables globales

La formulación finita de teorías físicas como el electromagnetismo se basa en topología algebraica básica. La descripción matemática de fenómenos físicos descansa en la existencia de atributos cuantitativos de sistemas físicos descritos mediante magnitudes físicas.

La mayoría de magnitudes físicas se asocian con cuatro elementos espaciales (puntos, líneas, superficies y volúmenes) y dos elementos temporales (instantes e intervalos) que se pueden combinar entre sí para dar ocho elementos espaciotemporales [5].

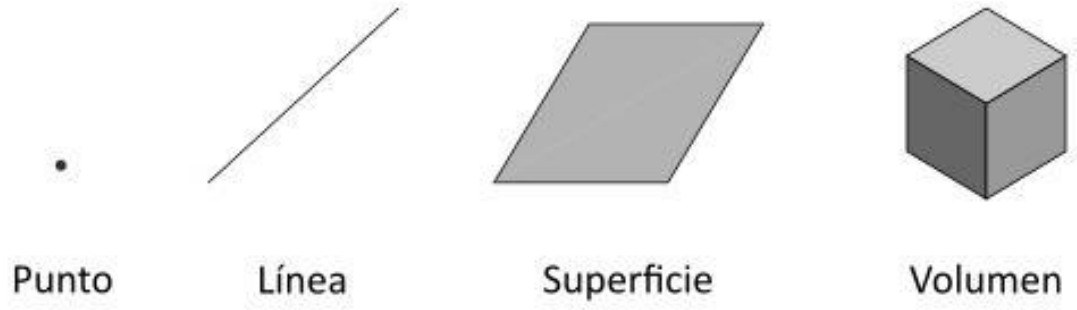

Figura 6. Elementos espaciales de la formulación finita.

En estas asociaciones serán de gran importancia los conceptos de orientación (interna o externa) y de compleja primal y dual.

La formulación discreta se basa en variables globales, comúnmente conocidas como variables de integración, ya que estas se asocian a elementos espaciales y temporales como puntos, líneas, superficies, volúmenes, instantes e intervalos de tiempo; los elementos básicos de esta teoría [4].

Las variables globales tienen la ventaja de poder medirse empíricamente, lo que permite mantener una conexión entre la simulación y los datos experimentales. Además, al utilizar este tipo de variable, no se necesitan condiciones de derivabilidad ni el uso de infinitos en límites matemáticos.

Dependiendo de la función de las variables globales, estas se pueden clasificar en variables de configuración, de fuente o de energía [6].

Las variables de configuración describen la configuración del campo, como el potencial eléctrico, el campo eléctrico, el flujo magnético, la densidad de flujo magnético o la temperatura.

Las variables fuente describen las fuentes de un campo, como el flujo de carga eléctrica, la corriente eléctrica, el flujo eléctrico o el flujo térmico.

Las variables de energía son las obtenidas a partir del producto de variables de configuración y de fuente, como resultaría ser el trabajo, el calor, la energía magnética o la energía potencial.

Según la regla de asociación espaciotemporal, las variables globales de configuración se asocian a elementos espaciales y temporales con orientación interna del primal, mientras que las variables globales fuente y de energía se asocian a elementos espaciales y temporales con orientación externa del dual.

3.3. Celdas complejas

Mientras que en la formulación diferencial se suele utilizar un sistema de coordenadas, en la formulación discreta el dominio del espacio físico se subdivide en subdominios de células contiguas de cualquier forma o dimensión.

Estas células pueden ser puntos, líneas, superficies y volúmenes, que se denominan 0-cells, 1-cells, 2-cells y 3-cells según su dimensión, siguiendo la nomenclatura inglesa. Las líneas y superficies pueden ser rectas o curvas, aunque para simplificar el problema se suelen considerar rectas. Las celdas complejas pueden estar formados por cuadrados y cubos para las 2-cells y 3-cells respectivamente, como suele suceder en física al facilitar la formulación diferencial. No obstante, para la solución numérica de problemas físicos se utilizan complejos simpliciales, con triángulos y tetraedros para las 2-cells y 3-cells respectivamente. De esta forma se pueden ajustar con mayor precisión las fronteras de las regiones y los mallados, especialmente en figuras con líneas o superficies curvas [3].

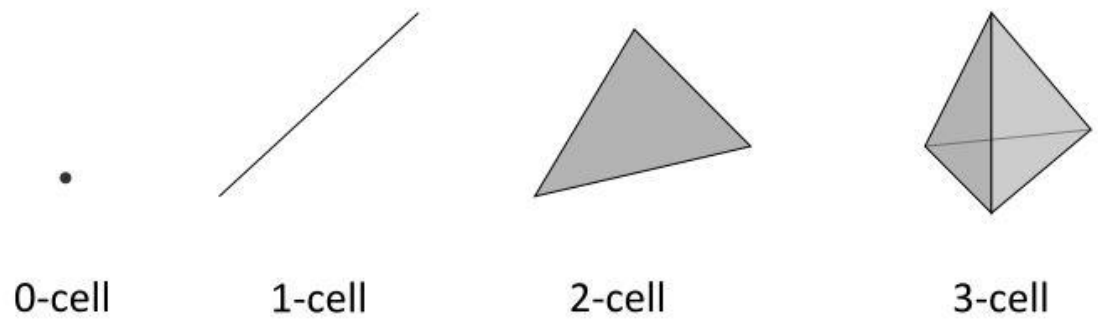

Figura 7. Células físicas de un complejo simplicial.

Los elementos como volúmenes, superficies o líneas tienen caras, que son elementos de una dimensión inferior que lo delimitan, por ejemplo, las superficies que delimitan un cubo.

Los elementos como puntos, líneas o superficies pueden tener "co-caras", que son aquellos elementos de una dimensión superior que lo tienen en común o lo comparten. Por ejemplo, teniendo dos habitaciones contiguas, las co-caras de la pared que separa las dos habitaciones son las propias habitaciones [6].

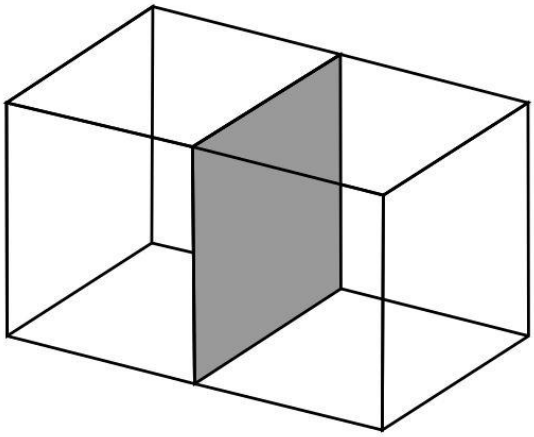

Figura 8. Co-caras de la superficie gris.

Además de elementos espaciales también existen elementos temporales. Estos podrán ser de cero dimensiones, como el instante; o de una dimensión, como el intervalo de tiempo.

Para los distintos ámbitos de la física, ya sea electromagnetismo, termodinámica o elasticidad, se pueden relacionar las variables globales con los distintos tipos de células. Por ejemplo, los potenciales se asocian a las 0-cells, las tensiones a las 1-cells, los flujos a las 2-cells, y los contenidos en las 3-cells [5].

Además, existen variables globales espaciotemporales, es decir, variables globales que se relacionan con un elemento espacial y temporal. Por ejemplo, el flujo magnético se asocia a la superficie y al instante, y el flujo de carga eléctrica a la superficie y al intervalo de tiempo.

3.4. Orientación

En la asociación de las variables globales a los elementos temporales y espaciales se debe tener en cuenta la orientación del elemento, ya que ésta condiciona el signo de dicha variable. De hecho, toda variable global física cambia de signo cuando se cambia la orientación del elemento al que se refiere [3].

Cuando la orientación del elemento espacial es interna, esta orientación se encuentra incluida en el propio elemento. Según se observa en la parte superior de la figura 9, los puntos o 0-cells con orientación interna actúan como fuentes o sumideros, siguiendo la nomenclatura del electromagnetismo o de la dinámica de fluidos; las líneas o 1-cells con orientación interna siguen la dirección de la propia línea; las superficies o 2-cells con orientación interna siguen una

orientación incluida en la propia superficie; y los volúmenes o 3-cells con orientación interna siguen la orientación de las superficies que lo delimitan.

Sin embargo, cuando la orientación del elemento espacial es externa, esta orientación depende del espacio donde se encuentre dicho elemento. Según se pueden ver en la parte inferior de la figura 9, los volúmenes con orientación externa siguen una orientación perpendicular a las superficies que lo delimitan; las superficies con orientación externa siguen una orientación perpendicular a dicha superficie; las líneas con orientación externa siguen la orientación de rotación con la propia línea como eje; y que un punto tenga orientación externa significa que las líneas que tienen un extremo en dicho punto también tienen orientación externa.

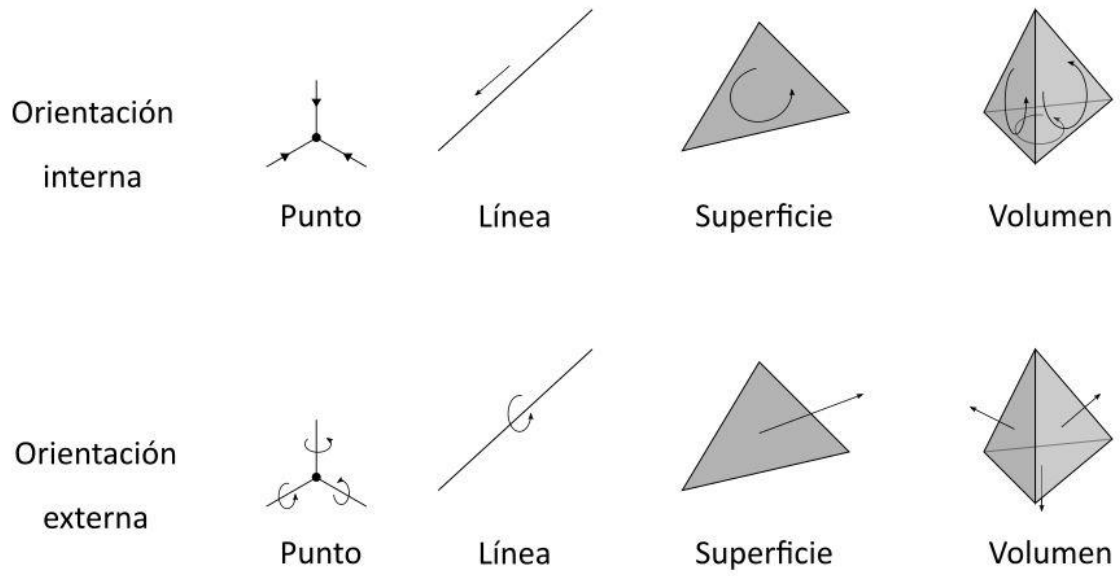

Figura 9. Orientación de los elementos físicos.

Los elementos temporales también tienen orientación, aunque en la práctica esta siempre será interior. Un intervalo tendrá orientación interna cuando se orientan del pasado al futuro, y un instante tendrá orientación interna cuando se conciba como sumidero [4].

En el método de la celda existe el concepto de cadena, que es un conjunto de p-cells o células de la misma dimensión que tienen la misma orientación.

Las variables globales físicas del electromagnetismo pueden tener orientación interna, como el impulso de tensión eléctrica (a) o el flujo magnético (b); u orientación externa, como el flujo eléctrico (c) o el impulso de tensión magnética (d) [3].

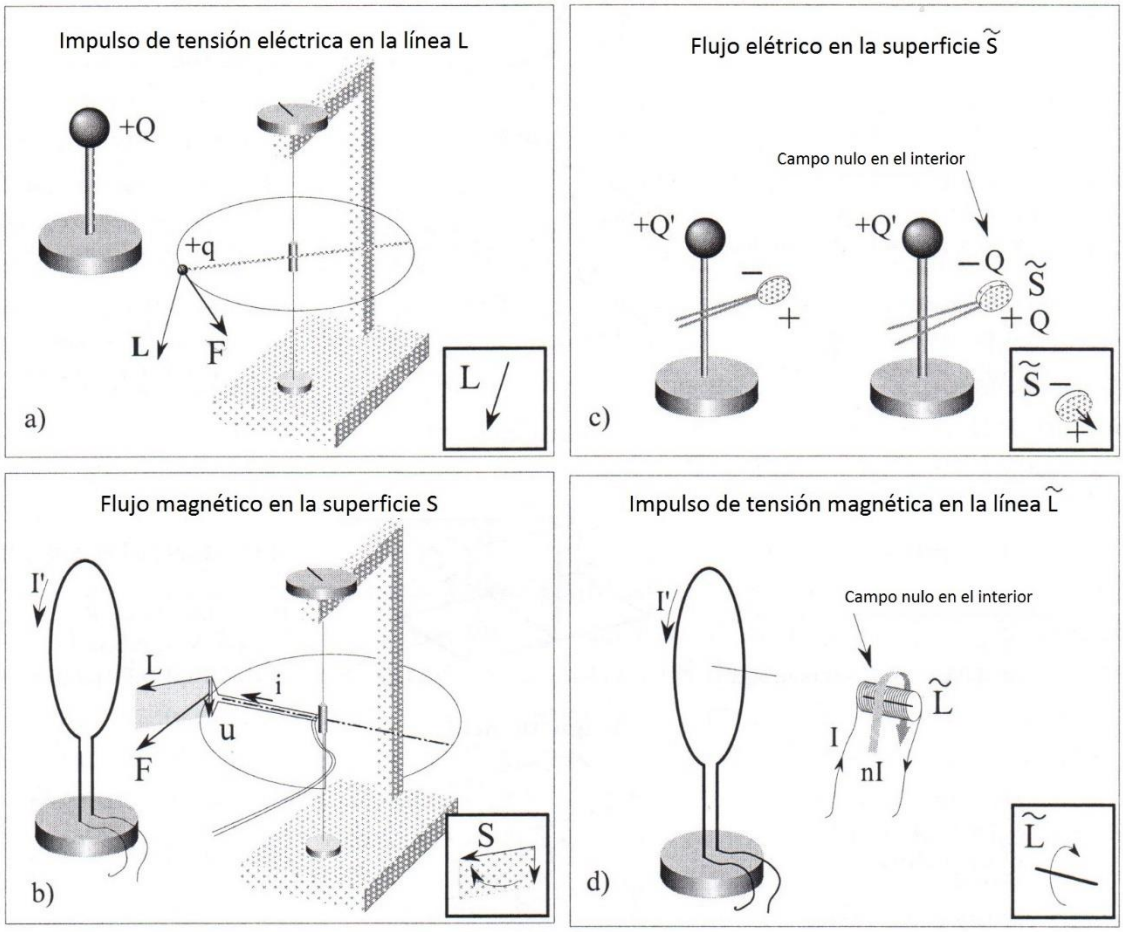

Figura 10. Ejemplos de variables globales del electromagnetismo asociadas a elementos físicos.

3.5. Complejo primal y dual

Las magnitudes físicas pueden estar asociadas a células de un complejo o a su dual. Las células del primal tendrán orientación interna mientras que las del dual tendrán orientación externa [4].

Dado una celda del primal de tres dimensiones, se situará un punto dentro de cada volumen, concretamente en su baricentro: este será el 0-cell del dual. Las 1-cells del dual serán las líneas que conecten los baricentros de dos 3-cells adyacentes del primal, es decir, las líneas del dual conectarán las 0-cells del dual. A partir de las 1-cells duales se construirá el resto del sistema dual: las 2-cells duales son las superficies delimitadas por las aristas duales y las 3-cells duales son los volúmenes delimitadas por las superficies duales. Esto se puede resumir en que los baricentros de los volúmenes del primal serán los vértices de los volúmenes del dual.

Basándose en el ejemplo térmico, la producción de calor S se relaciona con dos habitaciones o volúmenes contiguos (dual), y el flujo de energía o calor Q se asocia a la pared o superficie que los separa (dual). Sin embargo, la temperatura T de cada habitación o volumen se asocia al baricentro del mismo (primal), la diferencia de temperatura P se relaciona con la línea que une dichos baricentros (primal) [5].

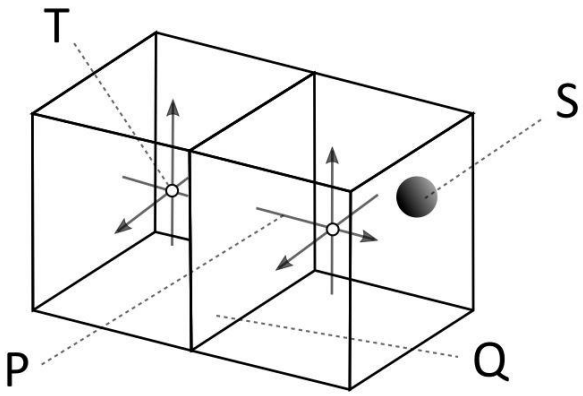

Figura 11. Relación de magnitudes físicas del campo térmico con elementos de los complejos primal y dual.

De la misma forma, se pueden asociar las magnitudes físicas del electromagnetismo a la célula compleja del primal y de su dual. En el campo eléctrico, el potencial eléctrico V se asocia al punto del primal, el voltaje E a la línea del primal, el flujo eléctrico F_e a la pared del dual y el contenido de carga Q_c al volumen del dual.

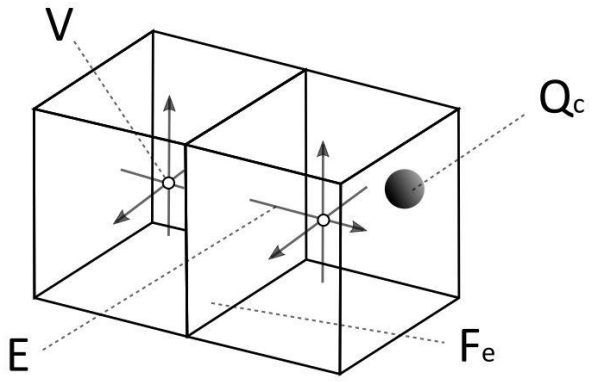

Figura 12. Relación de magnitudes físicas del campo eléctrico con elementos de los complejos primal y dual.

En el campo magnético se asocia el potencial magnético V_m al punto del dual, la tensión magnética U_m a la línea del dual, el flujo magnético F_m a la superficie del primal y el contenido de carga magnética G_c al volumen del primal.

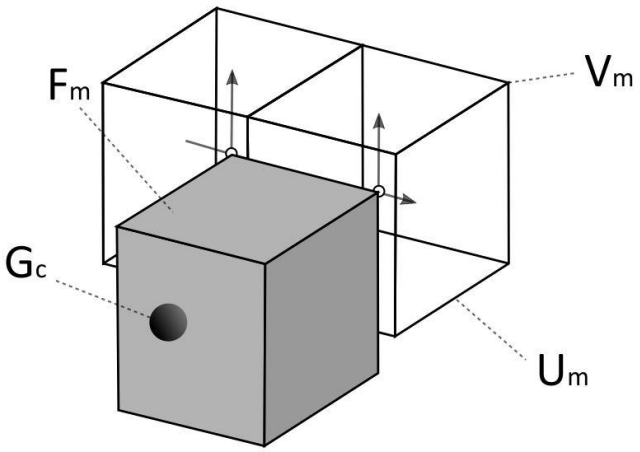

Figura 13. Relación de magnitudes físicas del campo magnético con elementos de los complejos primal y dual.

Al igual que existe un dual espacial, también existe un complejo dual temporal. Este estará formado por los instantes (dual) intermedios de los intervalos del primal, conectados mediante intervalos del dual. No obstante, los complejos temporales duales no se consideran en la Física, y es por ello que no se utiliza la orientación externa en los elementos temporales [4].

3.6. Matrices de incidencia

Las matrices de incidencia son elementos topológicos que representan la relación de los pares de elementos ordenados que los forman. La matriz de incidencia G representa la relación de incidencia entre líneas y puntos, la matriz C entre caras y líneas, y la matriz D entre volúmenes y caras. Las letras G, C y D que dan nombre a las matrices de incidencia provienen del gradiente, divergente y rotacional (curl en inglés) [6].

Las matrices de incidencia se forman a partir de los números de incidencia entre dos elementos de distinta dimensión de una celda compleja. Estos números podrán valer 0, si los elementos no son incidentes entre sí; 1 si son incidentes y sus orientaciones son compatibles; y -1 si son incidentes pero sus orientaciones no son compatibles.

Las matrices de incidencia del complejo dual se forman de la misma forma que las del complejo primal. La única diferencia radica en la forma de nombrarlas, utilizando un acento característico. Por ellos, las matrices de incidencia del dual serán la matriz $\widetilde{\mathrm{G}}$, la matriz $\widetilde{\mathrm{C}}$ y la matriz $\widetilde{\mathrm{D}}$.

Las matrices de incidencia del primal y del dual no son completamente independientes, al igual que no lo son los propios complejos primales y duales. Por ello, se cumplen las siguientes relaciones entre las matrices de incidencia de ambos complejos:

$$\widetilde{\mathrm{G}} = \mathrm{D}^{\mathrm{T}}$$

$$\widetilde{\mathrm{C}} = \mathrm{C}^{\mathrm{T}}$$

$$\widetilde{\mathrm{D}} = -\mathrm{G}^{\mathrm{T}}$$

$$C \cdot G = 0$$

$$D \cdot C = 0$$

3.6.1. Ejemplo de generación de matrices de incidencia

Para mostrar el cálculo de las matrices de incidencia será recomendable tener un sistema formado por al menos dos tetraedros. Seguidamente se muestra una imagen donde se enumeran arbitrariamente los cinco nudos del sistema [1]:

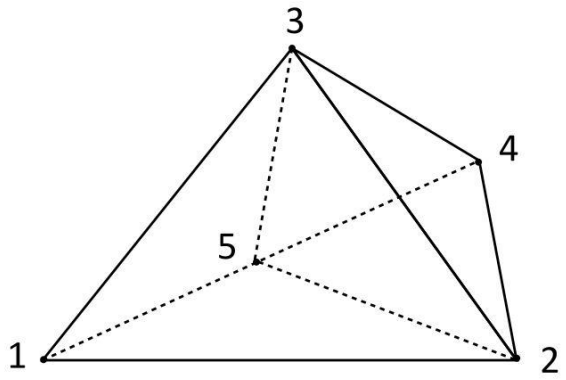

Figura 14. Nudos del sistema de dos tetraedros.

El conjunto de nudos es el siguiente:

Nudos = {1, 2, 3, 4, 5}

A continuación, se muestran las aristas del sistema, que cuentan con una orientación arbitraria:

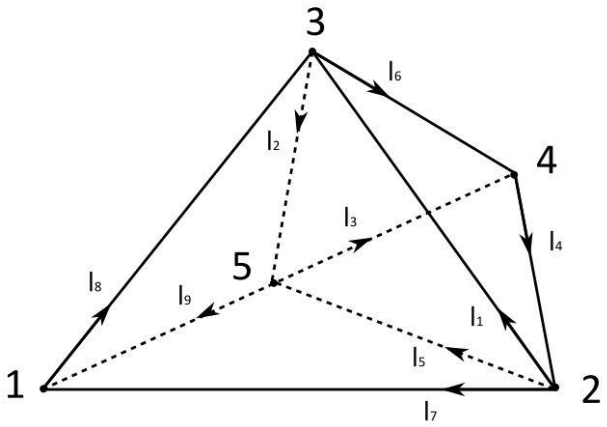

Figura 15. Aristas del sistema de dos tetraedros.

El conjunto de aristas es el siguiente:

Aristas = {l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8, l_9}

Cada arista está formada por un par de nudos, ordenados según el sentido de la misma:

Aristas: $l_1 = \{2, 3\}$; $l_2 = \{3, 5\}$; $l_3 = \{5, 4\}$; $l_4 = \{4, 2\}$; $l_5 = \{2, 5\}$; $l_6 = \{3, 4\}$;

$l_7 = \{2, 1\}$; $l_8 = \{1, 3\}$; $l_9 = \{5, 1\}$;

En la figura 16 se muestran las caras del sistema, con orientación arbitraria:

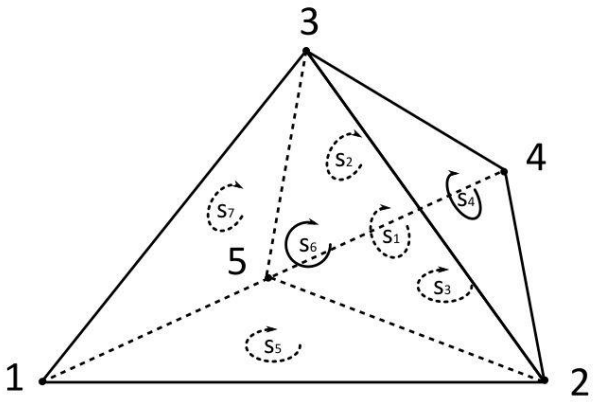

Figura 16. Caras del sistema de dos tetraedros.

El conjunto de caras es el siguiente:

$$\text{Caras} = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$$

Cada cara está formada por un triplete de nudos, ordenados según el sentido de la misma:

Caras: $s_1 = \{2, 5, 3\}$; $s_2 = \{3, 4, 5\}$; $s_3 = \{5, 4, 2\}$; $s_4 = \{3, 4, 2\}$; $s_5 = \{5, 2, 1\}$; $s_6 = \{2, 1, 3\}$; $s_7 = \{1, 3, 5\}$;

Por último, en la figura 17 se pueden observar los dos tetraedros del sistema:

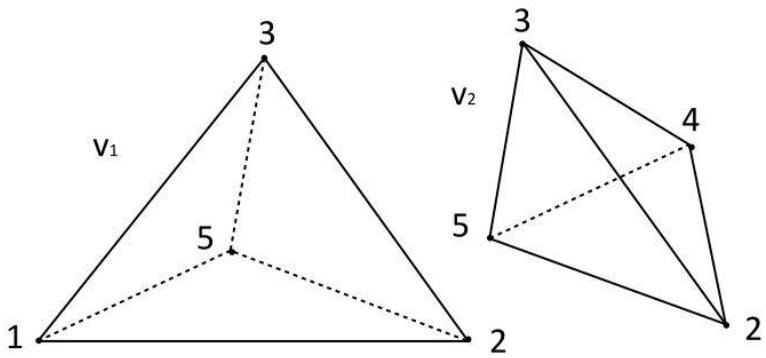

Figura 17. Volúmenes del sistema de dos tetraedros.

El conjunto de volúmenes es el siguiente:

$$\text{Volúmenes} = \{v_1, v_2\}$$

Cada volumen está formado por un conjunto de cuatro nudos, ordenados según el sentido arbitrario de sus aristas:

Volúmenes: v_1= {5, 2, 1, 3}; v_2 = {2, 3, 5, 4};

Una vez se tienen ordenados todos los elementos, se puede comenzar a construir las matrices de incidencia. La matriz G es la matriz de incidencia entre nudos y aristas. Por ello, será una matriz de 9 filas (correspondientes a las 9 aristas) y 5 columnas (correspondientes a los 5 nudos). Para cada arista, se indicará con un valor de -1 el nudo saliente y con valor de +1 el nudo entrante, según se puede ver en la siguiente matriz:

$$G = \begin{bmatrix} 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

La matriz C es la matriz de incidencia entre aristas y caras, por lo que será una matriz de 7 filas (correspondientes a las 7 caras) y 9 columnas (correspondientes a las 9 aristas). Para cada superficie, se indicará con un valor de -1 la arista con orientación contraria a la orientación de la superficie y con valor de +1 la arista con la misma orientación que la orientación de la superficie, según se puede ver en la siguiente matriz:

$$C = \begin{bmatrix} 1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

La matriz D es la matriz de incidencia entre caras y volúmenes, por lo que será una matriz de 2 filas (correspondientes a los 2 volúmenes) y 7 columnas (correspondientes a las 7 caras). Para cada volumen, se indicará con un valor de -1 la superficie con orientación hacia el interior del volumen (siguiendo la regla de la mano derecha) y con valor de +1 la superficie con orientación hacia el exterior del volumen (siguiendo la regla de la mano derecha), según se puede ver en la siguiente matriz:

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 & 0 & 0 & 0 \end{bmatrix}$$

3.7. La fórmula de Euler-Poincaré

La fórmula de Euler-Poincaré describe la relación entre el número de puntos (P), de líneas (L), de superficies (S) y de volúmenes (V) de un complejo de una variedad topológica. A continuación, se muestra la fórmula de Euler-Poincaré:

$$P - L + S - V = \chi(\Omega) \qquad \text{Ec. 3.1}$$

El último término de la ecuación, χ, es la constante de Euler-Poincaré. Para objetos de tres dimensiones, esta constante se determina mediante la siguiente expresión:

$$\chi = 1 - g \qquad \text{Ec. 3.2}$$

Donde g es el número de agujeros de dicho objeto [1].

Durante la realización del pre-procesado de los modelos estudiados, se ejecuta un programa encargado de comprobar y leer el número de nodos, aristas, caras y volúmenes del objeto para determinar así si la fórmula de Euler-Poincaré se cumple correctamente. De esta forma se verifica que los algoritmos previos donde se obtienen las aristas y las caras del objeto se han ejecutado correctamente. Seguidamente se muestran unos ejemplos del cálculo de la fórmula de Euler-Poincaré para objetos sólidos con distinto número de agujeros.

3.7.1. Sólido tridimensional sin agujeros - Bobina eléctrica

Para objetos tridimensionales sin agujeros (g = 0), la constante de Euler-Poincaré valdrá 1 ($\chi = 1 - 0$), quedando la fórmula de Euler-Poincaré de la siguiente forma:

$$P - L + S - V = 1 \qquad \text{Ec. 3.3}$$

A continuación, se muestra la estructura del sistema de la bobina y su mallado. El sistema está formado por la bobina, un núcleo ferromagnético y el aire que se encuentra dentro del cubo. Aunque son tres entidades independientes, todo el dominio dentro del cubo se encuentra mallado. Por ello, el conjunto se puede considerar un objeto sólido sin agujeros.

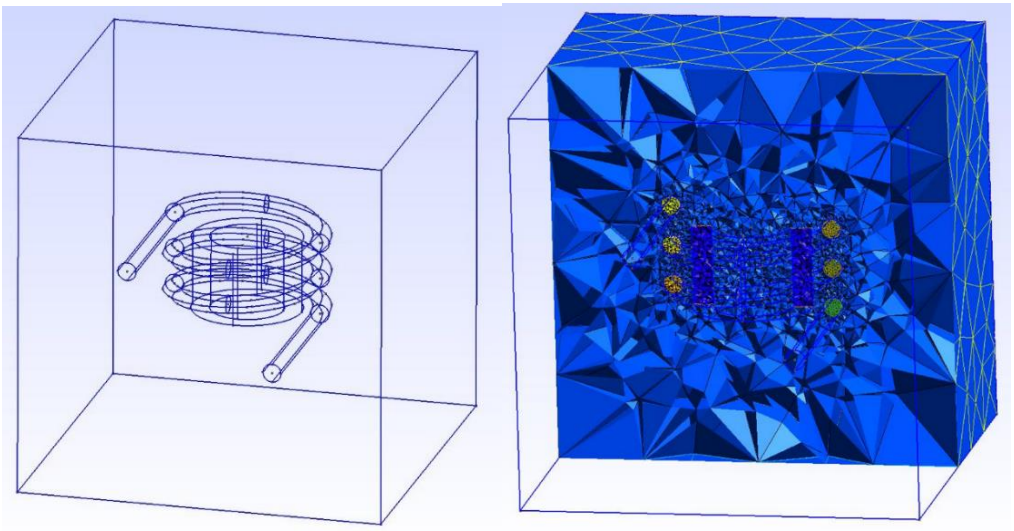

Figura 18. Bobina eléctrica con núcleo ferromagnético y aire.

Se ha escogido este ejemplo para explicar el objeto sólido tridimensional sin agujeros ya que es el modelo estudiado durante todo el trabajo. Al realizar la ejecución de los programas del pre-procesado, siempre se aseguraba que el modelo cumpliese con la fórmula de Euler-Poincaré, como en el siguiente caso:

Número de puntos (P): 2893

Número de líneas (L): 20022

Número de superficies (S): 34129

Número de volúmenes (V): 16999

Número de agujeros (g): 0

Sustituyendo en la fórmula de Euler-Poincaré para un objeto sólido sin agujeros (Ec. 3.3), se puede observar que se obtiene el resultado esperado:

$$2893 - 20022 + 34129 - 16999 = 1$$

3.7.2. Sólido tridimensional con un agujero - Toroide

En el caso de tener un sólido tridimensional con un agujero (g = 1), la constante de Euler-Poincaré vale 0 ($\chi = 1 - 1$). De esta forma, la fórmula de Euler-Poincaré queda de la siguiente forma:

$$P - L + S - V = 0 \qquad \text{Ec. 3.4}$$

Seguidamente se muestra el mallado de un toroide:

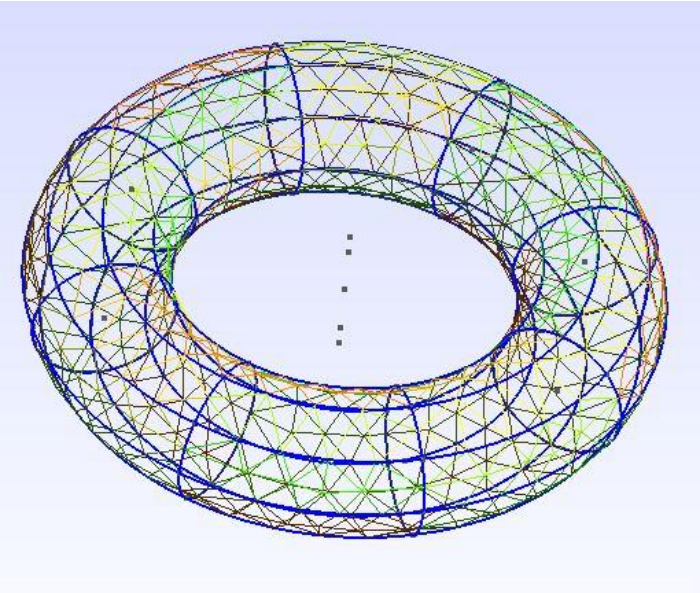

Figura 19. Mallado de un toroide.

En este caso, en el mallado se obtiene el siguiente número de elementos:

Número de puntos (P): 360

Número de líneas (L): 1899

Número de superficies (S): 2817

Número de volúmenes (V): 1278

Número de agujeros (g): 1

Sustituyendo en la fórmula de Euler-Poincaré para un objeto sólido con un agujero (Ec. 3.4), se puede observar que se obtiene el resultado nulo esperado:

$$360 - 1899 + 2817 - 1278 = 0$$

3.7.3. Sólido tridimensional con dos agujeros - Toroide doble

Para sólidos tridimensionales con dos agujero (g = 2), la constante de Euler-Poincaré vale -1 ($\chi = 1 - 2$). La fórmula de Euler-Poincaré resultante será la siguiente:

$$P - L + S - V = -1 \qquad \text{Ec. 3.5}$$

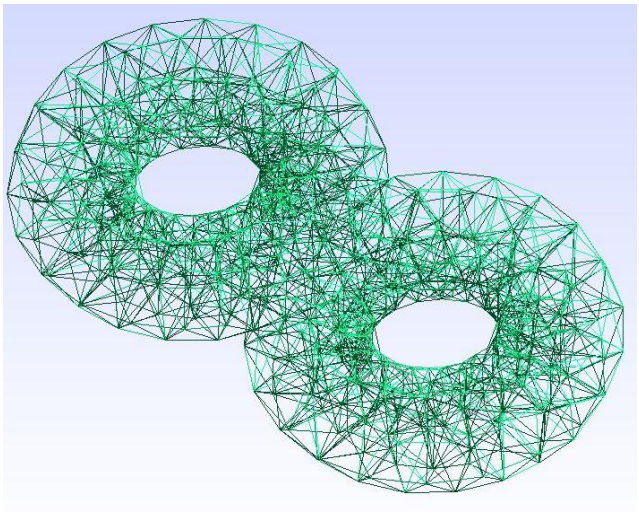

Figura 20. Mallado de un doble toroide.

El número de elementos que se obtiene del mallado es el siguiente:

Número de puntos (P): 748

Número de líneas (L): 4342

Número de superficies (S): 6736

Número de volúmenes (V): 3143

Número de agujeros (g): 2

Sustituyendo en la fórmula de Euler-Poincaré para un objeto sólido con dos agujeros (Ec. 3.5), se aprecia que se obtiene el resultado negativo esperado:

$$748 - 4342 + 6736 - 3143 = -1$$

3.7.4. Sólido tridimensional con tres agujeros - Toroide triple

Para sólidos tridimensionales con tres agujero (g = 3), la constante de Euler-Poincaré vale -2 ($\chi = 1 - 3$). La fórmula de Euler-Poincaré resultante será la siguiente:

$$P - L + S - V = -2 \qquad \text{Ec. 3.6}$$

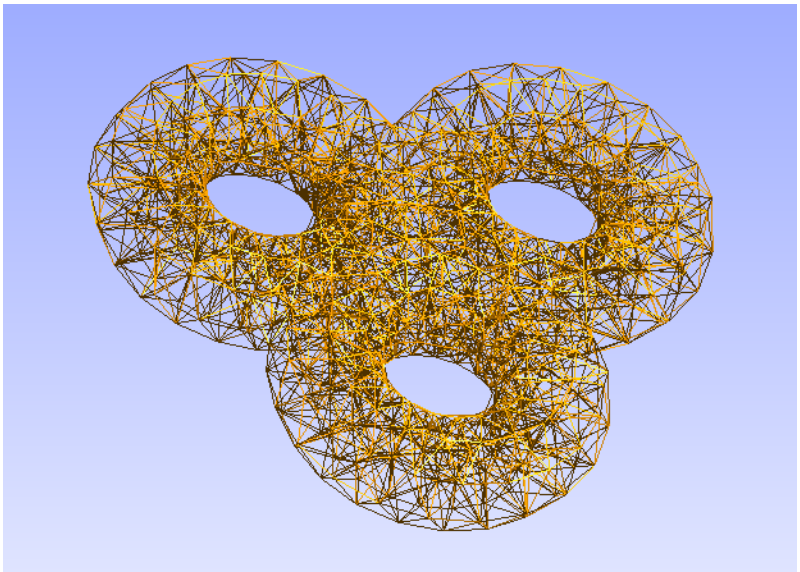

Figura 21. Mallado de un toroide triple.

El número de elementos que se obtiene del mallado es el siguiente:

Número de puntos (P): 1131

Número de líneas (L): 6653

Número de superficies (S): 10386

Número de volúmenes (V): 4866

Número de agujeros (g): 3

Sustituyendo en la fórmula de Euler-Poincaré para un objeto sólido con tres agujeros (Ec. 3.6), se comprueba que se cumple la ecuación:

$$1131 - 6653 + 10386 - 4866 = -2$$

3.8. Comparación con otros métodos numéricos

Existen diversas diferencias entre el método de la celda y otros métodos numéricos. Una de las principales distinciones es que el método de la celda no se fundamenta en la formulación diferencial, como hacen otros muchos métodos como el de los elementos finitos, el método de las diferencias finitas o el método de los volúmenes finitos [5].

Otra diferencia importante es que la formulación diferencial permite obtener el potencial en cualquier punto de su dominio y la formulación discreta solo lo permite en los puntos (0-cells) de su célula compleja. Sin embargo, en la práctica se puede obtener el valor del potencial en cualquier lugar dentro de un volumen (3-cell) utilizando funciones de interpolación.

Además, si se comparan los resultados del método de la celda con otros métodos numéricos, como el método de los elementos finitos, no siempre se obtienen los mismos valores. Cuando se tiene una fuente distribuida uniformemente en todo el dominio no hay

disparidades en los resultados, no obstante, cuando las fuentes no se distribuyen uniformemente, los resultados en ambos métodos no se corresponden exactamente.

3.9. Conclusiones

El método de la celda es un método numérico que no utiliza formulación diferencial ni integral, sino discreta, y que utiliza variables globales como punto de partida. Estas variables se pueden asociar a las células espaciotemporales del complejo primal o a las del dual, y tener orientación interna o externa. De esta forma, se construyen las matrices de incidencia que se emplearán en la resolución de ecuaciones electromagnéticas para poder solucionar así problemas físicos.

Capítulo 4. Leyes electromagnéticas en la Formulación Finita

4.1. Introducción

Las principales leyes del electromagnetismo se pueden reescribir dentro del marco de la formulación finita. Mediante la experimentación, las variables globales se relacionan con los elementos espaciales y temporales del método de la celda.

4.2. Ley de Gauss para el campo eléctrico

La ley de Gauss para el campo eléctrico demuestra que el flujo de campo eléctrico a través de la superficie de un volumen con orientación externa para un instante determinado es igual a la carga eléctrica contenida en el interior de ese volumen en ese preciso instante [28].

$$\Psi\left[t, \partial\widetilde{V}\right] = Q^{c}[t, \widetilde{V}] \qquad \text{Ec. 4.1}$$

4.3. Ley de Gauss para el campo magnético

La ley de Gauss para el campo magnético determina que el flujo magnético neto que atraviesa la superficie que contiene al volumen donde está la fuente de campo magnético es nulo, ya que el número de líneas de campo entrantes es igual al número de líneas salientes [2].

$$\Phi\left[t, \partial\widetilde{V}\right] = 0 \qquad \text{Ec. 4.2}$$

En formulación diferencial, la ley de Gauss para el campo magnético se expresa de la siguiente forma:

$$\vec{B} = \vec{\nabla}\times\vec{A} \qquad \text{Ec. 4.3}$$

Sin embargo, en formulación finita se utilizará la magnitud global del potencial magnético escalar *a*:

$$a = \int_{L} \vec{A} \cdot d\vec{L} \qquad \text{Ec. 4.4}$$

Sustituyendo el potencial magnético vectorial por el vector magnético escalar, y el rotacional por la matriz de incidencia C, se obtiene la expresión de la Ley de Gauss para el campo magnético en Formulación Finita:

$$[\Phi] = [C][a] \qquad \text{Ec. 4.5}$$

Multiplicando ambos miembros por la matriz de incidencia D se comprueba que el flujo magnético es nulo:

$$[\mathrm{D}][\Phi] = [\mathrm{D}][\mathrm{C}][\mathrm{a}] = 0[\mathrm{a}] = 0 \qquad \text{Ec. 4.6}$$

4.4. Ley de Faraday-Lenz-Neumann

La fuerza electromotriz inducida en los extremos de un hilo conductor es producida por las variaciones temporales del campo magnético que atraviesa la superficie con orientación interna de dicho hilo conductor.

$$\varepsilon\left[\tilde{\mathrm{T}}, \partial \mathrm{S}\right] = -\{\Phi[\tilde{\mathrm{t}}^{+}, \mathrm{S}] + \Phi[\tilde{\mathrm{t}}^{-}, \mathrm{S}]\} \qquad \text{Ec. 4.7}$$

En formulación diferencial, la ley de Faraday-Lenz-Neumann se puede expresar de la siguiente forma:

$$\vec{\mathrm{E}} = -\partial_t \vec{\mathrm{A}} \qquad \text{Ec. 4.8}$$

$$\vec{\nabla} \times \vec{\mathrm{E}} = \vec{\nabla} \times \left(-\partial_t \vec{\mathrm{A}}\right) \qquad \text{Ec. 4.9}$$

A la expresión anterior se le puede añadir un término adicional, que no variará el resultado, ya que el rotacional del gradiente es nulo:

$$\vec{\nabla} \times \vec{\mathrm{E}} = \vec{\nabla} \times \left(-\partial_t \vec{\mathrm{A}}\right) - \vec{\nabla} \times (\vec{\nabla} \mathcal{V}) \qquad \text{Ec. 4.10}$$

En la formulación finita, la rotación del campo eléctrico es sustituida por la fuerza electromotriz, los rotacionales por la matriz de incidencia C y los gradientes por la matriz de incidencia G:

$$[\mathrm{C}][\mathrm{U_e}] = [\mathrm{C}]\left\{-\frac{\mathrm{d}[\mathrm{a}]}{\mathrm{dt}} - [\mathrm{G}][\mathcal{V}]\right\} \qquad \text{Ec. 4.11}$$

4.5. Ley de Ampère-Maxwell

El impulso de fuerza magnetomotriz que aparece en la superficie con orientación externa en un intervalo de tiempo es igual a la suma de la producción de cargas eléctricas que atraviesan dicha superficie en el intervalo de tiempo y de la variación de flujo eléctrico en la ya mencionada superficie en el intervalo de tiempo.

$$\mathcal{F}\left[T, \partial \tilde{\mathrm{S}}\right] = \mathrm{Q}^{f}\left[T, \tilde{\mathrm{S}}\right] + \{\psi\left[t^{+}, \tilde{\mathrm{S}}\right] - \psi\left[t^{-}, \tilde{\mathrm{S}}\right]\} \qquad \text{Ec. 4.12}$$

En formulación diferencial, la ley de Ampère se puede expresar de la siguiente forma:

$$\vec{\nabla} \times \left(\mathcal{V} \vec{\nabla} \times \vec{\mathrm{A}}\right) = \vec{\mathrm{J}} \qquad \text{Ec. 4.13}$$

De la Ley de Faraday-Lenz-Neumann se puede deducir que:

$$\vec{\mathrm{E}} = -\partial_t \vec{\mathrm{A}} - \vec{\nabla} \mathcal{V} \qquad \text{Ec. 4.14}$$

Conociendo la siguiente relación constitutiva:

$$\vec{\mathrm{J}} = \sigma\vec{\mathrm{E}} \qquad \text{Ec. 4.15}$$

Si se sustituye el campo eléctrico en la expresión anterior se tiene la siguiente ecuación:

$$\vec{\mathrm{J}} = \sigma(-\partial_t\vec{\mathrm{A}} - \vec{\nabla}\mathcal{V}) \qquad \text{Ec. 4.16}$$

Sustituyendo la expresión anterior en la ley de Ampère diferencial se obtiene la siguiente ecuación:

$$\vec{\nabla}\times\left(\mathcal{V}\vec{\nabla}\times\vec{\mathrm{A}}\right) = \sigma(-\partial_t\vec{\mathrm{A}} - \vec{\nabla}\mathcal{V}) \qquad \text{Ec. 4.17}$$

Para pasar la expresión anterior a formulación diferencial se debe sustituir el potencial magnético vectorial por el vector magnético escalar, los rotacionales por matrices de incidencia C, los gradientes por matrices de incidencia G, el potencial por la matriz constitutiva magnética y la conductividad eléctrica por la matriz constitutiva eléctrica, obteniendo la siguiente ecuación:

$$[\tilde{\mathrm{C}}]\{[\mathrm{M}_{\mathcal{V}}][\mathrm{C}][\mathrm{a}]\} = -[\mathrm{M}_{\sigma}]\left\{\frac{\mathrm{d}[\mathrm{a}]}{\mathrm{dt}}[\mathrm{G}][\mathcal{V}]\right\} \qquad \text{Ec. 4.18}$$

Conociendo las propiedades de las matrices de incidencias se tiene la expresión final de la ley de Ampére:

$$[\mathrm{C}^{\mathrm{T}}]\{[\mathrm{M}_{\mathcal{V}}][\mathrm{C}][\mathrm{a}]\} = -[\mathrm{M}_{\sigma}]\left\{\frac{\mathrm{d}[\mathrm{a}]}{\mathrm{dt}}[\mathrm{G}][\mathcal{V}]\right\} \qquad \text{Ec. 4.19}$$

4.6. Matrices constitutivas

En la formulación finita, una ecuación constitutiva relaciona las variables de configuración y las variables de fuente. Además, explica el comportamiento de la materia cuando es atravesada por campos electromagnéticos. En el Método de la Celda se trabaja con matrices constitutivas, como se ha podido comprobar en la Ley de Ampère.

La matriz constitutiva magnética permite relacionar la variable de configuración de flujo magnético, con la variable de fuente de la fuerza magnetomotriz, según la siguiente expresión:

$$[\mathcal{F}_{\mathrm{m}}]^{\mathrm{L}}_{4\mathrm{x}1} = [\mathrm{M}_{\mathcal{V}}]^{\mathrm{L}}_{4\mathrm{x}4}[\Phi]^{\mathrm{L}}_{4\mathrm{x}1} \qquad \text{Ec. 4.20}$$

El superíndice L indica que es una matriz de celda. En formulación diferencial, la ecuación equivalente es la siguiente:

$$\vec{\mathrm{H}} = \mathcal{V}\vec{\mathrm{B}} \qquad \text{Ec. 4.21}$$

Por ello, en la Ley de Ampère se pudo sustituir el potencial por la matriz constitutiva magnética.

La matriz constitutiva eléctrica permite relacionar la variable de configuración de tensión eléctrica, con la variable de fuente de intensidad eléctrica, según la siguiente expresión:

$$[\mathrm{I}] = [\mathrm{M}_{\sigma}][\mathrm{U}] \qquad \text{Ec. 4.22}$$

En formulación diferencial, la ecuación equivalente es la siguiente:

$$\vec{J} = \sigma\vec{E} \qquad \text{Ec. 4.23}$$

Por eso en la Ley de Ampere se puede sustituir la conductividad eléctrica por la matriz constitutiva eléctrica.

4.7. Ecuaciones de Maxwell y circuitales en la formulación finita

Las ecuaciones circuitales definen circuitos eléctricos con elementos de parámetros concentrados. El método de la tabla, el método de la tabla reducida y el método nodal modificado se pueden adaptar a la formulación finita.

4.7.1. Método de la tabla

El método de la tabla debe cumplir con la primera ley de Kirchhoff, la segunda ley de Kirchhoff y la ecuación de definición.

Primera ley de Kirchhoff:

$$[A][I_b] = [0] \qquad \text{Ec. 4.24}$$

Segunda ley de Kirchhoff:

$$[U_b] = [A]^T[\mathcal{V}_n] \qquad \text{Ec. 4.25}$$

Ecuación constitutiva:

$$[Y_b][U_b] + [Z_b][I_b] = [W_b] \qquad \text{Ec. 4.26}$$

Donde:

$[A]$ es la matriz de incidencias nudos rama,

$[I_b]$ es la matriz de corrientes,

$[U_b]$ es la matriz de tensiones,

$[\mathcal{V}_n]$ es la matriz de potenciales en los nudos,

$[Y_b]$ es la matriz de admitancias,

$[Z_b]$ es la matriz de impedancias,

$[W_b]$ es la matriz de fuentes de tensión.

Para formar el sistema de ecuaciones, primero se deberá conocer las incógnitas a resolver. En este caso, las incógnitas serán la matriz de tensiones, la matriz de corrientes y la matriz de potenciales en los nudos.

Para obtener estas incógnitas se utilizará un sistema de ecuaciones basado en la primera ley de Kirchhoff, la segunda ley de Kirchhoff y la ecuación constitutiva. Sin embargo, antes de

formar el sistema de ecuaciones, es necesario ordenar correctamente la segunda ley de Kirchhoff (Ec. 4.25):

$$[\mathrm{U_b}] - [\mathrm{A}]^{\mathrm{T}}[\mathcal{V}_\mathrm{n}] = [0] \qquad \text{Ec. 4.27}$$

Entonces, el sistema de ecuaciones quedaría como se puede observar a continuación:

$$\begin{pmatrix} [1] & [0] & -[\mathrm{A}]^{\mathrm{T}} \\ [0] & [\mathrm{A}] & [0] \\ [\mathrm{Y_b}] & [\mathrm{Z_b}] & [0] \end{pmatrix} \begin{pmatrix} [\mathrm{U_b}] \\ [\mathrm{I_b}] \\ [\mathcal{V}_\mathrm{n}] \end{pmatrix} = \begin{pmatrix} [0] \\ [0] \\ [\mathrm{W_b}] \end{pmatrix} \qquad \text{Ec. 4.28}$$

4.7.2. Método de la tabla reducido

El método de la tabla reducido es una simplificación del método de la tabla original. Para ello, es necesario sustituir la matriz de tensiones de la segunda ley de Kirchhoff (ec. 4.25) en la ecuación constitutiva (Ec. 4.26):

$$[\mathrm{Y_b}][\mathrm{A}]^{\mathrm{T}}[\mathcal{V}_\mathrm{n}] + [\mathrm{Z_b}][\mathrm{I_b}] = [\mathrm{W_b}] \qquad \text{Ec. 4.29}$$

De esta forma, el sistema de ecuaciones final queda simplificado:

$$\begin{pmatrix} [\mathrm{Y_b}][\mathrm{A}]^{\mathrm{T}} & [\mathrm{Z_b}] \\ [0] & [\mathrm{A}] \end{pmatrix} \begin{pmatrix} [\mathcal{V}_\mathrm{n}] \\ [\mathrm{I_b}] \end{pmatrix} = \begin{pmatrix} [\mathrm{W_b}] \\ [0] \end{pmatrix} \qquad \text{Ec. 4.30}$$

4.7.3. Método Nodal Modificado

En el Método Nodal Modificado existen tres conjuntos de elementos:

A_1: elementos que cumplan con la característica $[\mathrm{I}] = [\mathrm{Y}][\mathrm{U}]$, como condensadores, fuentes de corriente dependientes de tensión o conductancias.

A_2: elementos que cumplan con la característica $[\mathrm{Y}][\mathrm{U}] + [\mathrm{Z}][\mathrm{I}] = [\mathrm{W}]$, como resistencias, bobinas o fuentes de tensión.

A_3: elementos que cumplan con la característica $[\mathrm{I}] = [\mathrm{J}]$, es decir, fuentes independientes de corriente.

Para cumplir con la segunda ley de Kirchhoff, también se debe cumplir la siguiente ecuación:

$$[\mathrm{U_b}] = [[\mathrm{A}_1] \quad [\mathrm{A}_2] \quad [\mathrm{A}_3]]^{\mathrm{T}} \cdot [\mathcal{V}_\mathrm{n}] \qquad \text{Ec. 4.31}$$

Teniendo en cuenta las propiedades de la matriz transpuesta, se tiene la siguiente expresión:

$$\begin{pmatrix} [\mathrm{U}_1] \\ [\mathrm{U}_2] \\ [\mathrm{U}_3] \end{pmatrix} = \begin{pmatrix} [\mathrm{A}_1]^T \\ [\mathrm{A}_2]^T \\ [\mathrm{A}_3]^T \end{pmatrix} \cdot [\mathcal{V}_\mathrm{n}] \qquad \text{Ec. 4.32}$$

En el Método Nodal Modificado también se aplica la primera ley de Kirchhoff:

$$[[A_1] \quad [A_2] \quad [A_3]] \cdot \begin{bmatrix} [I_1] \\ [\mathrm{I}_2] \\ [\mathrm{I}_3] \end{bmatrix} = [0] \qquad \text{Ec. 4.33}$$

$$[A_1] \cdot [I_1] + [A_2] \cdot [I_2] + [A_3] \cdot [I_3] = [0] \qquad \text{Ec. 4.34}$$

Para eliminar el mayor número posible de incógnitas, los términos A_1, A_3, I_1, I_3 se reducirán a términos de I_2 y $\mathcal{V}_\mathrm{n}$. Conociendo las características que deben cumplir los elementos A_1 ($[\mathrm{I}] = [\mathrm{Y}][\mathrm{U}]$) y A_3 ($[\mathrm{I}] = [\mathrm{J}]$) se puede reducir la expresión anterior (Ec. 4.34) de la siguiente forma:

$$[A_1] \cdot [\mathrm{Y}_1][\mathrm{U}_1] + [A_2] \cdot [I_2] + [A_3] \cdot [I_J] = [0] \qquad \text{Ec. 4.35}$$

$$[A_1] \cdot [\mathrm{Y}_1][\mathrm{U}_1] + [A_2] \cdot [I_2] = -[A_3] \cdot [I_J] \qquad \text{Ec. 4.36}$$

Al sustituir el término U_1de la segunda ley de Kirchhoff (Ec. 4.25), se puede simplificar aún más la expresión anterior (Ec. 4.36):

$$[\mathrm{A}_1] \cdot [\mathrm{Y}_1][\mathrm{A}_1]^\mathrm{T} \cdot [\mathcal{V}_\mathrm{n}] + [\mathrm{A}_2] \cdot [\mathrm{I}_2] = -[\mathrm{A}_3] \cdot [\mathrm{I}_\mathrm{J}] \qquad \text{Ec. 4.37}$$

Finalmente, a la ecuación constitutiva del Método de la Tabla (Ec. 4.26) se le puede sustituir el término $[\mathrm{U}_2]$ de la segunda ley de Kirchhoff (Ec. 4.25):

$$[\mathrm{Y}_2][\mathrm{A}_2]^T \cdot [\mathcal{V}_\mathrm{n}] + [\mathrm{Z}_2][\mathrm{I}_2] = [\mathrm{W}_2] \qquad \text{Ec. 4.38}$$

De esta forma, se puede construir el sistema de ecuaciones del Método Nodal Modificado:

$$\begin{pmatrix} [\mathrm{A}_1] \cdot [\mathrm{Y}_1][\mathrm{A}_1]^\mathrm{T} & [\mathrm{A}_2] \\ [\mathrm{A}_2]^\mathrm{T} & [\mathrm{Z}_2] \end{pmatrix} \begin{pmatrix} [\mathcal{V}_\mathrm{n}] \\ [\mathrm{I}_2] \end{pmatrix} = \begin{pmatrix} -[\mathrm{A}_3] \cdot [\mathrm{I}_\mathrm{J}] \\ [\mathrm{W}_2] \end{pmatrix} \qquad \text{Ec. 4.39}$$

4.7.4. Incorporación de elementos electromagnéticos de dominio continuo al sistema de ecuaciones circuitales del Método Nodal Modificado

En este método se añaden elementos de un continuo discretizado a un circuito de elementos discretos concentrados como es el Método Nodal Modificado. En primer lugar, se añade el elemento A_4, una matriz de incidencias nudos-ramas donde se incluyen las corrientes cohomológicas.

$$\begin{pmatrix} [\mathrm{U}_1] \\ [\mathrm{U}_2] \\ [\mathrm{U}_3] \\ [\mathrm{U}_4] \end{pmatrix} = \begin{pmatrix} [\mathrm{A}_1]^T \\ [\mathrm{A}_2]^T \\ [\mathrm{A}_3]^T \\ [\mathrm{A}_4]^T \end{pmatrix} \cdot [\mathcal{V}_\mathrm{n}] \qquad \text{Ec. 4.40}$$

$$[\mathrm{A}_1] \cdot [\mathrm{I}_1] + [\mathrm{A}_2] \cdot [\mathrm{I}_2] + [\mathrm{A}_3] \cdot [\mathrm{I}_3] + [\mathrm{A}_4] \cdot [\mathrm{I}_4] = [0] \qquad \text{Ec. 4.41}$$

Al incorporar un nuevo conjunto de elementos, también se debe definir la característica que deben cumplir, que en este caso será la siguiente:

$$[\mathrm{I}_\Omega] = [\mathrm{M}_\sigma] \cdot \{-\mathrm{j}\omega[\mathrm{a}] - [\mathrm{G}][\mathcal{V}_\mathrm{i}]\} \qquad \text{Ec. 4.42}$$

Se añade el término (jω) ya que el sistema varía armónicamente. $\mathcal{V}_{\mathrm{i}}$ representa los potenciales eléctricos en el dominio discretizado Ω.

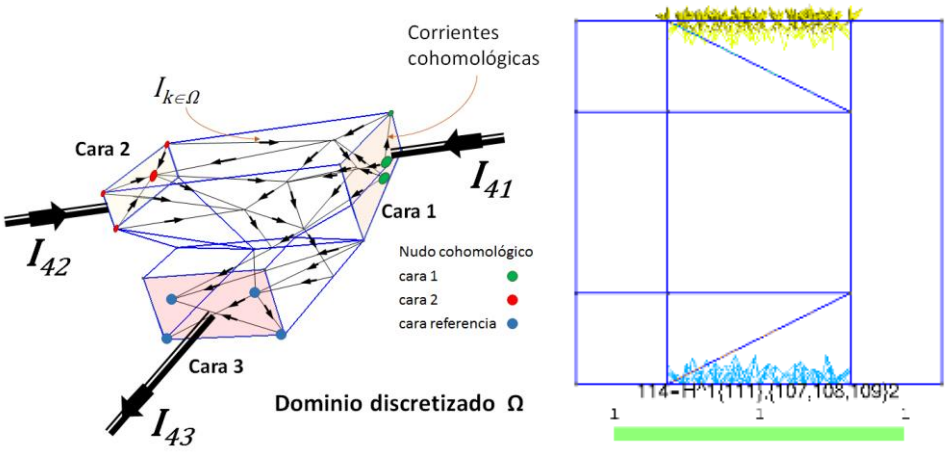

Figura 22. Dominio discretizado con corrientes cohomológicas.

En la figura 22 se muestra el dominio discretizado con corrientes cohomológicas, y las corrientes $[\mathrm{I}_{41}]$, $[\mathrm{I}_{42}]$ e $[\mathrm{I}_{43}]$, que son las corrientes que pertenecen al circuito de parámetros discretos concentrados y que comparten el dominio continuo discretizado Ω. Estas corrientes deben cumplir la siguiente relación:

$$[\mathrm{I}_{41}] + [\mathrm{I}_{42}] = [\mathrm{I}_{43}] \qquad \text{Ec. 4.43}$$

Para obtener el valor de estas corrientes es necesario construir la matriz de corrientes cohomológicas, que tendrá tantas filas como caras cohomológicas tenga el objeto, y tantas columnas como aristas tenga el mallado del dominio discretizado. Según si las aristas inciden en la cara comohológica o en el resto del mallado, a la matriz se le asigna el valor nulo si no incide, 1 si incide en sentido esperado o -1 si incide en sentido contrario al esperado. De esta forma, se obtendría la siguiente matriz para el caso expuesto:

$$[\mathrm{I_c}] = \begin{bmatrix} 1 & \dots & \dots & -1 & 0 & 1 \\ -1 & \dots & \dots & 0 & 0 & 0 \end{bmatrix}_{n4xna} \qquad \text{Ec. 4.44}$$

La corriente que se desea obtener es la siguiente:

$$[\mathrm{I_x}]_{lx1} = [\mathrm{M}_\sigma] \cdot \{-\mathrm{j}\omega[\mathrm{a}] - [\mathrm{G}][\mathcal{V}_{\mathrm{i}}]\} \qquad \text{Ec. 4.45}$$

Operando se obtiene la corriente del grupo 4:

$$[\mathrm{I_c}]_{n4xl} \cdot [\mathrm{I_x}]_{lx1} = [\mathrm{I_4}]_{n4x1} \qquad \text{Ec. 4.46}$$

Sustituyendo la ecuación 4.45 en la ecuación anterior (Ec. 4.46) se obtiene la siguiente expresión:

$$-\mathrm{j}\omega[\mathrm{I_c}][\mathrm{M}_\sigma][\mathrm{a}] - [\mathrm{I_c}][\mathrm{M}_\sigma][\mathrm{G}][\mathcal{V}_{\mathrm{i}}] - [\mathrm{I_4}] = [0] \qquad \text{Ec. 4.47}$$

Si se combina la expresión anterior (Ec. 4.47) con la Ley de Ampere (Ec. 4.19), se obtiene la siguiente ecuación:

$$\{[\mathrm{C}]^{\mathrm{T}}[\mathrm{M}_{\mathcal{V}}][\mathrm{C}] + \mathrm{j}\omega[\mathrm{M}_{\sigma}]\}[\mathrm{a}] + [\mathrm{M}_{\sigma}][\mathrm{G}][\mathcal{V}_{\mathrm{i}}] = [\mathrm{I}_{\mathrm{f}}] \qquad \text{Ec. 4.48}$$

Donde la intensidad $[\mathrm{I}_{\mathrm{f}}]$ corresponden a las corrientes suministradas por el circuito discreto, que circulan conjuntamente con las corrientes inducidas dentro del dominio continuo discretizado.

Seguidamente se muestra la ley de Ampere (Ec. 4.19) junto con la ley de Faraday (Ec. 4.11) en el dominio continuo discretizado:

$$-\mathrm{j}\omega[G]^{T}[\mathrm{M}_{\sigma}][\mathrm{a}] - [G]^{T}[\mathrm{M}_{\sigma}][\mathrm{G}][\mathcal{V}_{\mathrm{i}}] = [0] \qquad \text{Ec 4.49}$$

Al igual que se hizo en el Método Nodal Modificado, se puede reducir el número de incógnitas de la ecuación de continuidad al sustituir las características que deben cumplir los elementos A_1 ($[\mathrm{I}] = [\mathrm{Y}][\mathrm{U}]$) y A_3 ($[\mathrm{I}] = [\mathrm{J}]$), junto con la segunda ley de Kirchhoff (Ec. 4.25) en la ecuación 4.41:

$$[\mathrm{A}_1] \cdot [\mathrm{Y}_1][\mathrm{A}_1]^{\mathrm{T}} \cdot [\mathcal{V}_{\mathrm{n}}] + [\mathrm{A}_2] \cdot [\mathrm{I}_2] + [\mathrm{A}_4] \cdot [\mathrm{I}_4] = -[\mathrm{A}_3] \cdot [\mathrm{I}_{\mathrm{J}}] \qquad \text{Ec. 4.50}$$

Finalmente, de la ecuación característica de los elementos del grupo A_2 y de la segunda ley de Kirchhoff (Ec. 4.25) se puede obtener de nuevo la ecuación 4.38. De esta forma, se tienen todas las ecuaciones que conforman el sistema de ecuaciones de este método:

$$\{[\mathrm{C}]^{\mathrm{T}}[\mathrm{M}_{\mathcal{V}}][\mathrm{C}] + \mathrm{j}\omega[\mathrm{M}_{\sigma}]\}[\mathrm{a}] + [\mathrm{M}_{\sigma}][\mathrm{G}][\mathcal{V}_{\mathrm{i}}] = [\mathrm{I}_{\mathrm{f}}] \qquad \text{Ec. 4.48}$$

$$-\mathrm{j}\omega[G]^{T}[\mathrm{M}_{\sigma}][\mathrm{a}] - [G]^{T}[\mathrm{M}_{\sigma}][\mathrm{G}][\mathcal{V}_{\mathrm{i}}] = [0] \qquad \text{Ec. 4.49}$$

$$-\mathrm{j}\omega[\mathrm{I}_{\mathrm{c}}][\mathrm{M}_{\sigma}][\mathrm{a}] - [\mathrm{I}_{\mathrm{c}}][\mathrm{M}_{\sigma}][\mathrm{G}][\mathcal{V}_{\mathrm{i}}] - [\mathrm{I}_4] = [0] \qquad \text{Ec. 4.47}$$

$$[\mathrm{Y}_2][\mathrm{A}_2]^{T} \cdot [\mathcal{V}_{\mathrm{n}}] + [\mathrm{Z}_2][\mathrm{I}_2] = [\mathrm{W}_2] \qquad \text{Ec. 4.38}$$

$$[\mathrm{A}_1] \cdot [\mathrm{Y}_1][\mathrm{A}_1]^{\mathrm{T}} \cdot [\mathcal{V}_{\mathrm{n}}] + [\mathrm{A}_2] \cdot [\mathrm{I}_2] + [\mathrm{A}_4] \cdot [\mathrm{I}_4] = -[\mathrm{A}_3] \cdot [\mathrm{I}_{\mathrm{J}}] \qquad \text{Ec. 4.50}$$

El sistema de ecuaciones anterior se puede expresar en forma matricial:

$$\begin{bmatrix} [T_{11}]_{lxl} & [T_{11}]_{lx\eta} & [0]_{lxn} & [0]_{lxb2} & [0]_{lxb2} \\ [T_{11}]_{nxl} & [T_{11}]_{nx\eta} & [0]_{\eta xn} & [0]_{nxb2} & [0]_{nxb4} \\ [T_{11}]_{b4xl} & [T_{11}]_{b4x\eta} & [0]_{b4xn} & [0]_{b4xb2} & [-1]_{b4xb4} \\ [0]_{b2xl} & [0]_{b2x\eta} & [T_{11}]_{b2xn} & [Z_2]_{b2xb2} & [0]_{b2xb4} \\ [0]_{nxl} & [0]_{nx\eta} & [T_{11}]_{nxn} & [A_2]_{nxb2} & [A_4]_{nxb4} \end{bmatrix} \times \begin{bmatrix} [a]_{lx1} \\ [v_i]_{\eta x1} \\ [v_n]_{nx1} \\ [I_2]_{b2x1} \\ [I_4]_{b4x1} \end{bmatrix} = \begin{bmatrix} [I_f]_{lx1} \\ [0]_{\eta x1} \\ [0]_{nx1} \\ [W_2]_{b2x1} \\ [-A_3\ I_j]_{b4x1} \end{bmatrix}$$

Ec. 4.51

Debido a la complejidad del sistema, el sistema completo se muestra en la siguiente página con orientación horizontal.

.

$$
\begin{bmatrix}
[\{[C]^T[M_v][C] + j\omega\,[M_\sigma]\}]_{lxl} & [[M_\sigma][G]]_{lx\eta} & [0]_{lxn} & [0]_{lxb2} & [0]_{lxb2} \\
[-j\omega\,[G]^T[M_\sigma]]_{nxl} & [-[G]^T[M_\sigma][G]]_{nx\eta} & [0]_{\eta xn} & [0]_{nxb2} & [0]_{nxb4} \\
[-j\omega\,[I_c][M_\sigma]]_{b4xl} & [[I_c][M_\sigma][G]]_{b4x\eta} & [0]_{b4xn} & [0]_{b4xb2} & [-1]_{b4xb4} \\
[0]_{b2xl} & [0]_{b2x\eta} & [[Y_2][A_2]^T]_{b2xn} & [Z_2]_{b2xb2} & [0]_{b2xb4} \\
[0]_{nxl} & [0]_{nx\eta} & [[A_1][Y_1][A_1]^T]_{nxn} & [A_2]_{nxb2} & [A_4]_{nxb4}
\end{bmatrix}
\times
\begin{bmatrix}
[a]_{lx1} \\ [v_i]_{\eta x1} \\ [v_n]_{nx1} \\ [I_2]_{b2x1} \\ [I_4]_{b4x1}
\end{bmatrix}
=
\begin{bmatrix}
[I_f]_{lx1} \\ [0]_{\eta x1} \\ [0]_{nx1} \\ [W_2]_{b2x1} \\ [-A_3\,I_j]_{b4x1}
\end{bmatrix}
$$

Ec. 4.52

4.8. Conclusiones

El método de la celda es un método numérico que no se basa en la formulación diferencial en la que se suelen asentar las ecuaciones de la física, sino en formulación finita. Por ello, es necesario reescribir todas las leyes y ecuaciones del electromagnetismo que se vayan a emplear, como las ecuaciones de Maxwell y las ecuaciones circuitales, desde un punto de vista finito, utilizando para ello matrices de incidencia y matrices constitutivas.

Capítulo 5. Mallado

El mallado del sistema es un aspecto fundamental del proceso, ya que permite dividir el dominio a estudiar en elementos compatibles con el método de la celda, como puntos, líneas, superficies y volúmenes. De esta forma, se relaciona el sistema que se quiere analizar con el método numérico a emplear, lo que permite obtener información acerca de los elementos físicos que lo componen y construir matrices de incidencia con las que trabajar posteriormente en el procesado del sistema. Para obtener dicho mallado se emplea el software Gmsh.

5.1. Gmsh

Gmsh es un generador tridimensional de mallados de elementos finitos con un motor CAD y post-procesador incorporados. Su objetivo es proporcionar una herramienta de mallado rápida, ligera y fácil de usar, que cuenta con un sistema de entrada paramétrica y un sistema de visualización avanzado [29]. Gmsh se compone de cuatro módulos básicos: geometría, mallado, solver y post-procesamiento.

5.1.1. Geometría

Gmsh emplea una representación de contorno para describir geometrías. Se crean modelos en un flujo "bottom-up" mediante la definición sucesiva de puntos, líneas orientadas (segmentos de línea, círculos, elipses...), superficies orientadas (superficies planas, superficies triangulares...) y volúmenes. También se pueden definir grupos de entidades geométricas (llamadas "grupos físicos"), basados en estas entidades geométricas elementales. El lenguaje de programación de Gmsh permite parametrizar todas las entidades geométricas [25].

5.1.2. Mallado

Gmsh obtiene un mallado de elementos finitos que es un teselado de un subconjunto dado un espacio tridimensional. Este teselado se construye a partir de elementos geométricos elementales de varias formas (líneas, triángulos, cuadrángulos, tetraedros, prismas, hexaedros y pirámides), dispuestos de tal manera que, si dos de ellos se intersecan, lo hacen a lo largo de una cara, un borde o un nodo, y nunca de otra manera.

Todas los mallados de elementos finitos producidas por Gmsh se consideran "no estructurados", incluso si se generaron de una manera "estructurada" (por ejemplo, por extrusión). Esto implica que los elementos geométricos elementales se definen sólo por una lista ordenada de sus nodos, pero que no se asume una relación de orden predefinida entre dos elementos.

La generación del mallado se realiza de forma similar que la creación de geometría, siguiendo el flujo "bottom-up": Las líneas se discretizan primero; el mallado de las líneas se utiliza entonces para mallar las superficies; finalmente el mallado de las superficies se utiliza para mallar los volúmenes.

5.1.3. Solver

El solver predeterminado de Gmsh es GetDP, un solver libre de elementos finitos que utiliza elementos mixtos para discretizar los complejos de tipo Rham [30] en una, dos y tres dimensiones. Es capaz de solucionar los problemas discretos definidos en la información de entrada escrita en ficheros de datos ASCII al relacionarlos con las expresiones matemáticas simbólicas de dichos problemas.

Gmsh también puede utilizar solvers externos conectados con Gmsh a través de sockets Unix o TCP / IP, que permite modificar parámetros del solver, iniciar cálculos externos y procesar los resultados directamente desde dentro del módulo de post-procesamiento de Gmsh.

5.1.4. Post-procesado

Gmsh puede cargar y manipular múltiples mapas de post-procesamiento escalares, vectoriales o tensoriales junto con la geometría y el mallado del modelo. Los campos escalares están representados por líneas o superficies de valores semejantes o mapas de color, mientras que los campos vectoriales están representados por flechas tridimensionales o por mapas de desplazamiento.

5.2. Ejemplos

A continuación, se muestran una serie de ejemplos de sistemas simulados en Gmsh en los que se podrán observar los distintos módulos del software.

5.2.1. Cubo con agujero

Este es un ejemplo de simulación muy sencillo, para comenzar a manejar el software. Consiste en un cubo al que se le ha realizado un agujero. Seguidamente se mostrará la geometría del sistema:

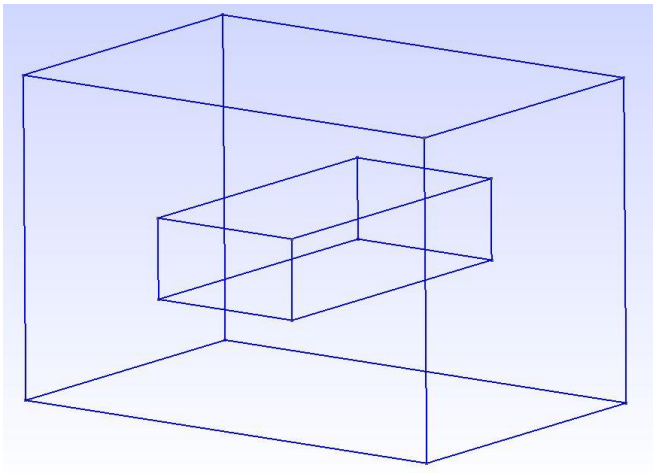

Figura 23. Geometría del cubo con agujero.

Este sistema geométrico se puede mallar en 3D, como se puede observar en la figura 24:

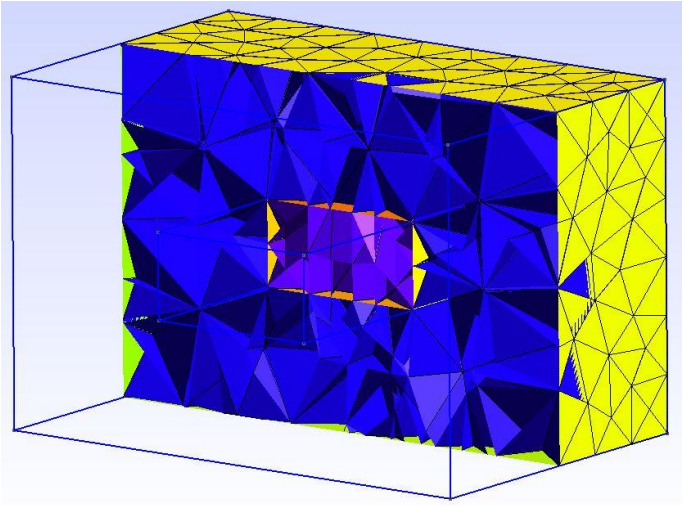

Figura 24. Mallado del cubo con agujero.

El sistema del cubo con agujero está mallado completamente. Sin embargo, en la imagen anterior se ha realizado un corte en la visualización para que se pueda apreciar mejor la totalidad del sistema. Como se puede apreciar en dicha imagen, el agujero también se encuentra mallado, pero como una entidad distinta al cubo. Esta aplicación es fundamental en sistemas más complejos con diversos materiales. En esos casos, cada material se mallará como una entidad propia e independiente una de otra.

5.2.2. Bobina eléctrica

El sistema de la bobina es de suma importancia en este trabajo, ya que ha sido el ejemplo utilizado en la paralelización de los algoritmos del pre-procesado y en la comparación de los resultados en los tiempos de ejecución. A continuación, se muestra la geometría de dicho sistema:

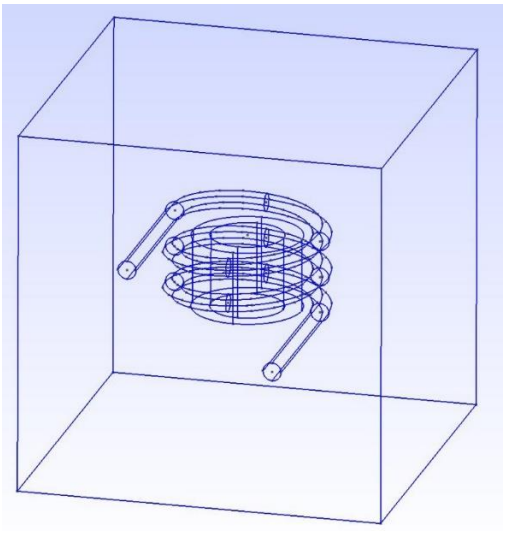

Figura 25. Geometría de la bobina eléctrica.

En la figura 25 se muestra la geometría del sistema de la bobina eléctrica. Como se puede observar, existen tres regiones distintas: la bobina, el núcleo ferromagnético, y el aire

que se encuentra dentro del cubo alrededor de la bobina y del núcleo ferromagnético. Seguidamente, en la figura 26 se muestra el mallado del sistema:

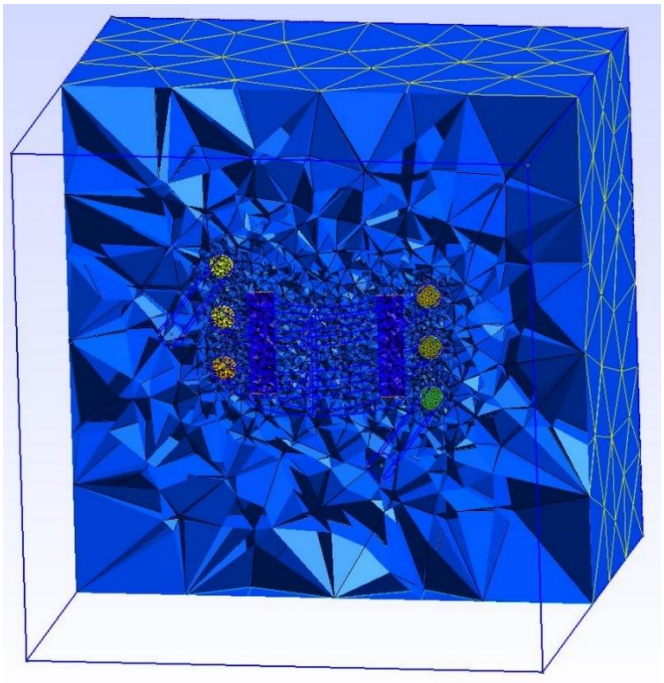

Figura 26. Mallado del sistema de la bobina eléctrica.

Al ser un sistema más complejo que el cubo con agujero, el corte del mallado no es suficiente para su correcta visualización. Por ello, en la figura 27 se muestra el mallado de la bobina y el núcleo ferromagnético, eliminando el aire de la visualización:

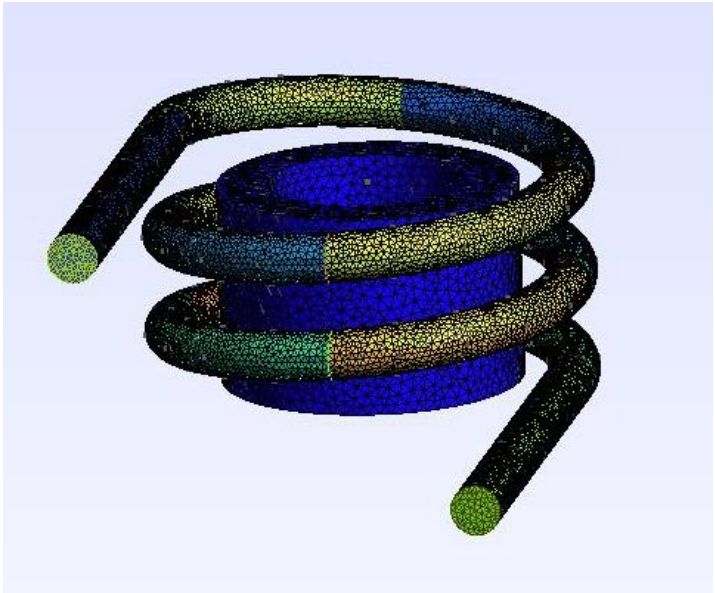

Figura 27. Mallado de la bobina eléctrica y el núcleo ferromagnético.

Al eliminar el aire de la visualización se puede apreciar mejor el detalle del mallado del resto del sistema. A continuación, se mostrará una comparación del detalle de los mallados de la bobina para distinto número de nudos en la simulación:

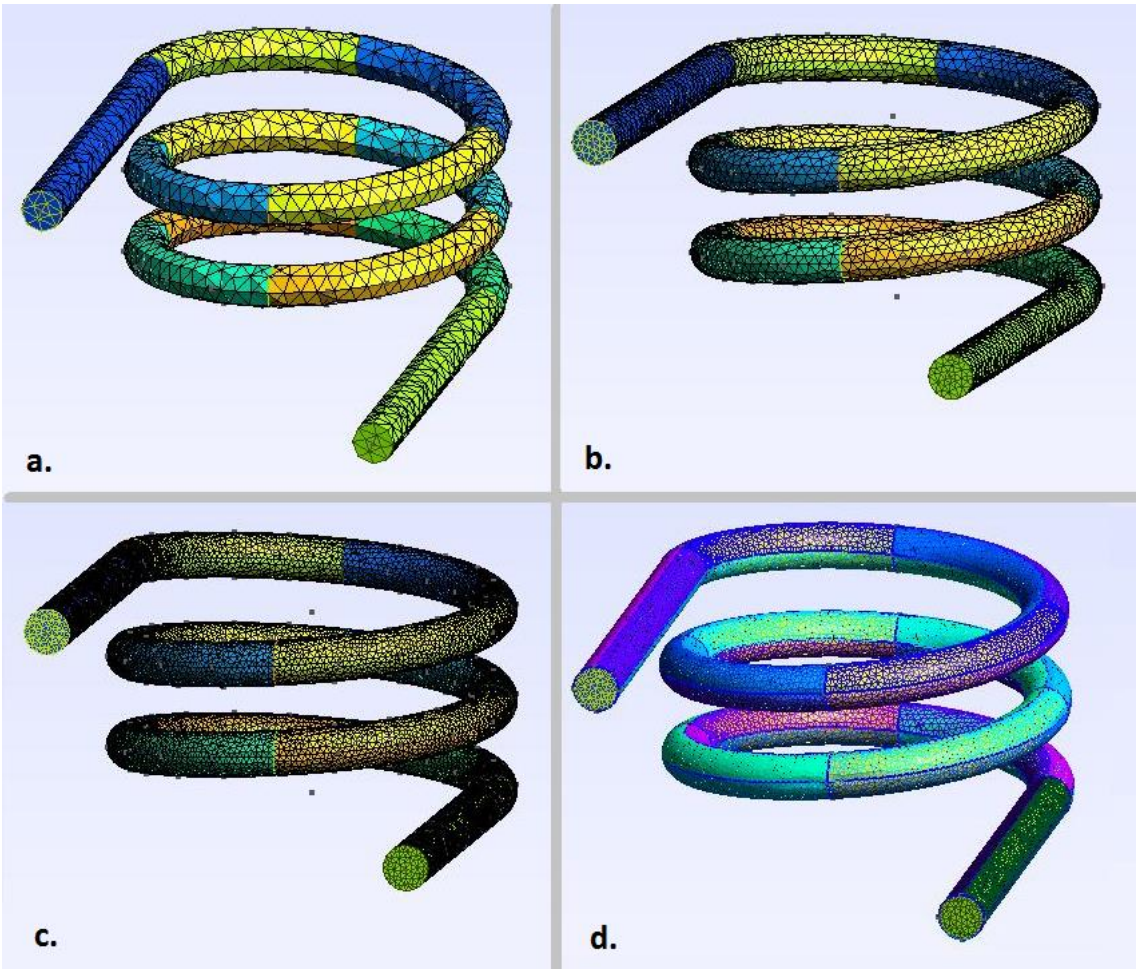

Figura 28. Comparación del mallado de la bobina para distinto número de nodos.

En la figura 28 se puede apreciar claramente como aumenta el detalle del mallado de una imagen a otra. El mallado de la bobina a. tiene 2893 nodos, el mallado de la bobina b. tiene 9778 nodos, el mallado de la bobina c. tiene 25534 nodos, y el mallado de la bobina d. tiene 42250. Como era de esperar, al aumentar el número de nodos del mallado, aumenta el detalle del mismo.

Anteriormente se han explicado ejemplos de los módulos del Gmsh de geometría y mallado, por lo que falta por mostrar el módulo del post-procesado, una vez ejecutado el módulo del solver. Por ello, en la siguiente imagen se podrá observar el post-procesado del cálculo de la densidad de corriente en el sistema de la bobina:

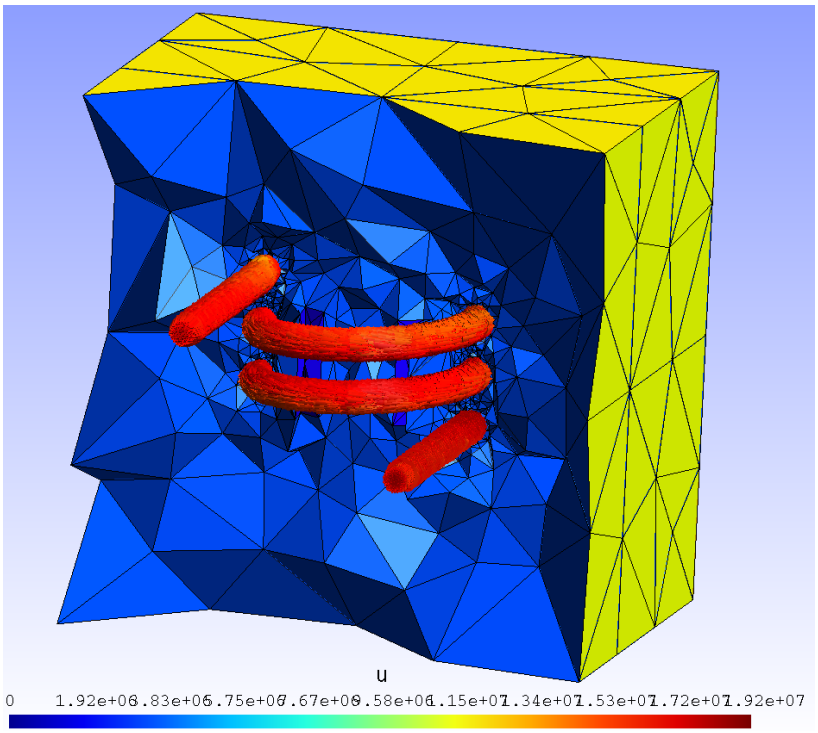

Figura 29. Visualización del post-procesado de la densidad de corriente que circula por la bobina.

5.2.3. Motor de inducción magnética

Gmsh también se puede emplear en simulaciones más complejas y en casos reales. En esta ocasión, este software se utiliza en la simulación de un modelo de motor de inducción magnética. A continuación, se muestra la geometría del mismo desde el eje z:

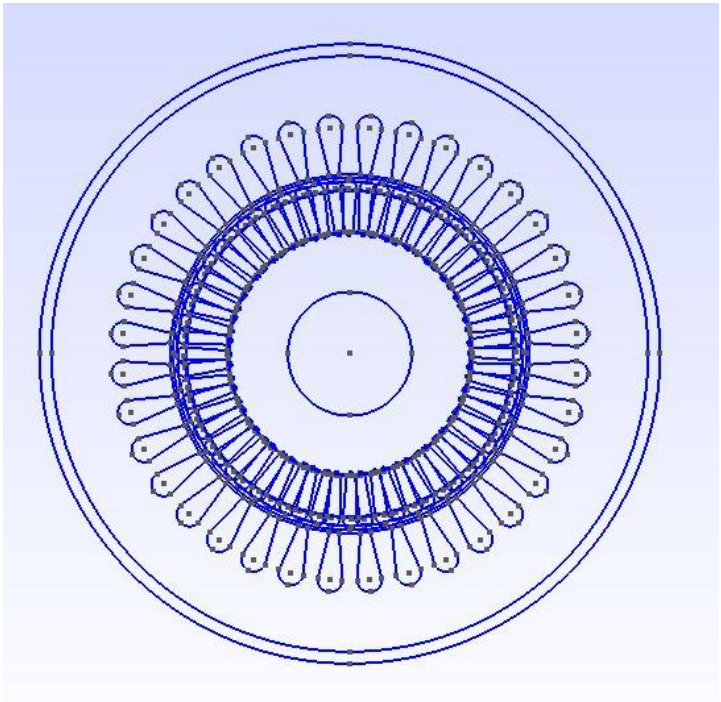

Figura 30. Geometría de motor de inducción magnética.

Seguidamente se podrá observar el mallado del motor de inducción magnética. De nuevo, se ha realizado un corte en la visualización para poder enseñar mejor el mallado del mismo.

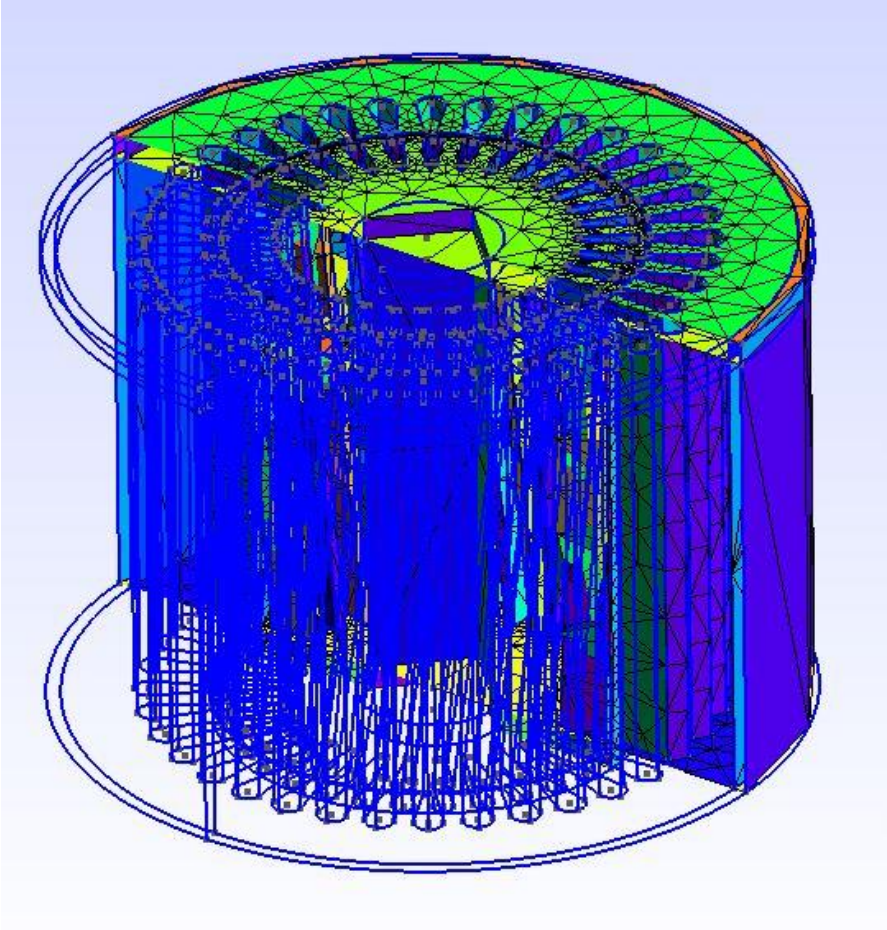

Figura 31. Mallado de motor de inducción magnética.

5.2.4. Microswitch

El último ejemplo de simulación en Gmsh es un microswitch capacitivo, mostrando así las posibilidades de este método de trabajo en aplicaciones novedosas, tanto del macro-mundo como del micro-mundo.

Un microswitch es un sistema MicroElectroMecánico (MEMs) basado en una viga apoyada sobre dos puntos, adaptado a escala de micras, que se flecta al aplicarle una carga. Este micro-dispositivo semiconductor cuenta con un buen aislamiento, una buena linealidad, un bajo consumo de potencia y un bajo coste, lo que le hace apropiado para aplicaciones en el ámbito de la ingeniería de telecomunicaciones. A continuación, se muestra la geometría y el mallado de este microswitch:

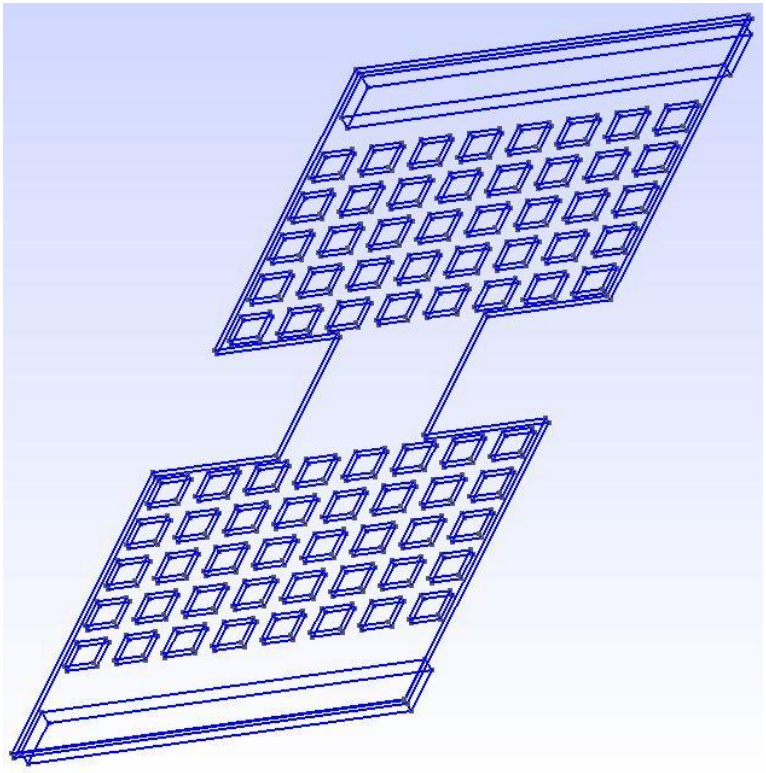

Figura 32. Geometría del microswitch.

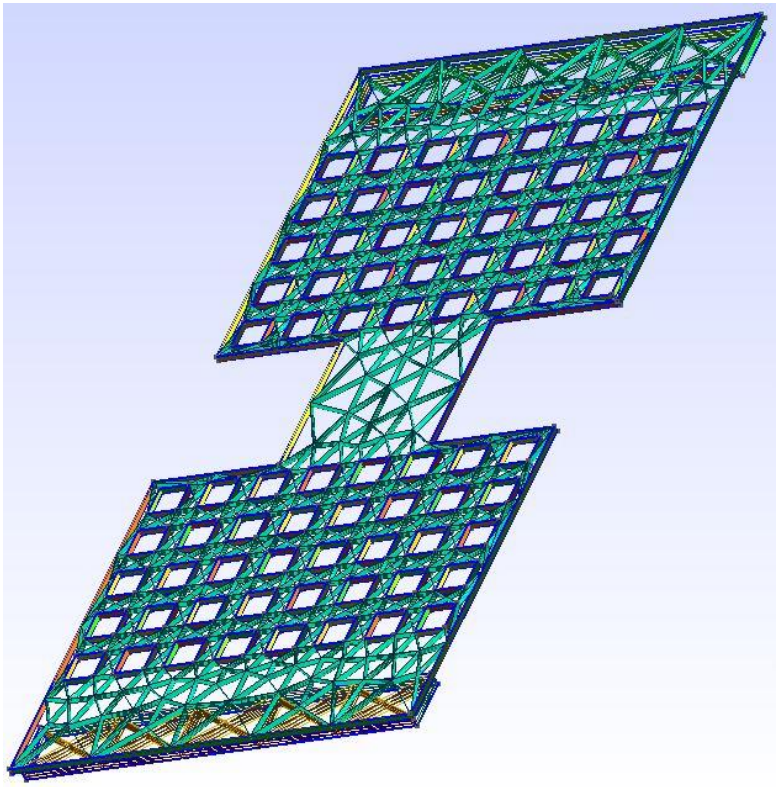

Figura 33. Mallado del microswitch.

Además, el ejemplo del microswitch sirve para mostrar como el módulo de post-procesamiento de Gmsh permite llevar a cabo análisis estacionarios y transitorios. Seguidamente se observa el análisis estacionario del desplazamiento absoluto del microswitch:

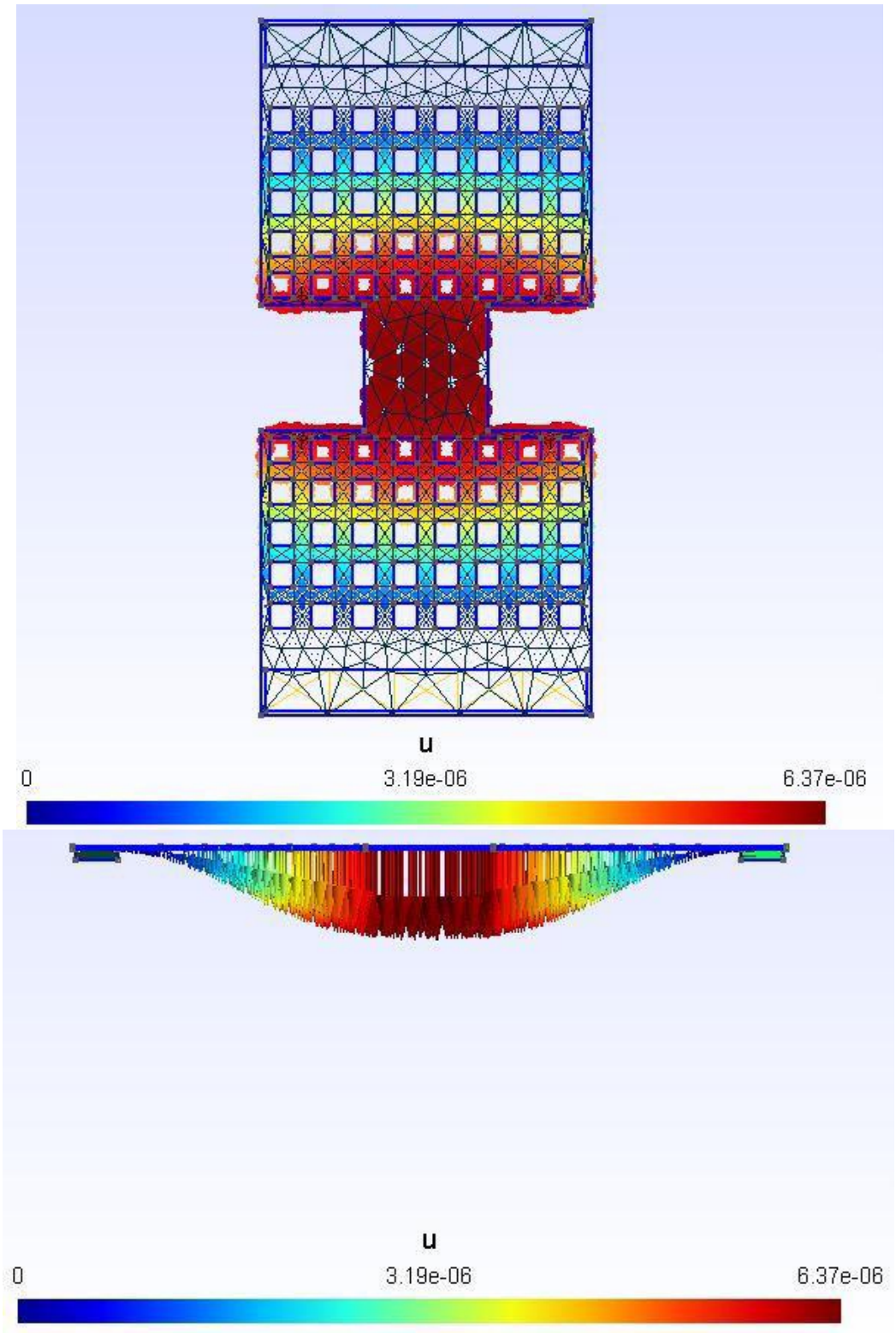

Figura 34. Análisis estacionario del desplazamiento absoluto del microswitch mostrado desde dos perspectivas distintas.

Finalmente, se muestra el análisis transitorio del desplazamiento por unidad de longitud:

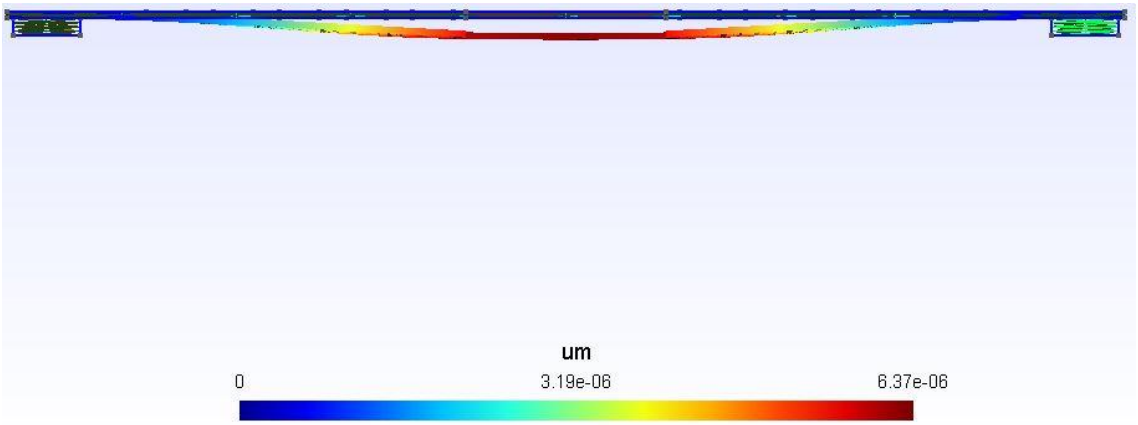

Figura 35. Análisis transitorio del desplazamiento por unidad de longitud del microswitch.

5.3. Conclusiones

El mallado de un dominio es imprescindible para poder estudiar un sistema contenido dentro de dicho dominio según el método de la celda, ya que este método numérico se basa en los elementos físicos como puntos, líneas, superficies y volúmenes que forman este mallado. Además, un software como Gmsh no solo permite diseñar la geometría del sistema y mallarlo, sino que posteriormente permite solucionar un problema físico mediante su solver, y llevar a cabo un post-procesado con el que visualizar los resultados, lo que resulta idóneo para esta aplicación.

Capítulo 6. Computación paralela

6.1. Introducción

La computación paralela es una forma de cómputo en la que varias instrucciones interrelacionadas se ejecutan simultáneamente. Normalmente esto se consigue empleando más de un procesador. El objetivo de la computación paralela es mejorar el rendimiento y la velocidad de ejecución.

Para paralelizar un algoritmo es necesario descomponer el problema principal en problemas más pequeños que serán asignados a procesadores distintos para ser resueltos simultáneamente. Esta descomposición podrá ser funcional o de dominio.

6.2. Tipos de paralelización

a) La descomposición funcional consiste en dividir el problema en subtareas que son asignadas a procesadores disponibles hasta que se repartan todas las tareas o hasta que no haya más procesadores disponibles. En el momento que un procesador termine con una tarea se le asigna otra tarea que esté en cola hasta que todas estén resueltas.

Sin embargo, la desventaja de la descomposición funcional es que las partes asignadas a los diferentes procesadores pueden requerir tiempos de ejecución significativamente diferentes unos de otros, sin que dichas diferencias puedan ser conocidas previamente. En ese caso, el tiempo de ejecución está determinado por el proceso que tarde más, mientras que los otros procesadores permanecen desocupados.

b) La descomposición de dominio consiste en dividir los datos en partes de similar tamaño y asignar estas partes a diferentes procesadores. En este caso, los procesadores pueden necesitar comunicarse para intercambiar datos, especialmente cuando finalicen las tareas y se quiera juntar los resultados asociados en un mismo archivo. Este tipo de computación paralela sigue un modelo SPMD (single program multiple data), es decir, una única tarea con distintos datos para cada procesador [31].

Otro aspecto fundamental en la programación de un algoritmo paralelo es el tipo de sistema de memoria, que pueden ser compartida o distribuida.

En un sistema de memoria compartida cada procesador tiene acceso a toda la memoria, por lo que hay un espacio de direccionamiento compartido. Todos los procesadores se encuentran comunicados con la memoria principal mediante un canal común. Para evitar que distintos procesadores tengan acceso simultáneo a regiones de memoria que provoquen algún error, los sistemas operativos deben utilizar algún mecanismo de control.

En un sistema de memoria distribuida, cada procesador cuenta con su propia memoria local. Para compartir información entre procesadores, es necesario el uso de mensajes que soliciten datos de la memoria de otro procesador. Por eso esta comunicación se le conoce como Paso de Mensajes.

6.3. Reducción del tiempo de ejecución

Uno de los principales objetivos de la paralelización es reducir el tiempo que tarda un programa serie en ejecutarse. De forma ideal, el tiempo de ejecución de un programa en paralelo ($t_{paralelo}$) sería la fracción del tiempo serie (t_{serie}) y el número de procesadores (p):

$$t_{paralelo} = t_{serie} / p$$

Sin embargo, en la realidad la reducción del tiempo al paralelizar un algoritmo está limitada por la Ley de Amdahl. Ésta dice que la aceleración está limitada por la parte de la tarea que no se puede paralelizar. En un algoritmo siempre existe una parte serie no paralelizable que limita la reducción del tiempo en paralelo. Por lo tanto, si un algoritmo tiene una parte serie no paralelizable que tarda t_s segundos, el tiempo de ejecución final en paralelo $t_{paralelo}$ nunca podrá ser menor que t_s [32].

Además, también hay que tener en cuenta los conceptos de tiempo ocioso (idle time) y el tiempo de comunicación.

El tiempo ocioso es el tiempo que un procesador gasta sin ejecutar ninguna tarea. Esto puede ocurrir cuando una tarea o subtarea no puede ser dividida entre los procesadores disponibles, como fue mencionado en la Ley de Amdahl. No obstante, un procesador también puede estar sin ejecutar ninguna tarea mientras espera datos de otros procesadores. Durante este tiempo el procesador no realiza trabajo útil.

El tiempo de comunicación es el tiempo empleado en enviar y recibir datos en forma de mensajes. Este problema es específico de los programas paralelos, ya que los programas secuenciales no utilizan comunicaciones entre procesos. Por ello, todo tiempo empleado en las comunicaciones es tiempo adicional que se añade al paralelizar un programa serie.

6.4. MPI

Como fue mencionado anteriormente, en el sistema de memoria distribuida cada procesador cuenta con su propia memoria. Para que los procesadores puedan comunicarse entre ellos se recurre al paso de mensajes, como ocurre en la interfaz MPI [26].

MPI (Message Passing Interface) es un interfaz estandarizado para la realización de aplicaciones paralelas basadas en paso de mensajes. Define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas con arquitectura paralela que utilicen múltiples procesadores.

MPI fue desarrollado por el MPI Forum, un grupo formado por investigadores de universidades, laboratorios y empresas involucrados en la computación de altas prestaciones. El interfaz nació en un congreso de estándares para el paso de mensajes en entornos de memoria distribuida celebrado en abril de 1992. En noviembre de este mismo año se publicó un primer borrador de MPI llamado MPI1 y en Junio de 1994 se presentó la primera versión de MPI.

El objetivo con MPI era crear un estándar universal para el paso de mensajes. Por ello, los investigadores incorporaron las características más útiles de diversos sistemas de IBM, Intel,

nCUBE, PVM, Express, P4 o PARMACS en vez de elegir uno solo y adoptarlo como estándar. De esta forma, se pretendía definir un entorno de programación único que garantizase la portabilidad de las aplicaciones paralelas y el interfaz de programación, favoreciendo la extensión del estándar al ofrecer una implementación de dominio público.

MPI se basa en especificaciones de lenguaje independientes (LIS por sus siglas en inglés) y una colección de bindings o librerías que permiten al usuario programar en diversos lenguajes a partir de C o Fortran.

MPI está diseñado para arquitecturas SPMD (single program, multiple data). Al arrancar una aplicación se lanzan en paralelo varios procesos o copias del mismo programa. Puesto que los procesos utilizan un espacio de memoria independiente unos de otros, el intercambio de información y la sincronización se lleva a cabo mediante paso de mensajes.

Un comunicador MPI conecta un grupo de procesos y les dota de una identificación para que estos puedan enviarse mensajes entre sí. El comunicador básico predefinido en MPI se denomina MPI_COMM_WORLD e incluye a todos los procesos activos durante la ejecución de una aplicación.

En MPI, a cada proceso se le asigna una variable que se denomina rank, con rango de 0 a p-1, donde p es el número total de procesos. Una parte importante del control de la ejecución del programa se realiza mediante esta variable, ya que determina qué proceso ejecuta una determinada porción de código.

Existen dos tipos de comunicaciones básicas entre procesos en MPI, dependiendo de la limitación en el número de procesos receptores: la comunicación punto a punto y la comunicación de funciones u operaciones colectivas.

La comunicación punto a punto se da únicamente entre dos procesos, como por ejemplo MPI_Send, que permite a un proceso en concreto mandar un mensaje a otro proceso.

La comunicación de funciones u operaciones colectivas se da entre todos los procesos pertenecientes un grupo de procesos. Este grupo podrá estar formado por todos los procesos de la aplicación o por un subconjunto de procesos previamente definidos. Un ejemplo de este tipo de comunicación broadcast es MPI_Bcast.

Además, existen dos modelos de comunicación dependiendo del tiempo que un proceso pasa bloqueado tras la llamada a una función de comunicación, el bloqueante y el no bloqueante.

Una función bloqueante mantiene a un proceso bloqueado hasta que la operación de comunicación solicitada se complete.

Una función no bloqueante inicia la operación de transferencia y recupera el control inmediatamente, sin asegurarse de que esta haya terminado. Para averiguar si la operación ha finalizado o no habrá que utilizar otra función como MPI_Test.

A continuación, se revisarán las funciones MPI más relevantes:

6.4.1. Funciones MPI

- #include "mpi.h",

Para poder ejecutar un algoritmo utilizando MPI, es necesario utilizar la instrucción *include* para llamar al fichero que contiene las definiciones, macros y prototipos de función necesarios para compilar los programas MPI.

- int MPI_Init(&argc,&argv)

Esta función permite inicializar una sesión MPI al realizar la configuración de la biblioteca MPI. Debe ser utilizada antes de cualquier otra función de MPI y solo requiere ser llamada una única vez. Sus argumentos son punteros a los parámetros de la función *main*(), *argc* y *argv*.

- int MPI_Comm_size(MPI_Comm comunicador, int* numprocs)

Esta función permite determinar el número total de procesos que pertenecen a un comunicador. Su primer argumento es el comunicador. En el segundo argumento retorna el número de procesos pertenecientes a dicho comunicador.

- int MPI_Comm_rank(MPI_Comm comunicador, int* identificador)

Esta función permite determinar el identificador (*rank*) del proceso actual. El primer argumento es el comunicador. En el segundo argumento retorna el identificador de un proceso.

- int MPI_Get_processor_name(char* nombre, int* longnombre)

Esta función permite conocer el nombre del procesador donde se está ejecutando cada proceso. Esta información puede ser útil para la monitorización de la ejecución de los programas.

- int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Esta función permite enviar información desde un proceso a otro de forma bloqueante. *Buf* es la dirección en memoria en donde se encuentra el dato, *datatype* es el tipo de dato, *count* es el número de datos que se encuentran en el buffer de envío, *dest* es el identificador del proceso destinatario del mensaje, *tag* es una etiqueta que se puede poner al mensaje, y *comm* es un comunicador.

- int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

Esta función permite recibir información desde otro proceso de forma bloqueante. A parte de *buf*, *datatype*, *tag* y *comm*, ya explicados en MPI_Send; MPI_Recv emplea *source*, que es el identificador del emisor del cual se espera un mensaje, y *status*, que es un resultado que se obtiene cada vez que se completa una recepción.

- int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)

Esta función es la análoga a MPI_Send en modo no bloqueante. Añade el objeto *request,* una solicitud de comunicación.

- int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

Esta función permite recibir información de forma no bloqueante.

- int MPI_Wait(MPI_Request *request, MPI_Status *status)

Esta función bloquea al proceso hasta que la operación correspondiente haya terminado. Utilizar esta función después de un envío de información no bloqueante equivale a utilizar un envío de información bloqueante.

- int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)

Esta función permite saber si la operación ha terminado o no mediante la actualización de un *flag*.

- int MPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Esta función permite que un proceso envíe un mensaje a todos los miembros del comunicador. *Root* es el indicador del proceso que envía el mensaje.

- int MPI_Cancel(MPI_Request *request)

Esta función permite cancelar una operación de comunicación que aún no se haya completado.

- int MPI_Finalize()

Esta función permite terminar una sesión MPI una vez el programa haya acabado de utilizar la biblioteca MPI. Debe ser llamada después del resto de funciones MPI utilizadas en el programa para liberar la memoria usada por MPI y limpiar así todos los trabajos no finalizados dejados por MPI.

El comunicador que se suele utilizar en todas estas funciones es "MPI_COMM_WORLD".

6.4.2. Tipos de datos

Los mensajes que se envían mediante MPI están formados por un cierto número de elementos de un mismo tipo conocidos como datatypes. Seguidamente se muestra una tabla con los datatypes usados en MPI más frecuentemente y su equivalencia en lenguaje C.

Tipos MPI	Tipos C equivalentes
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	Sin equivalente

Tabla 1. Tipos de datos MPI.

6.4.3. Ejemplos de programas básicos

Para mostrar el funcionamiento de MPI se explicarán dos programas básicos implementados con esta interfaz.

a) Hello world

"Hello world" es una adaptación del clásico programa didáctico para programación paralela. Es un programa muy sencillo en el que cada procesador muestra por pantalla "Hello world" además de indicar de qué procesador se trata. A continuación, se muestra el código que utiliza.

```
1  #include <stdio.h>
2  #include "mpi.h"
3  //using namespace std;
4
5  int main( argc, argv )
6  int  argc;
7  char **argv;
8  {
9      int rank, size;
10     MPI_Init( &argc, &argv );
11     MPI_Comm_size( MPI_COMM_WORLD, &size );
12     MPI_Comm_rank( MPI_COMM_WORLD, &rank );
13     printf( "Hello world from process %d of %d\n", rank, size );
14     MPI_Finalize();
15     return 0;
16 }
```

Figura 36. Código del programa "Hello world".

Como se puede observar, se emplean varias de las funciones MPI explicadas anteriormente. Primero se inicializa MPI con "MPI_Init"; se obtiene el valor de la variable "rank", es decir, el identificador del proceso, mediante la función "MPI_Comm_rank"; y también se obtiene el valor de la variable "size", es decir, el número de procesos ejecutado, mediante "MPI_Comm_size".

Después, cada proceso muestra por pantalla el mensaje "Hello world", donde se identifica el procesador que está mandando el mensaje. Antes de terminar, el programa finaliza MPI mediante "MPI_Finalize".

Seguidamente se mostrará el resultado de ejecutar el programa "Hello world" en un ordenador con siete procesadores.

```
apablo@lme0:~/paralelo1/intro_mpi> time mpirun -n 7 ./a.out
Hello world from process 4 of 7
Hello world from process 5 of 7
Hello world from process 6 of 7
Hello world from process 1 of 7
Hello world from process 2 of 7
Hello world from process 3 of 7
Hello world from process 0 of 7
```

Figura 37. Ejecución del programa "Hello world".

b) Integral

Este programa realiza el cálculo de la integral de coseno entre **π**/2 y 0. Puesto que se basa en MPI, el cálculo de la integral se puede llevar a cabo en más de un procesador.

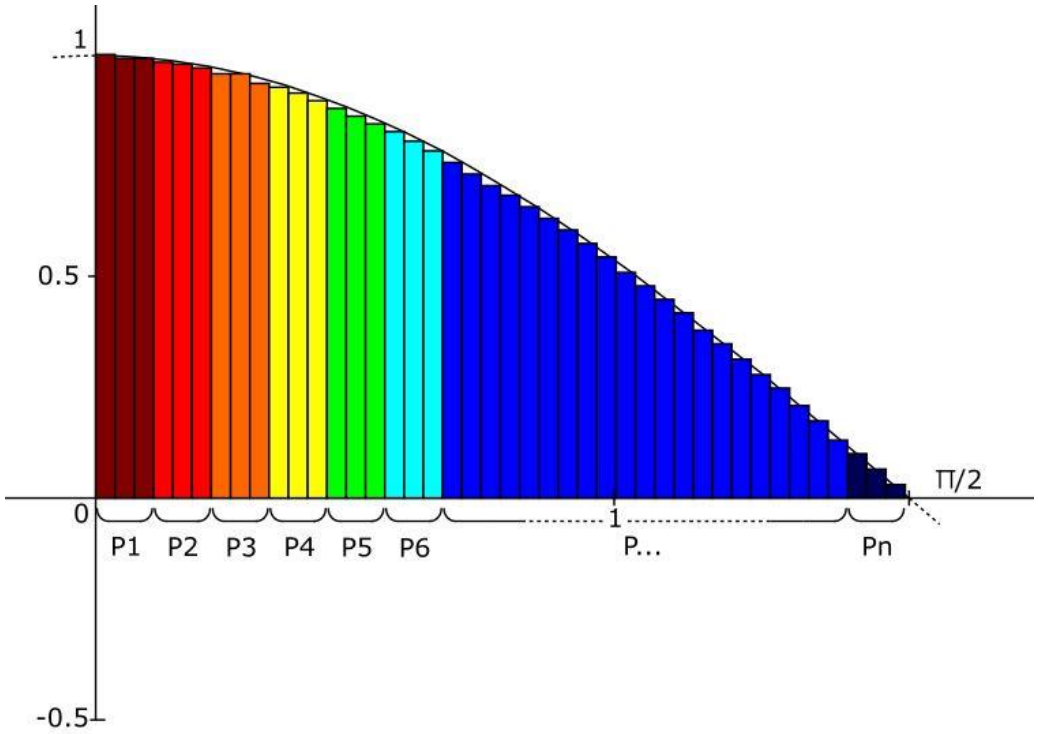

Figura 38. Integral de la función cos(x) en Pn procesadores.

En este programa se indica el número total de incrementos a integrar (n) y se divide entre el número de procesadores (p) para obtener el número de incrementos a calcular por cada procesador (num), como se puede observar en la siguiente imagen.

```
37        pi = acos(-1.0);  /* = 3.14159... */
38        a = 0.;           /* límite inferior de integración */
39        b = pi*1./2.;     /* límite superior de integración */
40        n = 20000000;          /* número de incrementos total */
41
42        dest = 0;         /* Define el proceso que calcula el resultado final*/
43        tag = 123;        /* Establece la etiqueta de identificación de esta tarea */
44
45    /* Inicializa MPI ... */
46
47        MPI_Init(&argc,&argv);              /* Inicia MPI */
48        MPI_Comm_rank(MPI_COMM_WORLD, &myid);  /* Obtiene el identificador del proceso */
49        MPI_Comm_size(MPI_COMM_WORLD, &p);     /* Obtiene el número de procesos */
50
51        h = (b-a)/n;    /* Longitud de cada incremento */
52        num = n/p;    /* Número de incrementos calculados por cada procesador*/
53        my_range = (b-a)/p;
54        my_a = a + myid*my_range;
55        my_result = integral(my_a,num,h);
56
57        printf("Process %d has the partial result of %f\n", myid,my_result);
```

Figura 39. Fragmento de código del programa Integral donde se reparten los incrementos.

En la figura 39 también se puede apreciar cómo se definen los límites de integración, se inicializa MPI, y se selecciona el proceso encargado de sumar los resultados parciales de todos los procesos (dest). Al final del fragmento de código se puede observar que se llama a la función "integral" y se guarda el resultado de cada procesador en la variable "my_result".

En la siguiente imagen se muestra la parte de código que junta los resultados de todos los procesos. En caso de ser el procesador encargado de dicha tarea (myid = 0), recibe los resultados parciales de la integral mediante la función "MPI_Recv", los suma en la variable "result", y lo muestra por pantalla. En caso de no ser el procesador encargado de sumar los resultados, simplemente envía su resultado parcial (my_result) al procesador correspondiente mediante la función "MPI_Send".

```
59        if(myid == 0) {
60          result = my_result;
61          for (i=1;i<p;i++) {
62            source = i;
63            MPI_Recv(&my_result, 1, MPI_REAL, source, tag,
64                     MPI_COMM_WORLD, &status);
65            result += my_result;
66          }
67          printf("The result =%f\n",result);
68        }
69        else
70          MPI_Send(&my_result, 1, MPI_REAL, dest, tag,
71                   MPI_COMM_WORLD);      /* Envía el resultado al proceso de destino*/
72        MPI_Finalize();                  /* Finaliza MPI ... */
73    }
```

Figura 40. Fragmento de código del programa Integral donde se suman los resultados parciales.

A continuación, se muestra la ejecución del programa en un ordenador con siete procesadores donde se calcula el tiempo de ejecución.

```
apablo@lme0:~/paralelo1/intro_mpi> time mpirun -n 7 ./a.out
Process 0 has the partial result of 0.218350
Process 1 has the partial result of 0.213646
Process 2 has the partial result of 0.186101
Process 3 has the partial result of 0.154026
Process 6 has the partial result of 0.025057
Process 5 has the partial result of 0.074378
Process 4 has the partial result of 0.119436
The result =0.990995

real    0m0.504s
user    0m1.353s
sys     0m0.163s
```

Figura 41. Ejecución del programa "Integral".

La siguiente tabla muestra la relación del número de procesadores que ejecutan este programa en paralelo y el tiempo que emplean en su ejecución. Aunque se trata de un programa sencillo, se observa como al trabajar con un mayor número de procesadores en paralelo se reduce el tiempo de ejecución respecto a utilizar un único procesador, equivalente a un programa serie.

Número de procesadores	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Tiempo (s)	1.129	0.751	0.690	0.688	0.546	0.527	0.504
Reducción temporal (s)	0	0.378	0.439	0.441	0.583	0.602	0.625
Mejora relativa (%)	0	33.48	38.884	39.061	51.639	53.322	55.359

Tabla 2. Relación del número de procesadores con el tiempo de ejecución del programa "Integral".

6.5. Conclusiones

La paralelización es una potente herramienta que permite mejorar el rendimiento y la velocidad de ejecución de los algoritmos mediante la utilización de varios procesadores de forma simultánea. Aunque existen diversas formas de paralelizar un programa, en esta aplicación se emplea la descomposición de dominio al dividir los datos de entrada en partes equivalente y asignarlas a distintos procesadores. Como se ha podido observar en el ejemplo del programa “Integral”, la paralelización de un programa realmente logra reducir los tiempos de ejecución, objetivo de este trabajo en los programas del preprocesamiento que se mostrarán más adelante.

Capítulo 7. PETSc

7.1. Introducción

PETSc es un entorno computacional desarrollado por el Laboratorio Nacional Argonne de Chicago, EEUU, cuyo nombre corresponde a las siglas "Portable, Extensible Toolkit for Scientific Computation". Es una herramienta que permite solucionar problemas físicos mediante ecuaciones diferenciales parciales en procesadores de altas prestaciones además de promover la flexibilidad y la reutilización de código y facilitar las tareas de paralelización de los algoritmos. Por ello, PETSc proporciona un entorno favorable para el modelado de aplicaciones científicas, diseño de algoritmos y prototipos, y cuenta con reconocimiento en el ámbito de las ciencias computacionales y la ingeniería.

La versión actual de PETSc, la versión 3.7, incluye *solvers* de ecuaciones paralelas lineales y no lineales, y ayudas para el empleo de matrices en paralelo, compatibles con códigos escritos en C, C ++, Fortran y Python. Emplea la interfaz de paso de mensajes (MPI) estándar para toda la comunicación de paso de mensajes, además de soportar rutinas de matrices paralelas distribuidas, útiles en los métodos de diferencias finitas [33].

7.2. Estructura

PETSc consiste en una serie de librerías en la que cada una interactúa con un tipo de objeto, como, por ejemplo, vectores, matrices, métodos de subespacios de Krylov, solvers de ecuaciones no lineales o controladores temporales para la resolución de ecuaciones en derivadas parciales [27]. La siguiente imagen muestra un diagrama de las interrelaciones entre los distintos elementos de PETSc y de su organización jerárquica.

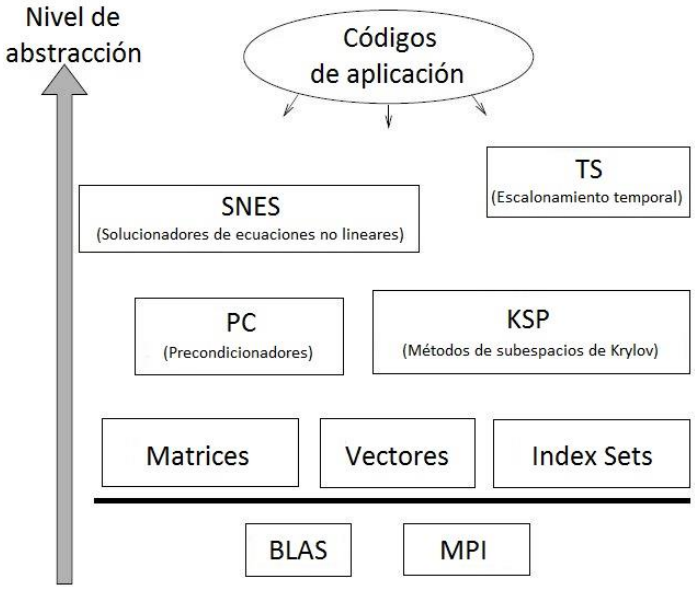

Figura 42. Organización de las librerías PETSc.

Como se puede observar en el esquema superior, los dos pilares básicos de PETSc son BLAS y MPI, este último ya explicado anteriormente.

BLAS, Basic Linear Algebra Subprograms, es una especificación que indica el conjunto de rutinas de bajo nivel para la ejecución de operaciones de algebra lineal, como la suma de vectores, la multiplicación de matrices o las combinaciones lineales. La funcionalidad de BLAS se puede dividir en tres categorías, conocidas como niveles, y que engloban diversas rutinas y algoritmos.

El nivel 1 consiste en las rutinas descritas en la primera presentación de BLAS en 1979, como operaciones vectoriales y productos escalares entre otros. Estas rutinas suelen tener una complejidad temporal lineal, es decir, que la relación entre el tiempo de ejecución y los datos de entrada es lineal.

El nivel 2 consiste en rutinas de operaciones entre matrices y vectores presentadas entre 1984 y 1986, como la multiplicación matriz-vector, por ejemplo. Estas rutinas suelen tener una complejidad temporal cuadrática, es decir, que la relación entre el tiempo de ejecución y el tamaño de los datos de entrada es cuadrática.

El nivel 3 fue presentado en 1990 e incluye operadores matriciales como la multiplicación matricial. En este caso, la complejidad temporal es cúbica, ya que la relación entre el tiempo de ejecución y el tamaño de los datos de entrada es cúbica.

Por encima del nivel de abstracción de BLAS y MPI en PETSc se encuentran los vectores, las matrices y los conjuntos de índices (Index Sets). Un index set es una generalización de un conjunto de índices de enteros que se utiliza para definir operaciones de vectores y matrices.

Los métodos de subespacios de Krylov (KSP) y los precondicionadores forman el siguiente nivel de la estructura de PETSc. Por un lado, el objeto KSP proporciona acceso a los paquetes de los solucionadores de sistemas lineales, ya sean paralelos o secuenciales, directos o iterativos. Por otro lado, los precondicionadores aplican una transformación para acelerar los ratios de convergencia de los procedimientos iterativos. De esta forma, la combinación de un método de subespacio de Krylov y un precondicionador forma el núcleo de un algoritmo para la solución iterativa de sistemas lineales.

Finalmente, en el último nivel de la arquitectura PETSc, justo por debajo de los algoritmos de aplicación, se encuentran los solucionadores de ecuaciones no lineales (SNES) y el escalonamiento temporal (TS). La librería SNES proporciona una estructura de rutinas para la solución de problemas no lineales de gran escala, permitiendo adecuar los solucionadores no lineales a la aplicación, ya sea de forma secuencial o paralela. La librería TS proporciona un entorno para la solución escalable de ecuaciones diferenciales ordinarias y algebraicas originarias de la discretización temporal de ecuaciones diferenciales parciales.

7.3. Funciones

El entorno PETSc cuenta con numerosas funciones propias, ya sea para trabajar con vectores (VecSetSize), matrices (MatSetValues), conjuntos de índices (ISGetIndices), objetos KSP

(KSPCreate), precondicionadores (PCSetType), solucionadores SNES (SNESSolve), escalonamiento temporal (TSSetFunction), o MPI. Las funciones PETSc de MPI se explicarán a continuación.

- PetscInitialize(int *argc,char ***argv,char *file,char *help)

Esta función inicializa una sesión MPI al llamar a la función MPI_Init(). El argumento "file" opcionalmente indica un nombre alternativo para el fichero de opciones de PETSc “. petscrc". El argumento "help" es una cadena de caracteres opcional que se imprimirá si el programa se ejecuta con la opción "-help".

Además, esta función establece el comunicador básico de PETSc llamado PETSC COMM WORLD.

- PetscFinalize()

Al terminar la sesión MPI se debe llamar a esta función que a su vez llamará a MPI_Finalize() para finalizar con las tareas de MPI y liberar así la memoria utilizada.

Para determinar el número total de procesos que pertenecen a un comunicador y el identificador o rank del proceso actual en PETSc se hace uso de las funciones MPI MPI_Comm_size y MPI_Comm_rank respectivamente. Sin embargo, el comunicador empleado será "PETSC_COMM_WORLD", como se podrá observar en los siguientes ejemplos.

7.4. Ejemplos de programas básicos

Para mostrar el funcionamiento de PETSc, se detallarán dos algoritmos básicos programados dentro del marco PETSc.

a) Hello world

"Hello world" es el mismo programa explicado en el capítulo de MPI, pero adaptado para funcionar en PETSc. A continuación, se mostrará el código utilizado.

```
3   #include <petscmat.h>
4   #undef __FUNCT__
5   #define __FUNCT__ "main"
6
7   int main(int argc, char **args)
8
9   {
10  PetscErrorCode  ierr;
11   int rank, size;
12   ierr = PetscInitialize(&argc,&args,(char*)0,help);CHKERRQ(ierr);
13     ierr = MPI_Comm_rank(PETSC_COMM_WORLD,&rank);CHKERRQ(ierr);
14    ierr = MPI_Comm_size(PETSC_COMM_WORLD,&size);CHKERRQ(ierr);
15
16      printf( "Hello world from process %d of %d\n", rank, size );
17
18    ierr = PetscFinalize();
19    return 0;
20  }
```

Figura 43. Código del programa "Hello world" en PETSc.

Como se puede apreciar en la figura 43, la estructura de este programa es exactamente igual al programa en MPI. Las diferencias entre ambos códigos son la forma de inicializar MPI, ya sea mediante PetscInitialize o MPI_Init; la forma de finalizar MPI, mediante PetscFinalize o MPI_Finalize; y el comunicador empleado en las funciones MPI_Comm_size y MPI_Comm_rank, ya sea PETSC_COMM_WORLD o MPI_COMM_WORLD.

b) Vector

"Vector" es un programa que inicializa un vector de longitud igual al número de procesos (variable "size") y lo rellena con los números naturales de 1 al valor de "size". Seguidamente se muestra el código del programa.

```
3   #include <petscmat.h>
4
5   #undef __FUNCT__
6   #define __FUNCT__ "main"
7   int main(int argc,char **argv)
8   {
9     Vec            X;
10    PetscScalar    *x;
11    PetscErrorCode ierr;
12    PetscInt       i;
13
14    PetscInitialize(&argc,&argv,0,help);
15    ierr = MPI_Comm_size(PETSC_COMM_WORLD,&size);CHKERRQ(ierr);
16
17    ierr = PetscMalloc1(size,&x);CHKERRQ(ierr);
18    for (i=0; i<size; i++) {
19      x[i] = i+1;
20    }
21    ierr = VecCreateSeqWithArray(MPI_COMM_SELF,1,size,x,&X);CHKERRQ(ierr);
22    ierr = VecAssemblyBegin(X);CHKERRQ(ierr);
23    ierr = VecAssemblyEnd(X);CHKERRQ(ierr);
24
25    ierr = VecView(X,0);CHKERRQ(ierr);
26
27    ierr = PetscFinalize();
28    return 0;
29  }
```

Figura 44. Código del programa "Vector".

Como se puede apreciar en la imagen anterior, después de crear y "montar" el vector con las funciones "VecCreateSeqWithArray", "VecAssemblyBegin" y "VecAssemblyEnd", se imprime por pantalla dicho vector mediante la función "VecView". Seguidamente se muestra la imagen de la ejecución del programa.

Figura 45. Ejecución del programa "Vector".

7.5. Conclusiones

PETSc es una herramienta que facilita la solución de problemas de aplicaciones numéricas que se basan en operaciones con matrices, como es el caso de las matrices de incidencia, por ejemplo; y que permite paralelizar los algoritmos serie para mejorar así los tiempos de ejecución y su rendimiento mediante la inclusión de la interfaz MPI. Por ello, PETSc es un entorno software excelente para este proyecto, ya que emplea MPI junto a otros elementos que favorece el empleo de matrices, elemento fundamental en los cálculos basados en el método de la celda.

Capítulo 8. Programas secuenciales del pre-procesado del Método de la Celda

8.1. Introducción

Los datos obtenidos en el software Gmsh deben pasar un preprocesamiento antes de poder ser procesados. En este caso, el modelo a simular será el de la bobina eléctrica con núcleo ferromagnético y aire circundante, como ya ha sido mencionado previamente en este trabajo.

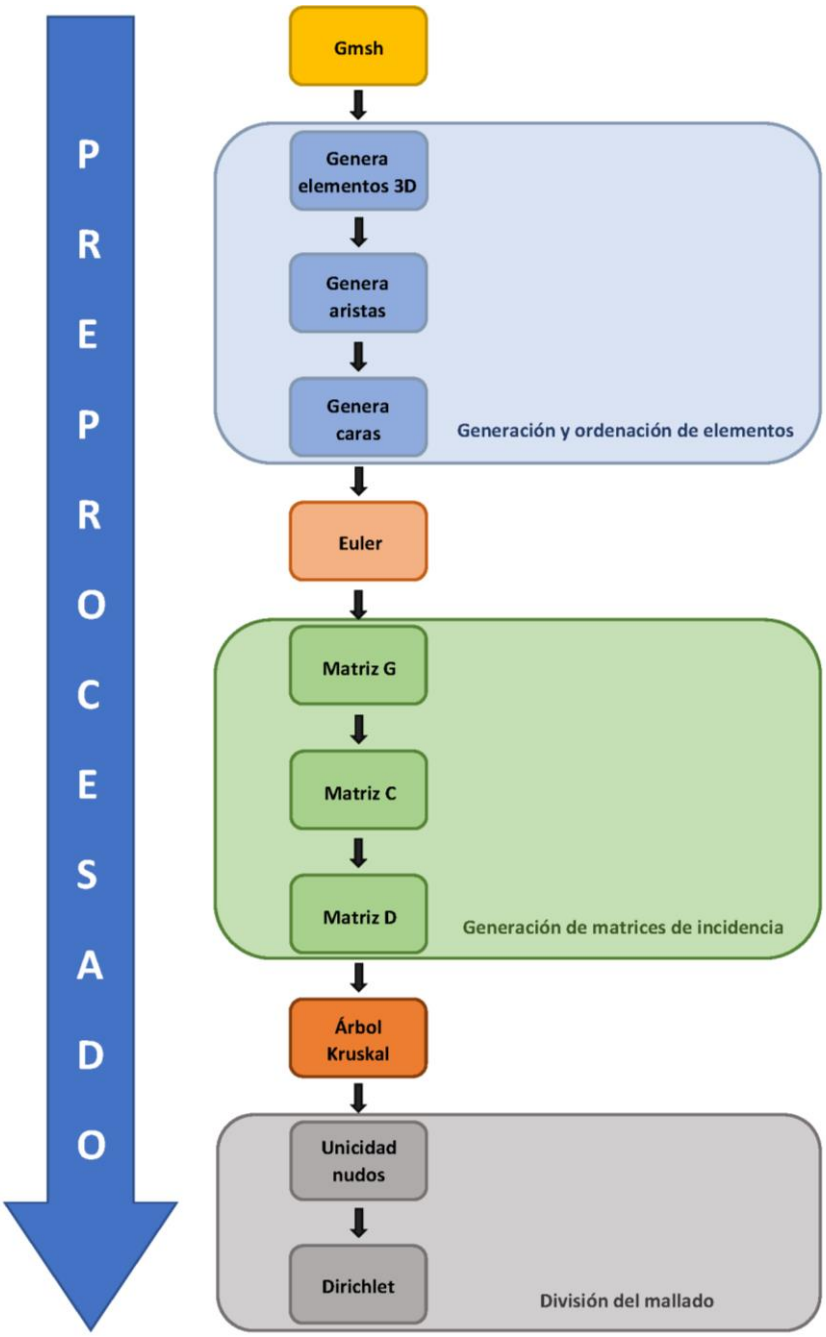

Figura 46. Esquema de los programas del pre-procesado

El preprocesamiento al que son sometidos estos datos se lleva a cabo en una serie de programas cuyo objetivo es la extracción y ordenación de datos, además de la obtención de elementos necesarios para los cálculos posteriores a partir de estos datos, como son las matrices de incidencias.

8.2. Programas secuenciales

A continuación, se explicarán los programas que forman parte del preprocesamiento de este proyecto.

8.2.1. Gmsh

El primer paso del preprocesamiento consiste en ejecutar el software Gmsh sobre un archivo de entrada *". geo"*, que contiene la geometría del sistema. Como salida se obtiene un fichero *".msh"* que contiene los datos de los elementos del mallado del sistema simulado. Por un lado, este fichero contiene las coordenadas de los puntos o nodos del modelo. Por otro lado, este fichero contiene información sobre el resto de elementos del sistema (líneas, superficies y volúmenes) en base a los puntos sobre los que se construyen, además de otros datos de interés respecto a estos elementos, como la etiqueta de dicho elemento, el tipo de elemento o la propiedad física a la que está asociado.

```
2889 -0.09381016107251675 0.03037475736122909 0.026072650381179
2890 -0.085148600496496 -0.015451486502966 0.0002944657091115185
2891 -0.03396453084521971 0.03182473411143337 0.02649093503711902
2892 -0.03911112683288862 0.03180431495689748 0.02442591911169356
2893 -0.04158033482039739 0.0289370750884401 0.02944996190466229
$EndNodes
$Elements
21116
1 1 2 2001 722 916 932
2 1 2 2001 722 1028 916
3 1 2 2001 722 1018 917
4 1 2 2001 722 917 1021
5 1 2 2001 722 918 934
6 1 2 2001 722 1038 918
7 1 2 2001 722 919 1032
8 1 2 2001 722 1034 919
9 1 2 2001 722 934 920
10 1 2 2001 722 920 1037
```

Figura 47. Detalle del fichero bobina.msh.

8.2.2. Genera elementos 3D

Este programa lee los datos del fichero ".msh" obtenido en el programa Gmsh y los organiza en distintos ficheros. En primer lugar, el algoritmo recorre todo el archivo ".msh" mientras lee los datos. Al reconocer la etiqueta que indica qué tipo de elemento se trata, el programa puede guardar los datos que acaba de leer sobre ese elemento en el fichero correspondiente. De esta forma, este programa crea cuatro ficheros de texto, uno para cada tipo de elemento: "nudos_msh.txt", "aristas_msh.txt", "triangulos_msh.txt" y "tetraedros_msh.txt". Estos ficheros contienen los datos de los elementos obtenidos, organizados de forma que en cada fila del archivo de texto tienen los puntos que forman el elemento y su propiedad física, en caso de las líneas, las superficies y los volúmenes; y la etiqueta del punto y sus tres coordenadas espaciales en caso de los nodos.

```
2612 0.0199178087089906 -0.0486048372198254 0.024541732695467
2613 -0.00999999999998247 -0.0357819624515177 -0.0351510344435522
2614 -0.0907860638904821 0.0282443214866757 0.0349127218585939
2615 0.0105317264585615 -0.0501290486832369 0.00454359922370495
2616 0.0458332260009525 -0.0212680655883899 -0.0116289059621863
2617 0.0105317264586054 0.0501290486831907 0.00545640077630795
2618 0.00853616790109704 0.0529354182095609 -0.0168242033420957
2619 -0.022591777914122 -0.0369351567520288 0.00812574038024068
2620 -0.0401350160766827 0.00160541549636564 0.005
2621 -0.0502187552761714 0.00060604086669009 0.0107672529740348
2622 -0.0298932534783091 -0.0165307381991738 0.00859971742427457
2623 0.0310757937611735 0.0178202402430756 -0.00275004417987483
2624 -0.0179493756009239 0.030960203734929 0.00186750884572787
2625 0.0176282977042003 0.031166036215747 -0.00208309655658398
2626 -0.0179186072658075 -0.0293534517626761 0.00897428369947519
2627 0.0310977110984677 -0.0169178833908579 -0.00656288634311689
2628 0.0169178833908588 0.0310977110984682 0.0184371136568831
2629 -0.0315100026040861 0.017478602450171 0.00285968397579612
2630 -0.0634699236146153 0.0233185871788986 0.00645856348160733
2631 0.021268065588389 0.0458332260009521 0.0133710940378135
```

Figura 48. Detalle del archivo nudos_msh.txt.

```
2923  1878 654 1876 2000
2924  1875 718 1863 2000
2925  718 1877 1863 2000
2926  745 1880 746 2000
2927  1878 1880 745 2000
2928  746 1860 747 2000
2929  1880 1860 746 2000
2930  748 747 1859 2000
2931  747 1860 1859 2000
2932  1859 749 748 2000
2933  1881 750 749 2000
2934  1855 1881 749 2000
2935  1855 749 1859 2000
2936  1854 751 750 2000
2937  750 1881 1854 2000
2938  751 932 916 2000
2939  932 751 1854 2000
2940  916 1857 752 2000
```

Figura 49. Detalle del archivo triangulos_msh.txt.

Adicionalmente, este programa crea un fichero más, "tetra3D_01.txt", que contiene información sobre los volúmenes del mallado. La diferencia entre los ficheros "tetra3D_01.txt" y "tetraedros_msh.txt" es que en "tetra3D_01.txt" todos los tetraedros tienen la misma orientación, externa en este caso. Esto se consigue calculando el determinante de una matriz formada por los nudos del volumen, y en caso de que sea negativo, permutando las dos primeras columnas de la matriz.

8.2.3. Genera aristas

Este programa lee los datos del archivo "tetra3D_01.txt", además del número de volúmenes del archivo de texto "datos_fisicos_msh.txt" con el objetivo de obtener los datos de las aristas del mallado respecto a los volúmenes ya orientados. El algoritmo recorre los cuatro nodos de cada tetraedro para guardar así en el archivo "aris3D_01.txt" los dos nodos de cada arista, de forma parecida a como se hizo en "aristas_msh.txt". Adicionalmente, el programa guarda el número de aristas en el archivo "numero.aristas".

8.2.4. Genera caras

Este programa es muy parecido al algoritmo anterior *genera aristas*. También lee los datos del archivo "tetra3D_01.txt" y el número de volúmenes del archivo "datos_fisicos_msh.txt". De esta forma, obtiene los datos de las caras o superficies del mallado respecto a los volúmenes ya orientados. El algoritmo recorre los cuatro nodos de cada tetraedro y guarda los tres nodos de cada superficie en el archivo "trian3D_01.txt". Al igual que el caso anterior, el programa guarda el número de caras en el archivo "numero.caras".

8.2.5. Euler

Este programa es el más sencillo de todos los algoritmos del pre-procesado. Simplemente lee el número de elementos de cada tipo y comprueba que la fórmula de Euler-Poincarè se cumpla, mostrando por pantalla el resultado de este programa. Puesto que el sistema de la bobina tiene un mallado sin agujeros, por defecto el algoritmo comprueba esta fórmula con la constante de Euler-Poincarè de valor 1 ($\chi = 1 - g = 1 - 0 = 1$).

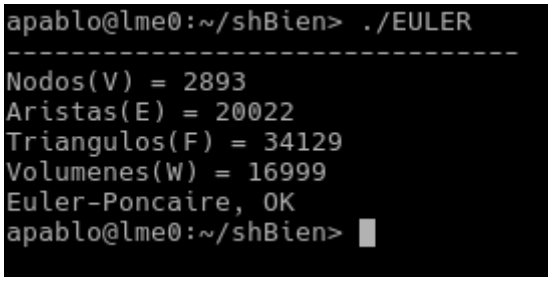

Figura 50. Captura de los resultados del programa Euler.

8.2.6. Matriz G

El objetivo de este programa es obtener la matriz de incidencias nudos-aristas o matriz G. Para ello, lee la información contenida en el fichero "aris3D_01.txt" y compara los nudos de las aristas con los otros nudos. En caso de que la arista y el nudo sean incidentes, en el archivo "matriz_G.txt" se guarda en una línea de este fichero el identificador del nudo, el identificador del primer nudo de la arista y el valor -1 como identificador de incidencia negativa; y en la siguiente línea del fichero se escribe el mismo identificador del nudo, el identificador del segundo nudo de la arista y el valor +1 como identificador de incidencia positiva.

```
1 939 -1
1 1085 1
2 1085 -1
2 104 1
3 104 -1
3 943 1
4 943 -1
4 939 1
5 939 -1
5 104 1
6 1085 -1
6 943 1
```

Figura 51. Detalle del archivo matriz_G.txt.

8.2.7. Matriz C

De forma similar al caso anterior, el objetivo de este programa es obtener la matriz de incidencias aristas-caras o matriz C. Se lee la información contenida en el fichero "trian3D_01.txt" y se comparan los nudos de las caras con los nudos de las aristas. En caso de que la cara y la arista sean incidentes, en el archivo "matriz_C.txt" se guarda el identificador de la cara, el identificador de la arista y el valor +1 en caso de que la incidencia sea positiva, o -1 en caso de que la incidencia sea negativa.

```
1 1 1
1 2 1
1 5 -1
2 2 1
2 3 1
2 6 -1
3 3 1
3 4 1
3 5 1
4 6 1
4 4 1
4 1 1
```

Figura 52. Detalle del archivo "matriz_C.txt".

8.2.8. Matriz D

Este programa construye la matriz de incidencias caras-volúmenes o matriz D. En este caso, el algoritmo lee la información de los volúmenes contenida en el fichero "tetra3D_01.txt" y lo compara con la información de las superficies del fichero "trian3D_01.txt". En caso de que la cara y el tetraedro sean incidentes, se escribe en el fichero "matriz_D.txt" el identificador del volumen, el identificador de la cara incidente, y el valor +1 en caso de que la incidencia sea positiva, o -1 en caso de que la incidencia sea negativa.

```
1 4 1
1 1 -1
1 3 -1
1 2 1
2 7 1
2 5 -1
2 4 -1
2 6 1
3 3 1
3 8 -1
3 10 -1
3 9 1
```

Figura 53. Detalle del archivo "matriz_D.txt".

8.2.9. Árbol Kruskal

Este programa es un algoritmo cuyo objetivo es encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor total de todas las aristas es igual al número de nudos menos la unidad. Para ello, el programa toma los datos del fichero "aris3D_01.txt", donde lee las aristas. El árbol con los identificadores de las aristas que lo forman se guarda en el fichero “vfila”.

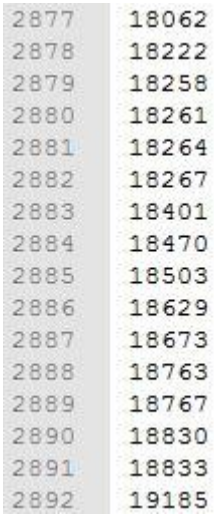

Figura 54. Detalle del archivo "vfila" con las aristas del árbol.

8.2.10. Unicidad nudos

Este programa divide el mallado original y se queda con una región del mismo, definiendo unos nuevos puntos y aristas en este nuevo mallado para facilitar así su futura utilización en otros programas. Estos nuevos puntos y mallados están referenciados con los del mallado original para guardar siempre coherencia entre los dos mallados. La división en el mallado se lleva a cabo para obtener un nuevo mallado con los elementos que compartan una propiedad física y así separar regiones físicas del espacio; o con los elementos de una cara o frontera del sistema para estudiar posteriormente el efecto del exterior del sistema en el mismo. Este programa utiliza el fichero ".msh", para obtener la información de los puntos y aristas del mallado.

8.2.11. Dirichlet

Este programa utiliza los mismos conceptos del programa "Unicidad nudos" para el caso particular de las condiciones de contorno Dirichlet, en las que es necesario conocer los valores de las soluciones que necesita la frontera del dominio para el procesado. Por ello, este programa prepara el mallado para aplicarle las condiciones Dirichlet a un subconjunto de puntos y aristas de la frontera del sistema previamente definido.

8.3. Resultados de programas secuenciales originales

Un aspecto estudiado en estos programas fue el tiempo de ejecución para distintos mallados de un mismo sistema, la bobina eléctrica. El tiempo de ejecución se midió utilizando el comando Unix "time", que indica el tiempo real, de usuario y de sistema. Sin embargo, en estos resultados solo se tendrá en cuenta el tiempo real, puesto que es el que interesa al comparar los tiempos de ejecución.

Estos programas se ejecutaron en varias ocasiones, para densidades de mallado distintas, desde 1,465 hasta 48,284 nodos en algunos casos.

Seguidamente se muestra una tabla con los resultados de las ejecuciones de los programas secuenciales para distinto número de nodos:

Nudos	1465	2893	5818	9778	19336	25534	36059
Gmsh	1.085s	1.645s	3.270s	5.284s	11.983s	19.611s	34.164s
Genera elementos	0.065s	0.078s	0.173s	0.241s	0.370s	0.414s	0.585s
Genera aristas	0.706s	2.477s	10.077s	28.053s	1m49.264s	3m9.399s	6m16.099s
Genera caras	1.633s	6.269s	26.861s	1m13.397s	4m49.386s	8m20.368s	16m30.693s
Euler	0.003s	0.001s	0.001s	0.003s	0.003s	0.001s	0.015s
Matriz G	0.032s	0.083s	0.127s	0.194s	0.410s	0.758s	0.884s
Matriz C	1.672s	5.873s	24.005s	1m4.864s	4m10.506s	7m46.153s	17m44.103s
Matriz D	3.787s	14.035s	59.779s	2m43.422s	11m8.631s	21m16.356s	42m5.707s
Árbol Kruskal	0.009s	0.016s	0.030s	0.017s	0.085s	0.045s	0.077s
Unicidad	0.916s	4.222s	14.314s	29.695s	1m46.741s	2m50.116s	4m37.761s
Diritlet	0.495s	0.921s	3.287s	8.654s	22.457s	27.979s	59.432s
Total	10.423s	35.642s	2m21.959s	6m13.870s	24m19.901s	44m11.250s	88m49.586s

Tabla 3. Tiempos de ejecución de programas secuenciales.

A continuación, se mostrarán unos gráficos para ilustrar estos resultados y facilitar su comprensión:

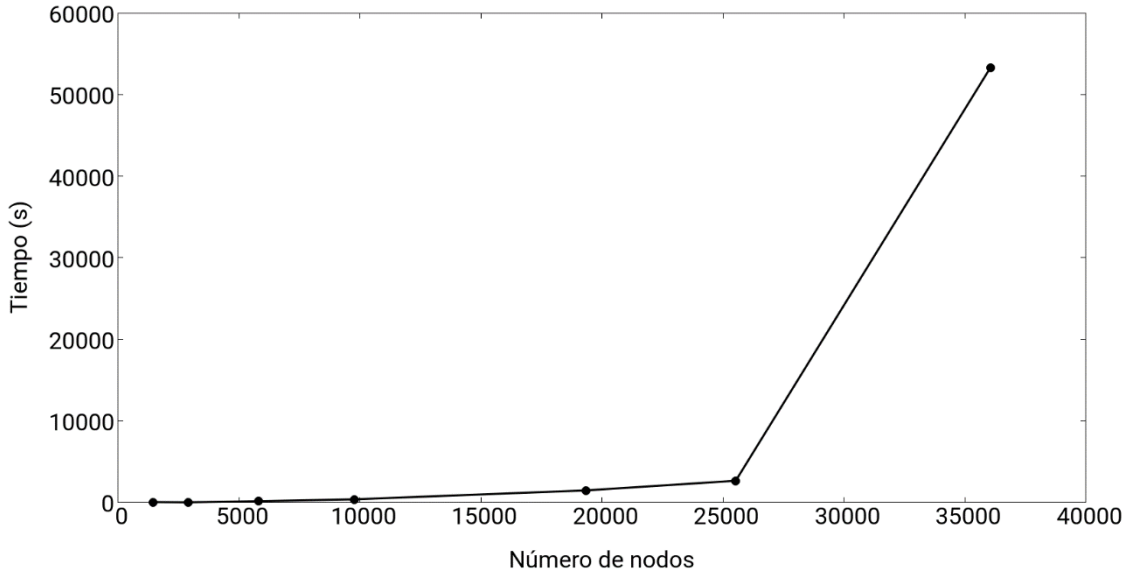

Figura 55. Gráfico del tiempo de ejecución total respecto al número de nodos.

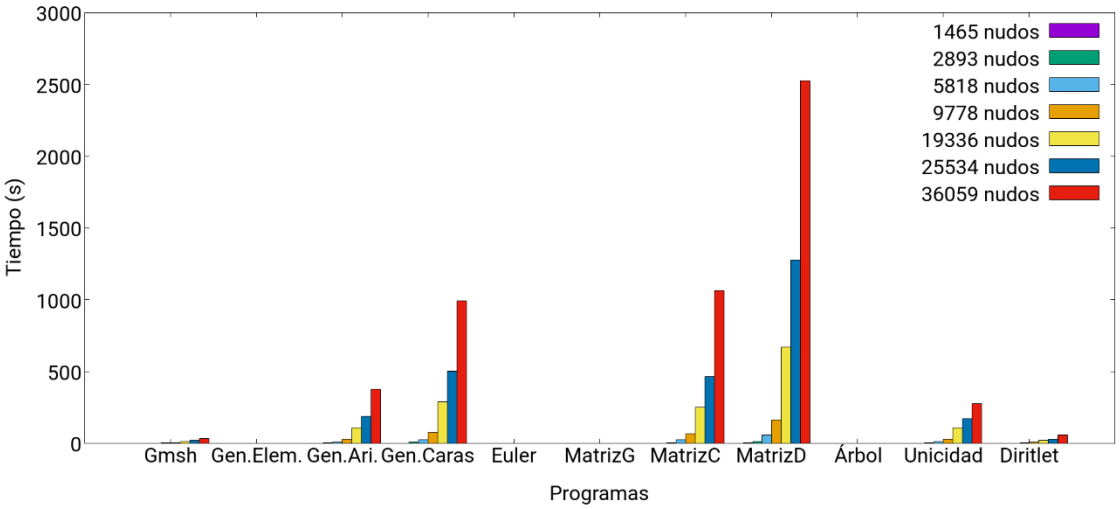

Figura 56. Gráfico de la comparación del tiempo de ejecución de cada programa respecto a los nudos del mallado.

Gracias a estos dos gráficos, se pueden obtener las siguientes conclusiones: el tiempo de ejecución total respecto al número de nudos sigue una relación no lineal, y el peso que tiene cada programa sobre esta relación no lineal no es uniforme.

Del primer gráfico se obtiene que es necesario reducir el tiempo de ejecución de los programas si se quiere mejorar el detalle del mallado y, con ello, de las simulaciones. Para ello, se optará por optimizar y/o paralelizar los programas secuenciales que así lo permitan.

Del segundo gráfico se obtiene la información de los programas que más tiempo consumen y, por ello, los que más necesitan dicha reducción de sus tiempos de ejecución. Por ejemplo, los programas Matriz D, Matriz C, Genera caras y Genera aristas son los que necesitan una mayor reducción temporal, como se puede contrastar con la tabla de tiempos de ejecución, donde se observa que alguno de ellos llega a los 40 minutos de tiempo de ejecución.

Por otro lado, los programas como Euler, Matriz G o Árbol Kruskal aparentemente no necesitan una gran reducción de sus tiempos, como se puede comprobar de nuevo con la tabla de tiempos de ejecución. En ella se puede observar como en programas como Euler, sus tiempos de ejecución no varían, o en otro como en la Matriz G, donde los tiempos aumentan, pero a menor ritmo que lo hace el número de nodos del mallado.

Aunque los resultados obtenidos utilizando el comando “time” son útiles para saber que es necesario reducir los tiempos de ejecución y en qué programa es especialmente necesario llevar a cabo una optimización o una paralelización del algoritmo, es conveniente conocer más información sobre qué líneas de código emplean más tiempo en ejecutarse. De esta forma, se puede focalizar los esfuerzos de mejora de los tiempos de ejecución en las secciones del algoritmo que más lo necesiten.

Para ello, en este caso se ha optado por utilizar el comando “gcov”, que forma parte de la colección de compiladores GNU [34]. Este comando analiza el rendimiento del algoritmo, no desde una perspectiva temporal, sino contando el número de veces que se ejecuta una línea del programa. De esta forma, se pueden localizar fácilmente las partes del algoritmo que deben ser optimizadas con mayor necesidad. Esto es de gran utilidad en el caso de tener bucles grandes, especialmente cuando existen bucles dentro de otros bucles.

A continuación, se muestra un ejemplo de los resultados obtenidos con la herramienta "gcov" para el programa "Matriz G", donde a la izquierda de las líneas de código se aprecia el número de veces que se ejecuta dicha línea. Como se puede observar, hay un bucle *while* dentro de un bucle *for*, por lo que la cantidad de veces que se ejecutan las líneas del bucle *while* es mucho mayor que la cantidad de veces que se ejecutan en el bucle *for*:

```
    -:   36: //Bucle de aristas. El contador debe empezar en i=1 pues las matriz G comienza
    -:   37: //por la fila 1, columna j=1.
136801:   38: for(i=1;i<na+1;i++)
    -:   39: {
    -:   40:  //Lee los nudos de las aristas
136800:   41:  p2>>ari0>>ari1;
    -:   42:
    -:   43:  //Inicializa el contador de nudos en el nudo de la última arista analizada
136800:   44: j=ari0;
    -:   45:
    -:   46:
410400:   47:  while (j<nn) //Bucle de nudos
    -:   48:  {
    -:   49:
    -:   50:   //Analiza el primer nudo de la arista i
136800:   51:   if (ari0==j)
    -:   52:   {
```

Figura 57. Detalle de los resultados obtenidos con la herramienta "gcov" para el programa Matriz C.

Seguidamente se muestra una tabla en la que se indica el mayor número de veces que una línea es ejecutada en cada programa, para el caso de un mallado de 2,893 nudos.

Nudos	**2893**
Genera elementos	340,508
Genera aristas	774,060,460
Genera caras	618,174,173
Euler	2
Matriz G	60,171
Matriz C	2,057,649,930
Matriz D	4,660,080,027
Árbol Kruskal	139,635
Unicidad	30,686,496
Diritlet	6,220,234

Tabla 4. Mayor número de veces que una línea es ejecutada en cada programa secuencial.

Aunque los datos recogidos en la tabla 4 son fiables, no deben tomarse como una relación exacta entre el tiempo de ejecución de un programa con el mayor número de veces que una línea del programa es ejecutada. Un programa puede tener una línea muchas veces ejecutada, pero tardar menos que otro programa con muchas más líneas de código ejecutadas un menor número de veces. Aun así, la herramienta "gcov" permite corroborar los datos obtenidos mediante el comando "time".

Sin embargo, como ya ha sido mencionado, la mayor utilidad de esta herramienta es localizar las líneas de código problemáticas y que necesitan ser optimizadas para reducir los tiempos de ejecución.

8.4. Optimización de programas secuenciales

Aunque el objetivo principal de este trabajo es la paralelización de los algoritmos del pre-procesado, también se han optimizado algunos de los programas secuenciales iniciales para reducir así su tiempo de ejecución. Una gran ventaja de la optimización de los programas es que es compatible con la paralelización de los mismos. De hecho, los dos programas optimizados fueron posteriormente paralelizados. De esta forma, los algoritmos son mejorados y depurados antes de llevar a cabo la paralelización de los mismos.

8.4.1. Genera aristas optimizado

La optimización del programa “Genera aristas” se obtiene al simplificar el código que obtiene los nodos de las aristas del fichero "tetra3D_01.txt" y de utilizar comandos Linux ya optimizados. En vez de hacer que un bucle ordene las aristas y elimine las repetidas, en la versión optimizada se emplea el comando “*sort”* para ordenar elementos con la opción “*uniq”* para eliminar los elementos repetidos.

8.4.2. Genera caras optimizados

Al igual que ocurre en el caso anterior, la optimización del programa “Genera cara” se lleva a cabo simplificando la parte del algoritmo que obtienen los nodos de las caras del fichero "tetra3D_01.txt" y utilizando comandos Linux previamente optimizados. De nuevo, se utiliza el comando “*sort”* con la opción “*uniq”* para ordenar los elementos y eliminar los repetidos.

8.5. Resultados de programas optimizados

La optimización de los programas “Genera aristas” y “Genera caras” ha conseguido reducir los tiempos de ejecución de dichos programas. Aunque aparentemente los cambios en la optimización de los algoritmos no son drásticos, sí que suponen una reducción temporal muy importante, como se puede apreciar en la siguiente tabla:

Nudos	Genera aristas		Genera caras		Diferencia temporal total	
	Original	Optimizado	Original	Optimizado	Absoluta	Relativa
1465	0.706s	0.298s	1.633s	0.177s	1.864s	79.692%
2893	2.477s	0.435s	6.269s	0.304s	8.007s	91.55%
5818	10.077s	0.822s	26.861s	0.654s	35.462s	96.004%
9778	28.053s	1.384s	1m13.397s	1.070s	1m38.996s	97.581%
19336	1m49.264s	2.782s	4m49.386s	2.208s	6m33.66s	98.748%
25534	3m9.399s	3.784s	8m20.368s	2.844s	11m28.139s	99.039%
36059	6m16.099s	5.339s	16m30.693s	4.087s	22m38.366s	99.310%
42250	8m50.995s	6.385s	23m21.421s	5.368s	32m0.663s	99.392%
48284	11m36.741s	7.210s	30m43.902s	5.629s	42m7.804s	99.495%

Tabla 5. Comparación de los tiempos de ejecución de los programas originales y optimizados.

En la tabla 5 se pueden comparar los tiempos de ejecución de los programas “Genera aristas” y “Genera cara” originales y los optimizados. Además, en las dos últimas columnas, se puede observar la diferencia total entre los dos programas originales y los dos optimizados, expresada en valor absoluto y relativo. Estos datos se pueden representar de forma gráfica, lo que facilita así su comprensión, como se muestran seguidamente:

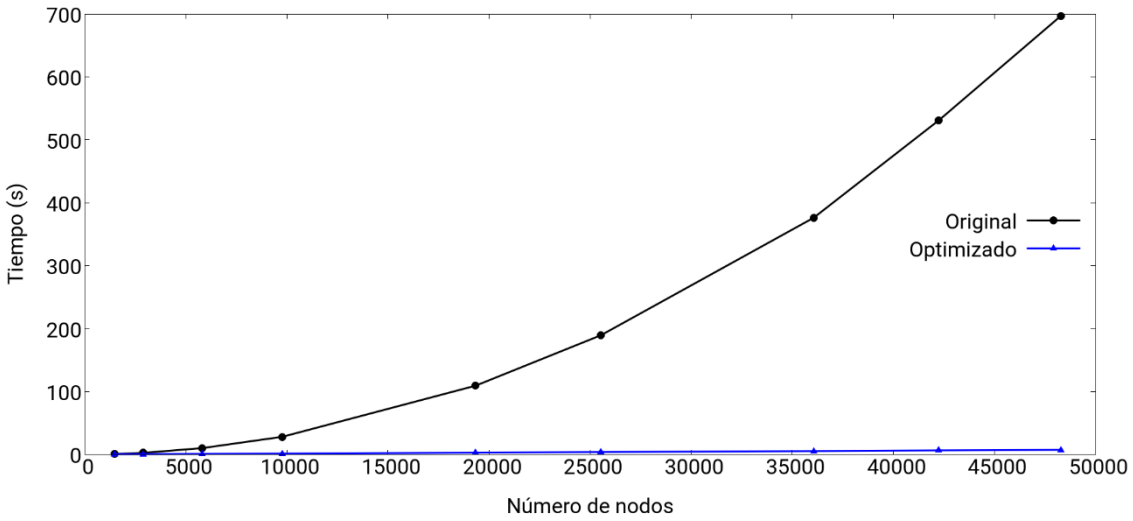

Figura 58. Gráfico comparativo entre el tiempo de ejecución del programa “Genera aristas” original y optimizado.

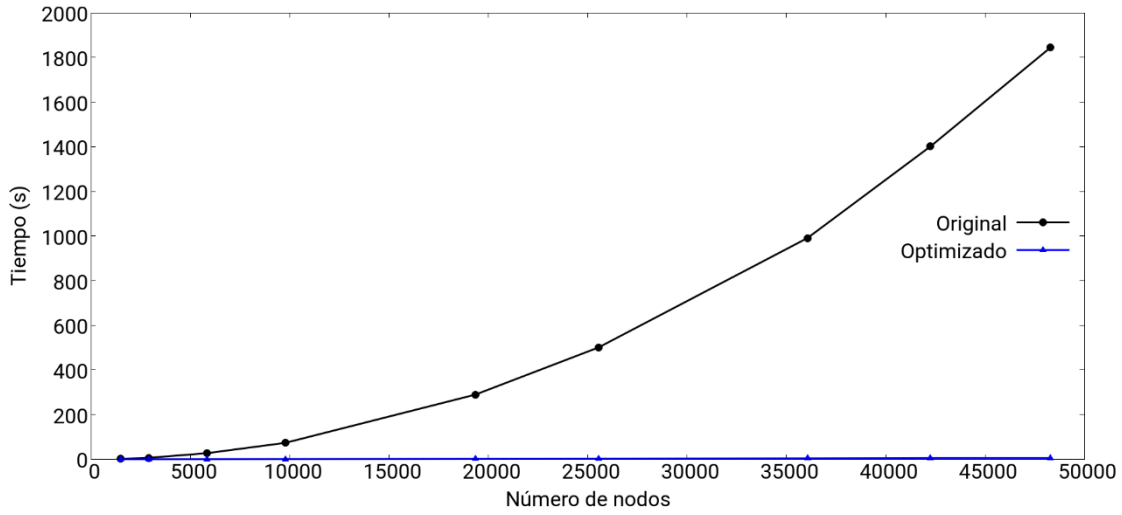

Figura 59. Gráfico comparativo entre el tiempo de ejecución del programa “Genera caras” original y optimizado.

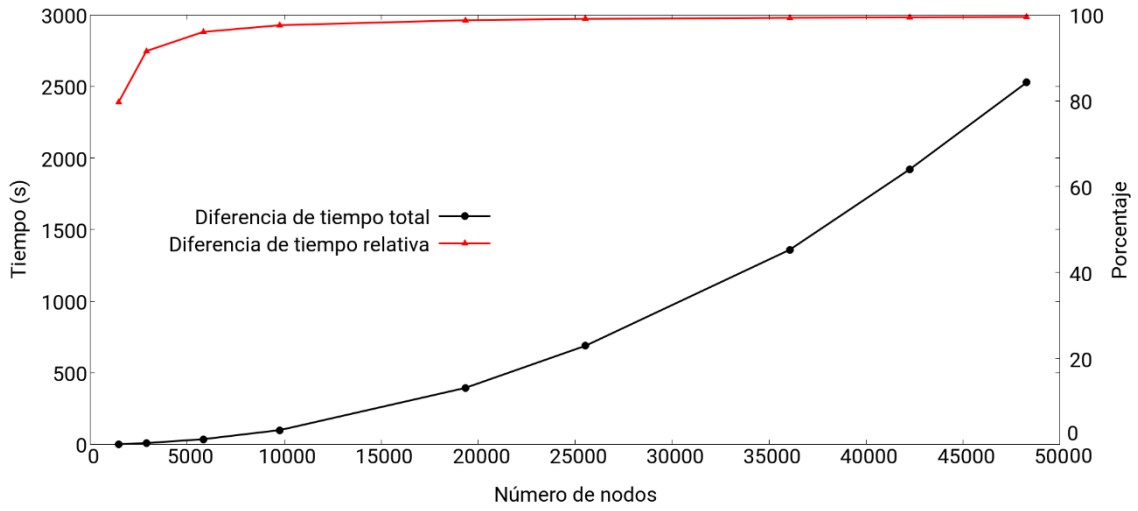

Figura 60. Gráfico comparativo de la diferencia en los tiempos de ejecución de los programas originales y los optimizados.

En los dos primeros gráficos se puede ver la diferencia temporal entre las ejecuciones de los programas originales y los optimizados al aumentar el número de nudos del mallado. Como se puede apreciar, la reducción temporal de los programas optimizados es muy llamativa, ya que éstos prácticamente no aumentan los tiempos de ejecución en comparación con los programas originales.

En el tercer gráfico se compara la diferencia total entre los tipos de programas, sumando las diferencias temporales entre el programa "Genera aristas" original y el optimizado, y las diferencias entre el programa "Genera caras" original y el optimizado. Al observar la diferencia relativa, se aprecia que la mejora producida por la optimización de los algoritmos es muy significativa, al alcanzarse mejoras desde un 79% hasta un 99%, como se puede comprobar en la tabla 5.

La herramienta "gcov" también se utilizó en los programas optimizados para corroborar así que dicha optimización había afectado al número de veces que el programa ejecutaba cada línea. A continuación, se muestra una tabla en la que se indica el mayor número de veces que una línea es ejecutada en cada programa, para el caso de un mallado de 2,893 y otro de 19,336 nudos.

Nudos	**Genera aristas**		**Genera caras**	
	Original	Optimizado	Original	Optimizado
2,893	774,060,460	34,033	618,174,173	17,033
19,336	17,746,420,555	116,993	28,538,371,551	58,453

Tabla 6. Comparación del mayor número de veces que una línea es ejecutada entre programas originales y optimizados.

Al igual que se comentó anteriormente, el mayor número de veces que una línea de programa es ejecutada no implica necesariamente una reducción o un aumento del tiempo de ejecución de estos programas. Sin embargo, estos datos sí que sirven como indicativo de que la optimización de los programas originales han sido un éxito, ya que el mayor número de veces que una línea de programa es ejecutada se ha reducido considerablemente. De esta forma se corrobora la reducción temporal previamente obtenida mediante la herramienta "time".

8.6. Conclusiones

En este capítulo se ha explicado el funcionamiento de los programas secuenciales, se ha mostrado como el tiempo de ejecución del conjunto de programas aumenta a mayor ritmo que el número de nodos del mallado, y cómo afecta cada programa individualmente a ese incremento temporal. De esta forma, se ha podido establecer un criterio de los programas que más necesitan una reducción temporal. Además, gracias a la herramienta “gcov”, se ha podido averiguar las líneas de código de los algoritmos que más se ejecutan, lo que ha facilitado su optimización o paralelización.

Adicionalmente, se han optimizado los programas “Genera aristas” y “Genera caras” y se han explicado los resultados obtenidos tan satisfactorios, al alcanzar reducciones temporales mayores a un 75% en todos los casos.

Capítulo 9. Paralelización de programas secuenciales del pre-procesado del Método de la Celda

9.1. Introducción

La paralelización de los programas del pre-procesado se ha llevado a cabo siguiendo una filosofía de descomposición de dominio, es decir, dividiendo los datos en partes de tamaño parecido para cada procesador.

Los programas sujetos a una paralelización son aquellos con mayores tiempos de ejecución, como “Matriz D” y “Matriz C”; además de programas que, aunque no tengan tiempos de ejecución tan grandes, se emplean más frecuentemente, como “Genera aristas”, “Genera caras” o “Matriz G”.

Con el objetivo de estudiar los resultados de los tiempos de ejecución, estos programas se han ejecutado en un Intel Core i7 – 3820, de 3.6 GHz de frecuencia, 32 GB de memoria, con 4 núcleos y 8 hilos de ejecución (dos por núcleo). Además, este equipo cuenta con una tarjeta gráfica GPU NVIDIA GM107 (GeForce GTX 750 Ti) de 2 GB.

Este equipo, identificado con el nombre de LME 0, forma parte de un clúster de 14 ordenadores conectados entre sí mediante una red local. Este clúster podrá ser de utilidad en futuras líneas de trabajo para utilizar los procesadores de equipos que forman parte del clúster en las simulaciones paralelas de forma simultánea.

Figura 61. Equipos del clúster. De derecha a izquierda, LME 0, LME 6, LME 7 y LME 8.

9.2. Procedimiento

La paralelización de los programas se ha llevado a cabo, de forma general, dividiendo el bucle *for* más externo de cada programa en partes iguales para cada procesador. En lugar de avanzar desde 0 hasta el valor *N*, cada procesador tiene un bucle *for* que avanza desde un valor inicial (*I*) y un valor final (*F*). Las variables *I* y *F* se determinan, para cada procesador, mediante la siguiente subrutina:

```
int RPN(int rank, int size, int N,int *I,int *F)
{
//Esta función divide intervalos segun rank size y N y dev. I, F


if (rank !=0) {*I=rank*int(floor(N/size))+1;}
else {*I=0;}

if (rank !=(size-1)){*F=(rank+1)*int(floor(N/size));}
else {*F=N-1 ;}

return 1;

} //FIN
```

Figura 62. Subrutina que determina el rango de un bucle para cada procesador.

Sabiendo que *rank* es el identificador del procesador, *size* el número de procesadores y *N* la cantidad de datos, se puede observar como esta rutina determina el valor de las variables *I* y *F* dividiendo la cantidad de datos *N* entre el número de procesadores *size*, y multiplicando el valor obtenido por el identificador del proceso *rank* o *rank* + 1 en caso de querer obtener el valor final *F*.

```
81
82  N=volumenes;
83  RPN(rank, size,N ,&I,&F);
84
85  I=I+1;
86  F=F+1;
87
88  for(contador2=0;contador2<I;++contador2){
89  PARAMETER_read>>nodoA>>nodoB>>nodoC>>nodoD>>atrib;
90  }
91
92   for(contador2=I;contador2<=F;++contador2){
93     PARAMETER_read>>nodoA>>nodoB>>nodoC>>nodoD>>atrib;
94
95
96  A=nodoA;B=nodoB;C=nodoC;D=nodoD;
```

Figura 63. Ejemplo de paralelización por descomposición de dominio.

En la imagen anterior se observa el uso de la función RPN para la paralelizar un algoritmo por descomposición de dominio. En primer lugar, se inicializa la variable *N* como el número de volúmenes a leer y se llama a la función. Después se ajustan los valores de las variables *I* y *F* añadiéndoles una unidad. Seguidamente, se utiliza un bucle *for* para lograr que todos los procesadores comiencen la lectura del fichero del siguiente bucle *for* en la posición indicada por

la variable de inicio *I*. Finalmente, con el segundo bucle *for*, desde la variable de inicio *I* hasta la variable de finalización *F*, se lleva a cabo la paralelización por descomposición de dominio al dividir los datos de entrada entre todos los procesadores.

9.2.1. Genera aristas

El programa “Genera aristas” paralelo comienza inicializando el entorno PETSc y MPI, como se puede ver a continuación:

```
32
33  PetscErrorCode  ierr;
34  int rank, size;
35  ierr = PetscInitialize(&argc,&args,(char*)0,help);CHKERRQ(ierr);
36  ierr = MPI_Comm_rank(PETSC_COMM_WORLD,&rank);CHKERRQ(ierr);
37  ierr = MPI_Comm_size(PETSC_COMM_WORLD,&size);CHKERRQ(ierr);
38
```

Figura 64. Inicialización de PETSc y MPI.

Este programa se ha paralelizado según se explicó en el ejemplo anterior. Se utilizó la función RPN para establecer el valor de las variables de inicio y fin del bucle para cada procesador, se ajustó la posición de lectura del fichero de datos de entrada en la posición de inicio y se siguió el procedimiento normal del algoritmo “Genera aristas” con los datos de entrada correspondientes para cada procesador. Los datos de salida se guardan en el fichero "aris3D_01_1.txt", como en el caso de los programas secuenciales. Sin embargo, en el programa paralelo se utiliza la opción “ios :: app” para que cada procesador escriba en una línea nueva al final del archivo. De esta forma, se evita que los procesadores se solapen y escriban datos en la misma línea, y se proporciona un método de escritura para varios procesadores, en un mismo archivo, de forma simultánea.

9.2.2. Genera caras

El programa Genera caras se paraleliza de la misma forma que el programa Genera aristas, al contar con estructuras análogas. En primer lugar, se inicializa PETSc y MPI, después se ejecuta la función RPN para obtener el valor de las variables de inicio y fin, se ajusta el fichero de entrada de datos en la posición de la variable de inicio, y finalmente, el programa “Genera caras” obtiene los datos de las caras a partir del fichero "tetra3D_01.txt", al igual que sucede en la versión secuencial, pero en este caso mediante una paralelización por descomposición de dominio.

9.2.3. Matriz G

El programa Matriz G paralelizado es similar a la versión secuencial con la inicialización de PETSc y MPI al comienzo del mismo y con el bucle de comparación de nudos y aristas paralelizado gracias a las variables iniciales y finales obtenidas en la función RPN. Los datos obtenidos por los diversos procesadores se guardan simultáneamente en el fichero "matriz_G.txt", al igual que en la versión secuencial.

9.2.4. Matriz C

El programa de Matriz C paralelizado comienza inicializando PETSc y MPI, como es habitual. Sin embargo, en este caso se paraleliza la función del programa “IsMember” ya que es la parte del código que más veces es ejecutada. Por ello, cada vez que se llama a dicha función, se paraleliza el bucle *for* más externo de la misma. De nuevo, el programa paralelo utiliza la

opción “ios :: app” para que la información obtenida en cada procesador no se mezcle y se escriba correctamente en el fichero "matriz_C.txt".

9.2.5. Matriz D

El programa Matriz D tiene una estructura similar al programa Matriz C. Por eso, la forma de paralelizarlo es semejante: se inicializa PETSc y MPI y se paraleliza la función “IsMember” al dividir su bucle *for* entre los distintos procesadores. También se vuelve a utilizar la opción “ios :: app” para guardar la información en el fichero "matriz_D.txt" de forma correcta.

9.3. Resultados

La paralelización de los programas secuenciales ha dado como resultado reducciones considerables en los tiempos de ejecución. A continuación, se muestran las tablas donde se observa esta reducción de los tiempos de ejecución de los programas paralelizados frente a los programas serie, tanto en valores absolutos como en valores relativos:

Nudos		1465	2893	5818	9778	19336	25534	36059
Genera aristas	1 procesador	0.298s	0.435s	0.822s	1.384s	2.782s	3.784s	5.339s
	2 procesadores	0.216s	0.294s	0.537s	0.85s	1.529s	1.913s	2.706s
	3 procesadores	0.16s	0.264s	0.468s	0.694s	1.421s	1.882s	2.569s
Genera caras	1 procesador	0.177s	0.304s	0.654s	1.070s	2.208s	2.844s	4.087s
	2 procesadores	0.173s	0.232s	0.428s	0.602s	1.161s	1.420s	2.555s
	3 procesadores	0.129s	0.172s	0.33s	0.514s	1.005s	1.308s	1.948s
Matriz G	1 procesador	0.032s	0.083s	0.127s	0.194s	0.410s	0.758s	0.884s
	2 procesadores	0.083s	0.121s	0.308s	0.835s	2.072s	3.413s	6.432s
	3 procesadores	0.078s	0.119s	0.295s	0.629s	1.612s	2.583s	5.069s
Matriz C	1 procesador	1.672s	5.873s	24.005s	1m4.864s	4m10.506s	7m46.153s	17m44.103s
	2 procesadores	1.037s	2.947s	13.350s	33.502s	2m11.492s	4m18.422s	9m16.934s
	3 procesadores	0.728s	2.074s	9.192s	23.990s	1m27.183s	2m32.291s	6m24.963s
Matriz D	1 procesador	3.787s	14.035s	59.779s	2m43.422s	11m8.631s	21m16.356s	42m5.707s
	2 procesadores	2.228s	7.297s	30.427s	1m21.910s	5m52.166s	10m47.917s	22m2.976s
	3 procesadores	1.439s	4.946s	20.953s	55.987s	3m55.687s	7m20.770s	15m11.673s

Tabla 7. Resultados absolutos de tiempos de ejecución de los programas paralelos.

Nudos		1465	2893	5818	9778	19336	25534	36059
Genera aristas	1 procesador	0.298s	0.435s	0.822s	1.384s	2.782s	3.784s	5.339s
	2 procesadores	27,517%	32,414%	34,672%	38,584%	45,040%	49,445%	49,316%
	3 procesadores	46,309%	39,310%	43,066%	49,855%	48,922%	50,264%	51,882%
Genera caras	1 procesador	0.177s	0.304s	0.654s	1.070s	2.208s	2.844s	4.087s
	2 procesadores	2,260%	23,684%	34,557%	43,738%	47,418%	50,070%	37,485%
	3 procesadores	27,119%	43,421%	49,541%	51,963%	54,484%	54,008%	52,337%
Matriz G	1 procesador	0.032s	0.083s	0.127s	0.194s	0.410s	0.758s	0.884s
	2 procesadores	-159,375%	-45,783%	-142,519%	-330,412%	-405,365%	-350,263%	-627,601%
	3 procesadores	-143,75%	-43,373%	-132,283%	-224,226%	-293,17%	-240,765%	-473,416%
Matriz C	1 procesador	1.672s	5.873s	24.005s	1m4.864s	4m10.506s	7m46.153s	17m44.103s
	2 procesadores	37,978%	49,821%	44,387%	48,350%	47,509%	44,563%	47,662%
	3 procesadores	56,459%	64,686%	61,708%	63,015%	65,197%	67,330%	63,823%
Matriz D	1 procesador	3.787s	14.035s	59.779s	2m43.422s	11m8.631s	21m16.356s	42m5.707s
	2 procesadores	41,167%	48,009%	49,101%	49,878%	47,330%	49,237%	47,620%
	3 procesadores	62,002%	64,760%	64,949%	65,741%	64,751%	65,467%	63,904%

Tabla 8. Resultados relativos de tiempos de ejecución de los programas paralelos frente a los programas secuenciales.

Nudos		1465	2893	5818	9778	19336	25534	36059	42250	48284
Genera aristas	1 procesador	0.298s	0.435s	0.822s	1.384s	2.782s	3.784s	5.339s	6.385s	7.210s
	2 procesadores	0.216s	0.294s	0.537s	0.85s	1.529s	1.913s	2.706s	3.162s	3.665s
	3 procesadores	0.16s	0.264s	0.468s	0.694s	1.421s	1.882s	2.569s	2.994s	3.31s
Genera caras	1 procesador	0.177s	0.304s	0.654s	1.070s	2.208s	2.844s	4.087s	5.368s	5.629s
	2 procesadores	0.173s	0.232s	0.428s	0.602s	1.161s	1.420s	2.555s	2.456s	3.327s
	3 procesadores	0.129s	0.172s	0.33s	0.514s	1.005s	1.308s	1.948s	2.125s	2.657s

Tabla 9. Resultados absolutos completos de tiempos de ejecución de los programas Genera aristas y Genera caras paralelos.

Nudos		1465	2893	5818	9778	19336	25534	36059	42250	48284
Genera aristas	1 procesador	0.298s	0.435s	0.822s	1.384s	2.782s	3.784s	5.339s	6.385s	7.210s
	2 procesadores	27,517%	32,414%	34,672%	38,584%	45,040%	49,445%	49,316%	50.478%	49.168%
	3 procesadores	46,309%	39,310%	43,066%	49,855%	48,922%	50,264%	51,882%	53.109%	54.092%
Genera caras	1 procesador	0.177s	0.304s	0.654s	1.070s	2.208s	2.844s	4.087s	5.368s	5.629s
	2 procesadores	2,260%	23,684%	34,557%	43,738%	47,418%	50,070%	37,485%	54.247%	40.895%
	3 procesadores	27,119%	43,421%	49,541%	51,963%	54,484%	54,008%	52,337%	60.414%	52.798%

Tabla 10. Resultados relativos completos de tiempos de ejecución de los programas Genera aristas y Genera caras paralelos frente a estos programas secuenciales.

De estas tablas se puede concluir que la paralelización de los programas secuenciales ha conseguido reducir sus tiempos de ejecución de forma significativa. Se observa que, en general, cuanto más fino es el mallado y más nudos tiene el mismo, más aumenta la reducción temporal. Esto era de esperar, ya que cuantos más datos haya que procesar, mayor peso tiene la parte paralela frente a la secuencial en cada programa. Además, también se aprecia que, a mayor número de procesadores, mayor es la reducción de los tiempos de ejecución, como se suponía que iba a suceder.

Sin embargo, es importante mencionar unos resultados inesperados en el programa de la Matriz G: todos los tiempos de ejecución fueron mayores en las simulaciones del programa paralelizado que en programa secuencial. Si bien es cierto que los tiempos son menores con tres procesadores que con dos, todos los resultados con dos o tres procesadores son peores que con uno únicamente.

Aunque de los resultados obtenidos para los programas secuenciales se deduce que el programa Matriz G no requiere ser paralelizado, este algoritmo fue paralelizado ya que seguía la misma estructura que los programas Matriz C y Matriz D. Sin embargo, los resultados no han sido tan positivos como en el caso de estos últimos programas, sino que, en el caso de Matriz G, los resultados son negativos.

El motivo de estos resultados desfavorables es que los cálculos y procesos detrás de la paralelización del algoritmo consumen más tiempo de lo que ahorra la propia paralelizacion del programa. El programa debe iniciar MPI, obtener el valor del número de procesos (size), el identificador de cada proceso (rank) y calcular el inicio y final del bucle *for* a paralelizar. No obstante, dentro del bucle solo se realiza una lectura de datos muy rápida, por lo que el tiempo ahorrado al dividir los datos entre varios procesadores es menor que lo que tarda el procesador en hacer esa división de datos. En consecuencia, la paralelización del programa Matriz G no es recomendable mientras que el tiempo invertido en paralelizar el algoritmo sea mayor que el tiempo ahorrado al paralelizarlo.

Otro aspecto de los resultados que merece ser comentado son aquellos valores puntuales que se alejan de la tendencia que sigue el resto de valores, como puede ser el caso

del programa Genera caras, con dos procesadores y 36059 nudos. Esta disparidad es resultado del comando de ordenación y eliminación de resultados utilizado, ya que éste no funciona con un orden establecido sino aleatorio. Por ello, hay ocasiones en las que el comando elige un orden más rápido y requiere menos tiempo para cumplir con su objetivo, mientras que en otras ocasiones elige un camino más lento. Este efecto no fue tan llamativo en el caso de la comparación de programas secuenciales originales y optimizados debido a que la diferencia temporal entre ellos era mucho mayor que en el caso de los programas paralelos respecto a los secuenciales. Por lo tanto, la variación de tiempo producida por la aleatoriedad del comando es más visible en el caso de la comparación de los programas paralelos. Sin embargo, este efecto no empaña los resultados obtenidos en los que se aprecia claramente como la paralelización de los programas secuenciales trae consigo una reducción temporal considerable en la mayoría de casos.

A continuación, se muestran los gráficos construidos a partir de las tablas anteriores:

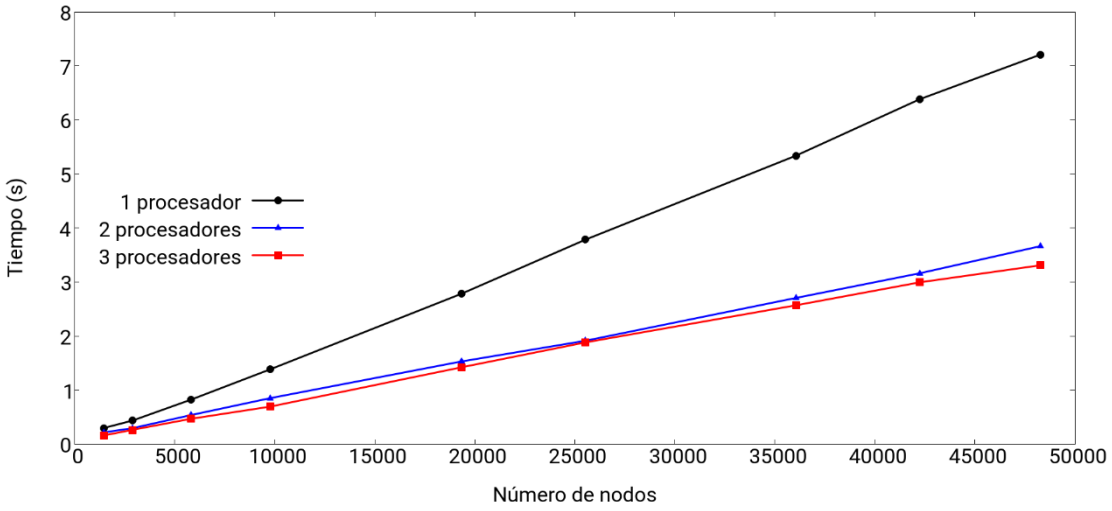

Figura 65. Gráfico comparativo de los tiempos absolutos de ejecución del programa Genera aristas en uno, dos y tres procesadores.

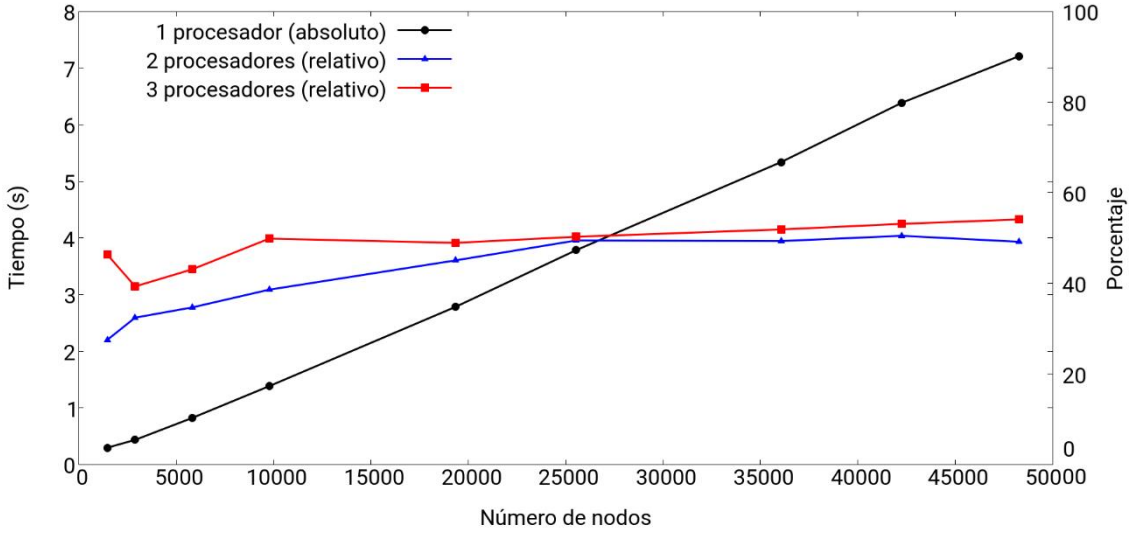

Figura 66. Gráfico de los tiempos relativos de ejecución del programa Genera aristas de dos y tres procesadores respecto a un único procesador.

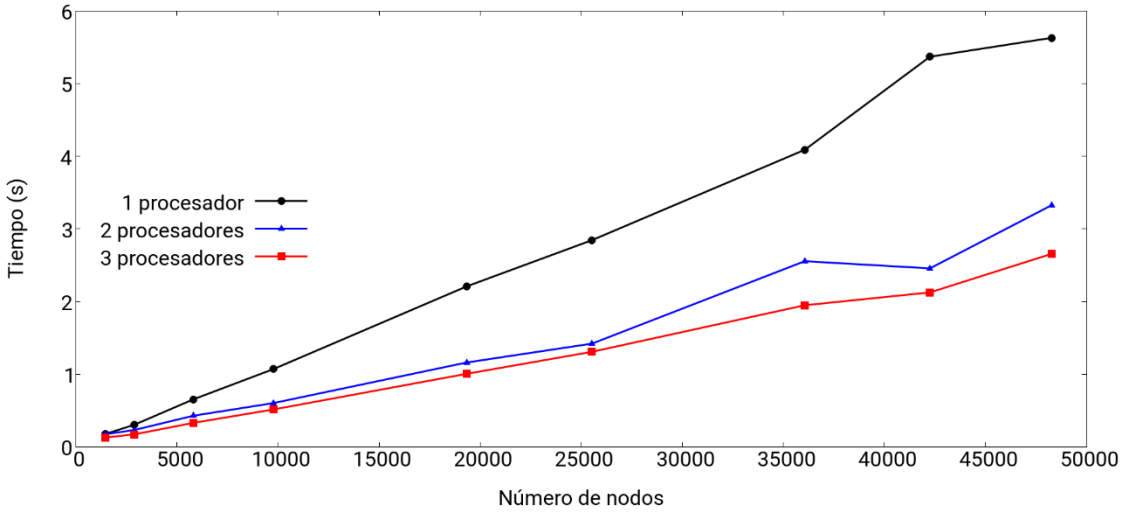

Figura 67. Gráfico comparativo de los tiempos absolutos de ejecución del programa Genera caras en uno, dos y tres procesadores.

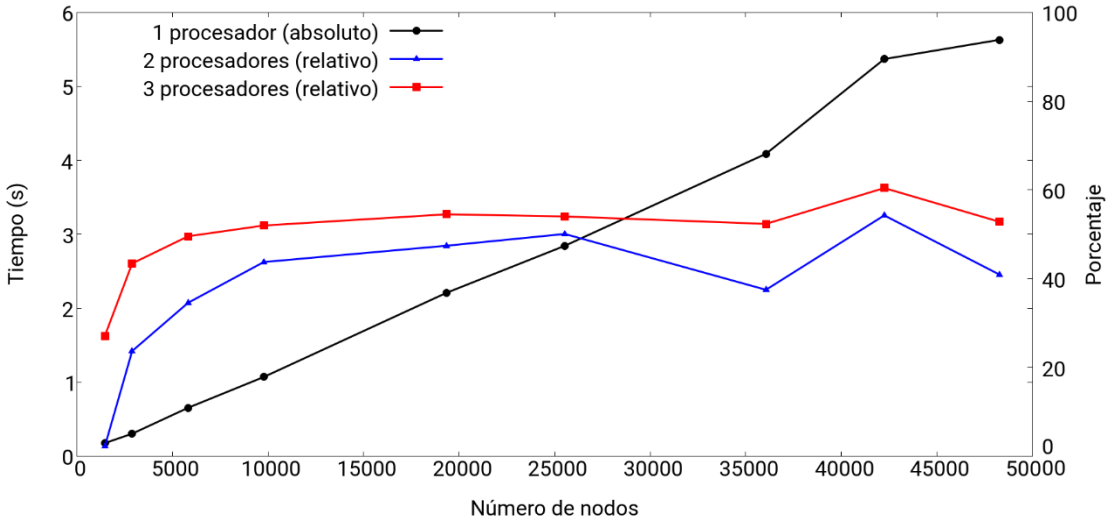

Figura 68. Gráfico de los tiempos relativos de ejecución del programa Genera caras de dos y tres procesadores respecto a un único procesador.

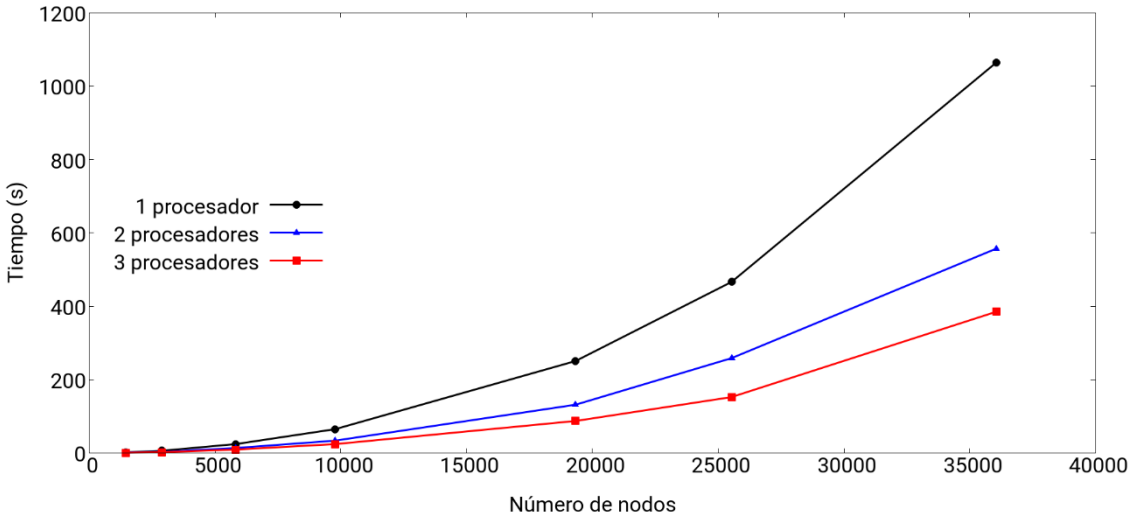

Figura 69. Gráfico comparativo de los tiempos absolutos de ejecución del programa Matriz C en uno, dos y tres procesadores.

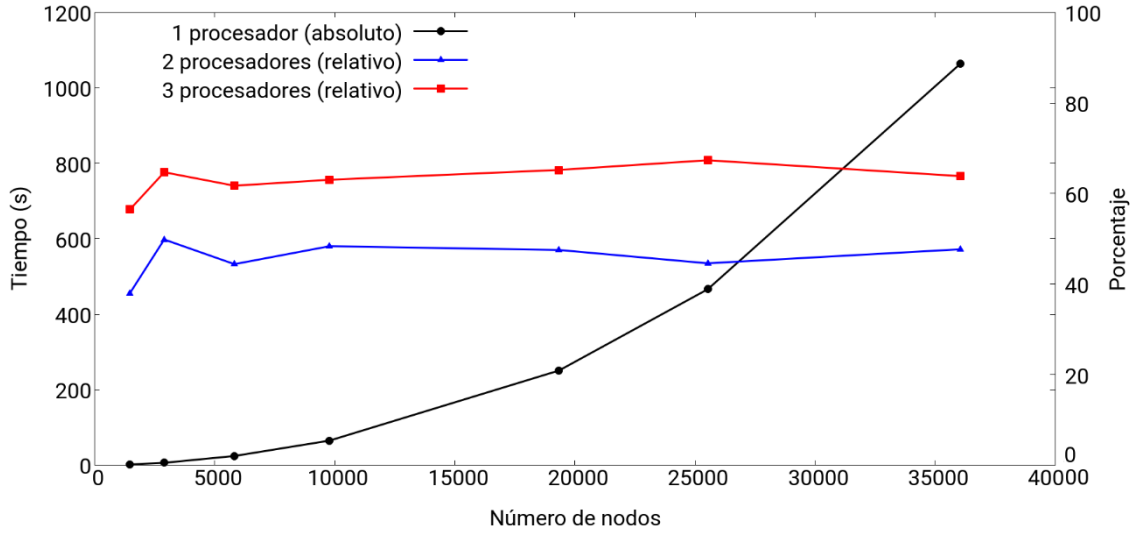

Figura 70. Gráfico de los tiempos relativos de ejecución del programa Matriz C de dos y tres procesadores respecto a un único procesador.

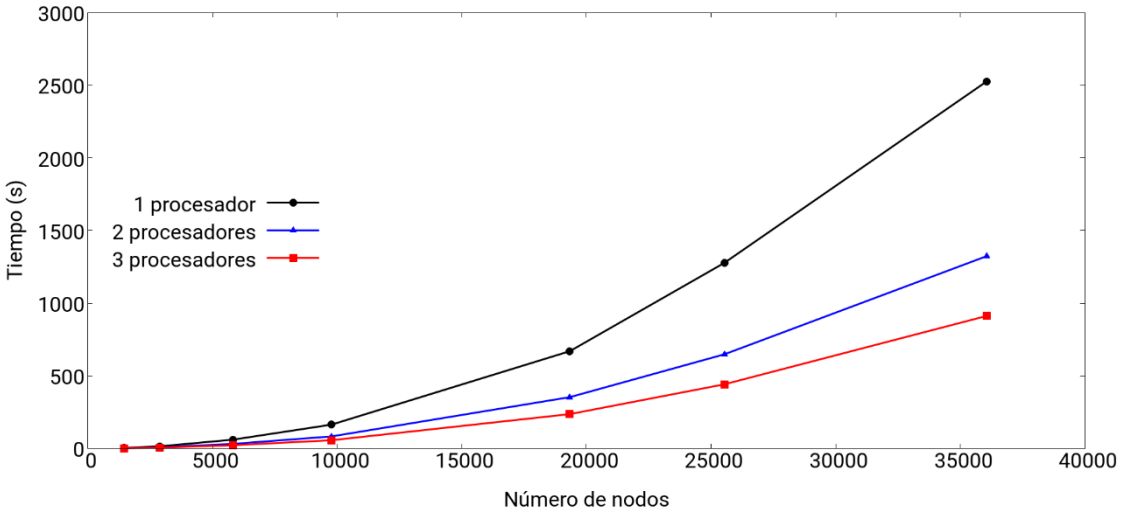

Figura 71. Gráfico comparativo de los tiempos absolutos de ejecución del programa Matriz D en uno, dos y tres procesadores.

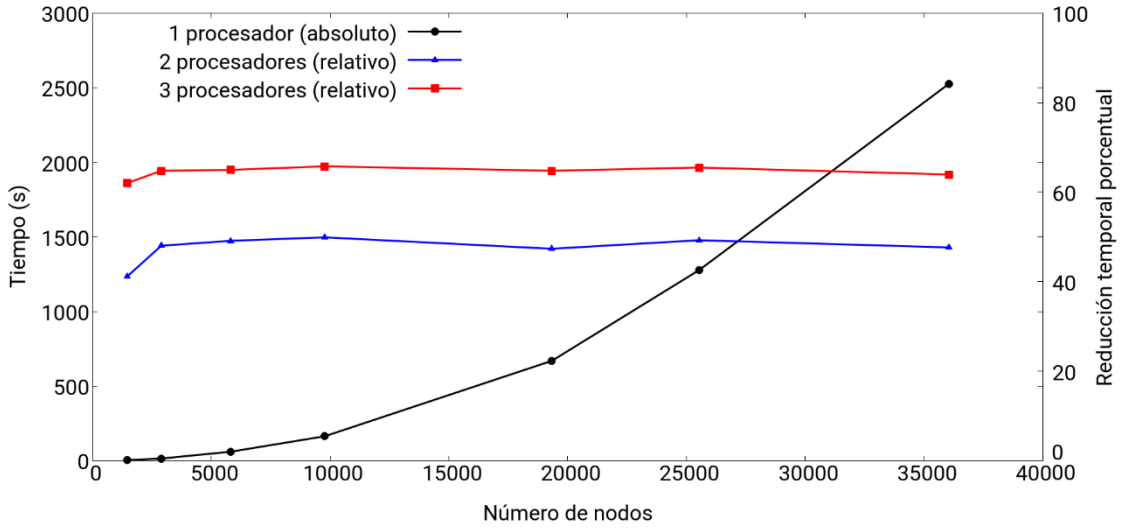

Figura 72. Gráfico de los tiempos relativos de ejecución del programa Matriz D de dos y tres procesadores respecto a un único procesador.

En los gráficos anteriores se pueden observar los tiempos de ejecución de los programas Genera aristas, Genera caras, Matriz C y Matriz D, para uno, dos y tres procesadores. Estos tiempos se muestran en valores absolutos, donde se aprecia la diferencia de tiempo entre los programas ejecutados en distinto número de procesadores; y en valores relativos, donde se observan los valores porcentuales de los tiempos de ejecución con dos y tres procesadores respecto a un único procesador.

De estos gráficos se puede deducir que la reducción temporal es mayor cuanto mayor número de procesadores se utiliza, como se puede observar tanto de forma absoluta como relativa. Además, la reducción en los tiempos de ejecución también aumenta al incrementar el detalle del mallado o el número de nudos: cuanto mayor sea el número de nudos, más tiempo se ahorrará al ejecutar el programa en paralelo.

No obstante, esta reducción temporal aumenta solo de forma absoluta: en términos relativos, la reducción temporal es aproximadamente constante al aumentar el número de nudos. En las gráficas se puede observar cómo una vez los valores de tiempo relativo se estabilizan al alcanzar un número de nudos considerable, éstos no varían de forma considerable.

9.4. Conclusiones

Los programas del pre-procesado, estudiados en el capítulo anterior, han sido paralelizados siguiendo una estrategia de descomposición de dominio al dividir los datos en partes aproximadamente iguales para cada procesador. Esto se ha logrado repartiendo los datos del bucle *for* más externo de cada programa, desde una variable de inicio hasta una de fin, dependientes del identificador de cada procesador. De esta forma, se han logrado unas reducciones temporales considerables en la mayoría de programas, normalmente oscilando entre el 40% y el 60% de reducción temporal respecto al programa secuencial como referencia.

Capítulo 10. Conclusiones y resultados

10.1. Resultados del post-procesado

El objetivo de este proyecto es el de mejorar los análisis y simulaciones de sistemas físicos mediante la paralelización de los programas empleados para ello. En este documento se ha explicado la forma de paralelizar los programas encargados del pre-procesado del análisis y de los resultados obtenidos de esta mejora.

Sin embargo, también es de vital importancia mejorar los tiempos de ejecución de los programas encargados del post-procesado de las simulaciones. Esto es así ya que una vez que se obtienen los resultados de los análisis, es necesario representar dichos resultados de forma gráfica para facilitar su comprensión. El post-procesado no solo manipula los datos obtenidos de la resolución de las ecuaciones circuitales y de las ecuaciones de Maxwell, reescritas para adaptarse al Método de la Celda, sino que además es la parte visual del proceso, que representa, literalmente, los resultados obtenidos en las simulaciones.

Puesto que el post-procesado es una parte muy importante de este proyecto, es recomendable describir brevemente su funcionamiento y mostrar los resultados finales obtenidos de una simulación.

En general, los programas del post-procesado utilizarán los resultados obtenidos del análisis del sistema, junto a datos geométricos calculados en el pre-procesado, para elaborar así un fichero final compatible con el sistema del post-procesado del software Gmsh. En este caso, se empleará el fichero de texto “solución.txt”, procedente del solver del Gmsh, y que cuenta con los datos numéricos procedentes de la resolución de la ecuación matricial 4.52. Parte de la solución se observa en la figura 71:

```
183  7.6516234525307950e-09 -7.5622749141316546e-09
184  -2.3183280347984815e-08 2.9125629131339459e-08
185  -1.2961928496370845e-08 1.5339479589254732e-08
186  0.0000000000000000e+00 0.0000000000000000e+00
187  0.0000000000000000e+00 0.0000000000000000e+00
188  0.0000000000000000e+00 0.0000000000000000e+00
189  2.8870263607768198e-09 -5.5383779520356107e-09
190  6.2735465847645489e-08 -6.5348017789017470e-08
191  -2.9416893606016280e-08 2.9577977520275265e-08
192  0.0000000000000000e+00 0.0000000000000000e+00
193  3.1761700594141804e-09 -3.4857355773402485e-09
194  2.2883568375965827e-08 -2.5010889184725798e-08
195  0.0000000000000000e+00 0.0000000000000000e+00
196  0.0000000000000000e+00 0.0000000000000000e+00
197  0.0000000000000000e+00 0.0000000000000000e+00
198  -6.2894328428046966e-09 6.0077063142871775e-09
199  -1.1461289028477650e-08 1.4834375283556236e-08
200  8.4213428990503682e-10 1.3841327126333693e-10
```

Figura 73. Detalle del fichero “solución.txt”.

En el caso de la bobina, también se utilizarán datos del pre-procesado, como pueden ser la información de volúmenes contenida en el fichero "tetra3D_01.txt" o datos sobre los nudos del fichero "nudos_msh.txt".

Posteriormente, se organizan estos datos procedentes del pro-procesado para, finalmente, obtener los ficheros de la visualización del campo magnético alrededor de la bobina y los ficheros de la visualización del flujo de corriente que circula a través de dicha bobina. Estos ficheros son tipo ".post", compatibles con el módulo del post-procesado de Gmsh. Seguidamente se muestran dos imágenes de los resultados finales del análisis del sistema compuesto por la bobina, el núcleo ferromagnético y el aire circundante. La primera imagen corresponde con la visualización de la densidad de corriente de la bobina, ya comentada en el capítulo en el que se explicaba el software Gmsh. La segunda imagen muestra el campo magnético alrededor de la bobina.

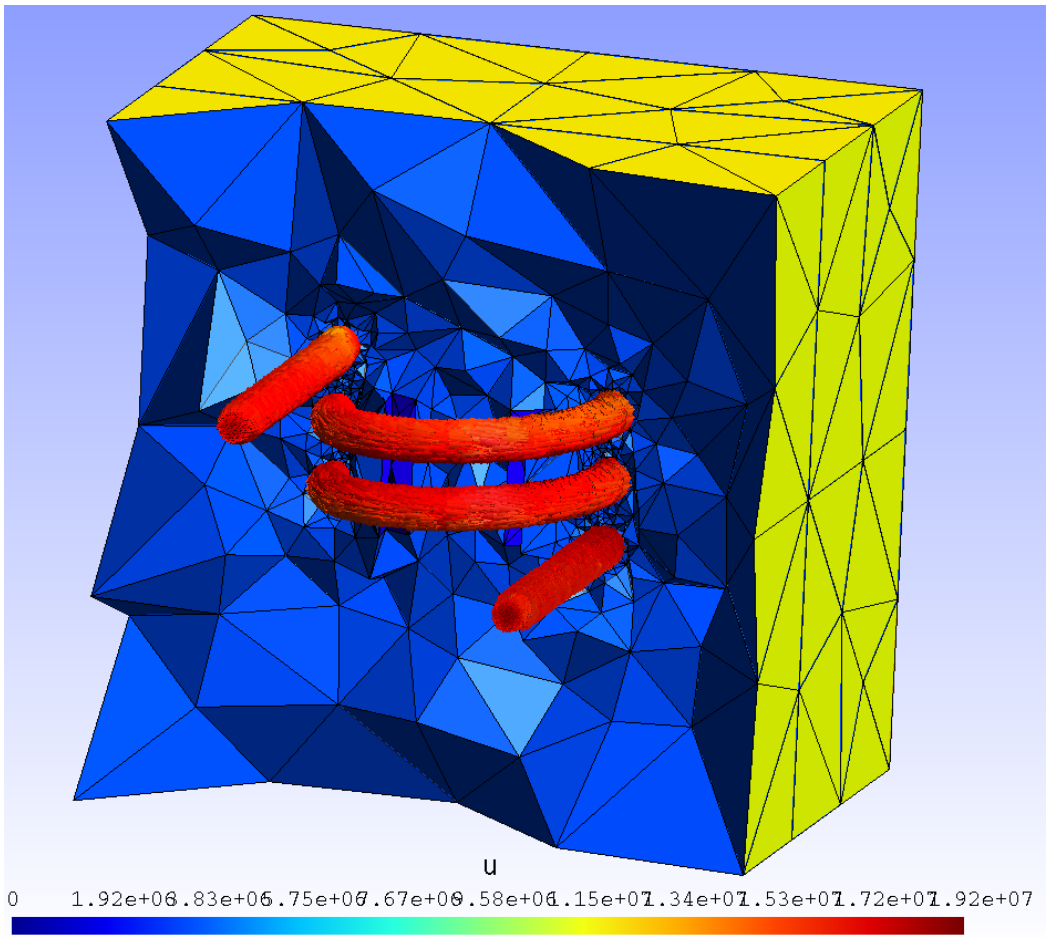

Figura 74. Visualización del post-procesado de la densidad de corriente que circula por la bobina en un problema de corrientes inducidas.

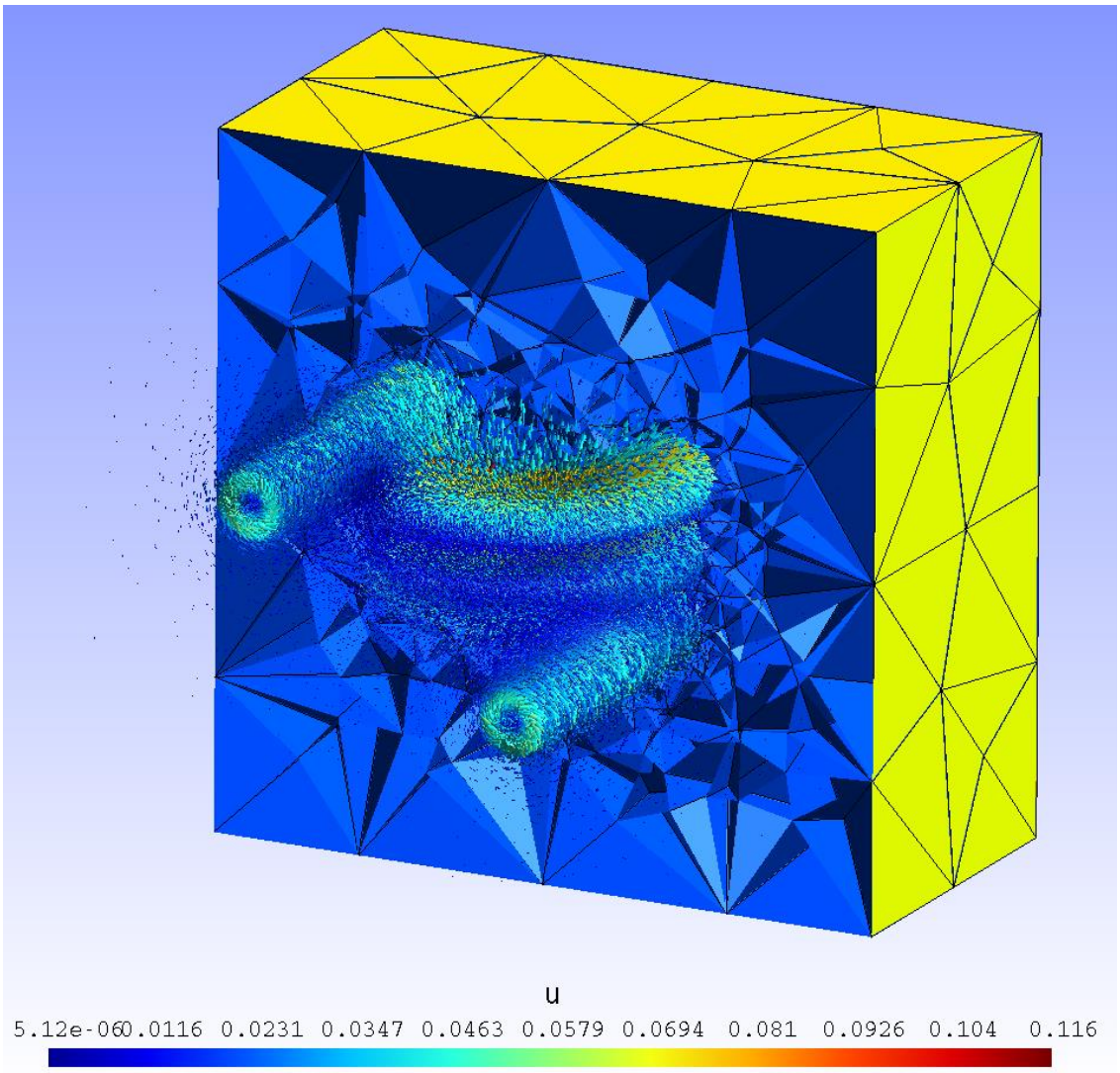

Figura 75. Visualización del post-procesado del campo magnético de la bobina en un problema de corrientes inducidas.

10.2. Revisión de objetivos y resultados

El principal objetivo de este Trabajo de Fin de Máster es el de paralelizar los programas del pre-procesado para mejorar sus tiempos de ejecución y estudiar los resultados obtenidos para determinar si la paralelización de estos programas es viable. De esta forma, se podría decidir utilizar la implementación paralela de estos programas en las simulaciones y análisis reales de sistemas físicos que se lleven a cabo.

Como objetivos adicionales de este trabajo se incluye el estudio del Método de la Celda, de las principales leyes electromagnéticas reescritas desde la perspectiva de este método numérico y de la programación en paralelo.

Como se ha podido comprobar durante capítulos anteriores de este documento, todos estos objetivos se han cumplido satisfactoriamente.

En primer lugar, se han paralelizado los programas con mayor tiempo de ejecución y con más relevancia en el pre-procesado. Los programas paralelizados, “Genera aristas”, “Genera

caras", "Matriz G", "Matriz C" y "Matriz D", permiten obtener información sobre los elementos físicos de las líneas y las superficies del mallado y las matrices de incidencia G, C y D. Por ello, estos programas son imprescindibles en el análisis de sistemas físicos basados en el Método de la Celda, ya que los datos obtenidos son utilizados posteriormente en el procesado y post-procesado.

Además, los resultados obtenidos a partir de la paralelización de dichos programas ha sido un éxito. Descartando la paralelización del programa "Matriz G", lo que no supone un inconveniente puesto que originalmente era uno de los programas con menor tiempo de ejecución, todos los programas han mejorado sus resultados temporales de forma significativa.

La paralelización de estos programas ha supuesto una reducción temporal mayor del 50% en muchos casos. Teniendo un número de nodos suficiente para obtener resultados estables, utilizando dos procesadores se tiene una reducción temporal aproximada de entre el 35% y el 55%, mientras que al emplear tres procesadores se obtiene una reducción temporal de entre el 45% y el 65%. En tiempos absolutos, esto se puede traducir en reducciones en los tiempos de ejecución mayores de 25 minutos en algunos de los casos estudiados.

Habiendo obtenido estos resultados, se puede concluir que la paralelización de los programas del pre-procesado es una opción recomendable para mejorar el detalle de las simulaciones. No solo se ahorra tiempo, sino que esta técnica puede permitir el aumentar el número de nodos del mallado a simular utilizando de forma paralela los demás procesadores con los que cuente la máquina donde se realicen estas simulaciones.

También es importante mencionar que los análisis realizados en este trabajo eran estacionarios. Sin embargo, un análisis transitorio de un sistema se podría beneficiar aún más de la paralelización de estos programas, ya que sus tiempos de ejecución aumentan proporcionalmente con el tiempo asignado al análisis transitorio. Por lo tanto, el tiempo absoluto ahorrado mediante esta filosofía paralela también aumentaría de forma proporcional.

Adicionalmente, teniendo en cuenta que el ordenador utilizado para estas simulaciones contaba únicamente con tres núcleos útiles para utilizar de forma paralela, puesto que el cuarto procesador se encuentra ocupado con el Sistema Operativo del ordenador, se pueden pronosticar incluso mejores resultados si se utilizase un ordenador con mejores prestaciones.

Finalmente, cabe añadir que los objetivos adicionales marcados para este Trabajo de Fin de Máster también han sido cumplidos con éxito. Por un lado, en los primeros capítulos teóricos de este documento se explica el Método de la Celda y la reformulación de las ecuaciones de Maxwell y ecuaciones circuitales empleadas posteriormente, lo que facilita la comprensión del marco teórico detrás de este trabajo. Por otro lado, en los siguientes capítulos de esta memoria se explican las principales herramientas software utilizadas durante la realización de este trabajo. Al añadir ejemplos básicos se muestra la progresiva adquisición de competencias en los programas de mallado y programación paralela, además de ayudar a comprender el funcionamiento de estas herramientas.

10.3. Líneas futuras

La paralelización de los programas encargados del análisis de sistemas basado en el Método de la Celda cuenta con una buena proyección de futuro debido a que es el único estudio

que se haya podido encontrar hasta el momento que relacione este método numérico con la computación paralela y debido a los buenos resultados obtenidos.

Un siguiente paso para avanzar en esta técnica y confirmar las expectativas generadas en este trabajo será el de llevar a cabo simulaciones en paralelo en un ordenador con mejores prestaciones. Específicamente, se debería utilizar un ordenador con muchos más procesadores disponibles para comprobar que al aumentar el número de procesadores, el rendimiento de la computación paralela aumenta al disminuir los tiempos de ejecución.

Este objetivo también se puede lograr mediante un clúster de ordenadores. En vez de utilizar un único ordenador con un mayor número de procesadores, se pueden utilizar los procesadores de un conjunto de ordenadores, conectados mediante una red local, que compartan los datos de entrada y las tareas a realizar.

Adicionalmente, se estudiará en detalle el procesado y el post-procesado del Método de la Celda con el objetivo de paralelizar el código de estos programas. De esta forma, se intentará reducir el tiempo total de ejecución del programa completo.

Finalmente, se ampliarán las métricas asociadas a tiempos de sistemas paralelos. En este trabajo se utilizó la relación directa entre el tiempo de ejecución de los programas secuenciales como referencia y del tiempo de los programas paralelos. Por ello, en un futuro se estudiarán otros métodos de comparación de los tiempos de ejecución, como son los análisis de speed-up, de eficiencia o de escalabilidad.

Bibliografía

[1] L. Simón Rodríguez, “Aportaciones al Método de la Celda en el Diseño y Análisis de un Modelo de Máquina Rotativa Trifásica de Inducción Magnética”, Tesis Doctoral, Marzo 2015.

[2] P. I. González Domínguez, “Aportaciones al Diseño de Máquinas Eléctricas de Inducción mediante el Método de la Celda. Análisis Térmico y Electromagnético”, Tesis Doctoral, Noviembre 2015.

[3] E. Tonti, "On the Geometrical Structure of Electromagnetism", Gravitation, Electromagnetism and Geometrical Structures, 1995.

[4] E. Tonti, "A Direct Discrete Formulation For The Wave Equation", International Conference on Theoretical and Computational Acoustics, Mayo 1999.

[5] E. Tonti, “A Direct Discrete Formulation of Field Laws: The Cell Method”, Computer Modeling in Engineering and Sciences, 2001.

[6] E. Tonti, “Why starting from differential equations for computational physics?”, Journal of Computational Physics, 2014.

[7] M. Repetto, y F. Trevisan, "Global formulation for 3D magneto-static using flux and gauged potential approaches", International Journal for Numerical Methods in Engineering, 2002.

[8] M. Bullo, F. Dughiero, M. Guarnieri, y E Tittonel, “Isotropic and Anisotropic Electrostatic Field Computation by Means of the Cell Method”, IEEE Transactions On Magnetics, Marzo 2004.

[9] M. Bullo, F. Dughiero, M. Guarnieri, y E. Tittonel, "Nonlinear Coupled Thermo-Electromagnetic Problems With the Cell Method", IEEE Transactions On Magnetics, Abril 2006.

[10] M. Bullo, V. D’Ambrosio, F. Dughiero, y M. Guarnieri, "Coupled Electrical and Thermal Transient Conduction Problems With a Quadratic Interpolation Cell Method Approach”, IEEE Transactions On Magnetics, Abril 2006.

[11] M. Bullo, V. D’Ambrosio, F. Dughiero, y M. Guarnieri, "A 3-D Cell Method Formulation for Coupled Electric and Thermal Problems", IEEE Transactions On Magnetics, Abril 2007.

[12] J. W. Parker, R. D. Ferraro y P. C. Liewer, “An Examination of Finite Element Formulations and Parameters for Accurate Parallel Solution of Electromagnetic Scattering Problems”, IEEE 1990.

[13] A. Chatterjee y J.L. Volakis, “Parallel Computation of 3D Electromagnetic Scattering using Finite Elements and Conformal ABCs”, IEEE, Transactions On Magnetics, Septiembre 1994.

[14] A. Amin, P. Sadayappan y M. Gudavalli, “A Clustered Reduced Communication Element by Element Preconditioned Conjugate Gradient Algorithm for Finite Element Computations” IEEE, 1994.

[15] N. Kapadia, B. Lichtenberg, J. A. B. Fortes, J. L. Gray, H. J. Siegel, y K. J. Webb " Parailel Solution of Unstructured Sparse Finite Element Equations”, IEEE, 1995.

[16] U. D. Navsariwala' y S. Gedney, “An Unconditionally Stable Parallel Finite Element Time Domain Algorithm”, IEEE, 1996.

[17] C. Vollaire, L. Nicolas y A. Nicolas, “Finite Elements and Absorbing Boundary Conditions for Scattering Problems on a Parallel Distributed Memory Computer”, IEEE Transactions On Magnetics, Marzo 1997.

[18] C. Vollaire y L. Nicolas, “Implementation of a Finite Element and Absorbing Boundary Conditions Package on a Parallel Shared Memory Computer”, IEEE Transactions On Magnetics, Septiembre 1998.

[19] S. McFee, y D. Ma, "Generalized h-p Triangles and Tetrahedra for Adaptive Finite Element Analysis in Parallel Processing Environments", IEEE Transactions On Magnetics, Marzo 2004.

[20] P. Sypek, M. Wiktor, y M. Mrozowski, "Parallel Implementation of the Matrix Formulation of the FDTD Scheme", The International Conference on “Computer as a Tool”, EUROCON 2007.

[21] S. Aliabadi, K. Shujaee y T. Tezduyar, “Parallel Simulation of Two-phase Flow Problems Using the Finite Element Method”, Frontiers of Massively Parallel Computation, 1999.

[22] T. Dziubak, M. Wiktor, S. Orłowski y T. P. Stefanski, “Acceleration of the DGF-FDTD Method on GPU Using the CUDA Technology”, IEEE, Abril 2015.

[23] T. P. Stefanski, T. Dziubak y S. Orłowski, “Parallel Implementation of the DGF-FDTD Method on GPU Using the CUDA Technology”, IEEE, 2016.

[24] Y. Takahashi, T. Tokumasu, M. Fujita, T. Iwashita, H.Nakashima, S. Wakao, y K. Fujiwara, “Time-Domain Parallel Finite-Element Method for Fast Magnetic Field Analysis of Induction Motors”, IEEE Transactions On Magnetics, Mayo 2013.

[25] C. Geuzaine y J.-F. Remacle, "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities", International Journal for Numerical Methods in Engineering, 2009.

[26] MPI Forum, “MPI: A Message-Passing Interface Standard, Version 3.1”, Junio 2015.

[27] Argonne National Laboratory, “PETSc Users Manual, Revision 3.7”, Abril 2016.

[28] Tonti, “Finite Formulation Of The Electromagnetic Field”, Progress In Electromagnetics Research, 2001.

[29] C. Geuzaine y J.-F. Remacle, "Gmsh Reference Manual”, Mayo 2017.

[30] P. Dular y C. Geuzaine, “GetDP Reference Manual”, Mayo 2017.

[31] C. Fu, "Parallel Finite Element Computation Based On Coarse-Fine Mesh Formulation", International Conference on Computer Science and Software Engineering, 2008.

[32] L. Kleinrock y J.-H. Huang, “On parallel processing systems: Amdahl's law generalized and some results on optimal design”, IEEE Transactions on Software Engineering, Mayo 1992.

[33] S. Balay, W. D. Gropp, L. C. McInnes y B. F. Smith, "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries", Modern Software Tools in Scientific Computing, 1997.

[34] B. J. Gough, “An Introduction to GCC - for the GNU compilers gcc and g++”, Network Theory Limited, Agosto 2005.

Anexo 1. Publicaciones

Finite Formulation in Parallel Computation. Application to Electromagnetic Field in 3-D

Adrián de Pablo Sánchez, José Miguel Monzón-Verona, Leopoldo Simón Rodríguez, Pablo Gonzalez Domínguez
Departamento de Ingeniería Eléctrica
Instituto Universitario de Microelectrónica Aplicada
Las Palmas de Gran Canaria, Spain

***Abstract*—The Cell Method is a proposed numerical solution for electromagnetic field applications. It is based on Finite Formulation, so it is directly related to experimentation due to the global variables attached to the points, lines, surfaces and volumes of this theory. In this project, the Cell Method is used to simulate a real electromagnetic problem: a coil with a cylindrical metallic core and the air of their surroundings model. This paper shows the time consume problem faced during the pre-processing of the information obtained in the simulations and the proposed solution by the parallelization of the algorithms used. The preliminary results of the parallel pre-processing will be shown and compared with the serial procedure ones.**

Keywords—Cell Method; electromgnetic field; pre-processing; parallelization; MPI; speedup.

I. INTRODUCTION

The Cell Method (CM) is a numerical method that uses discrete formulation of field equations [1]. It is based on the use of global variables and a pair of cell complexes, a primal and a dual, where each element has an orientation that can be internal or external.

This method associates physical properties of electromagnetism in form of global variables, commonly known as integral variables, with spatial and temporal elements such as points (P), lines (L), surfaces (S), volumes (V), instants (I) and time intervals (T), the basic elements of this theory [2].

Incidence matrices are built with the incidence numbers between two elements of different dimension of a cell complex. If the elements are not incident, the incidence number is zero. If they are incident and their orientations are compatible, the incidence number is 1. If they are incident but their orientations are not compatible, the incidence number is -1. Incidence matrices can be formed between lines and points (matrix G), between faces and lines (matrix C), and between volumes and faces (matrix D).

II. FIELD LAWS IN FINITE FORM

Experimentation can lead to rewrite fundamental equations of electromagnetism in a finite form. By this, global variables are linked to physical and temporal elements of this theory.

Magnetic Gauss' law:

$$\phi[\partial V, I] = 0 \tag{1}$$

Faraday's electromagnetic induction law:

$$\varepsilon[\partial S, T] = \phi[S, I^-] - \phi[S, I^+] \tag{2}$$

Faraday's electrostatic induction law/electric Gauss' law:

$$\psi[\partial \tilde{V}, \tilde{I}] = Q^c[\tilde{V}, \tilde{I}] \tag{3}$$

Maxwell - Ampère's law:

$$F_m[\partial \tilde{S}, \tilde{T}] = \psi[\tilde{S}, \tilde{I}^+] - \psi[\tilde{S}, \tilde{I}^-] + Q^f[\tilde{S}, \tilde{T}] \tag{4}$$

Conservation of charge:

$$Q^f[\partial \tilde{V}, \tilde{I}] = Q^c[\tilde{V}, \tilde{I}^-] - Q^c[\tilde{V}, \tilde{I}^+] \tag{5}$$

III. APPLICATION

Finite formulation, and specifically CM, can be applied to solve physical problems based on electromagnetism. In this case, a coil model is being simulated to obtain the results of different electromagnetic equations form its entire domain.

The simulations have been realized with an Intel Core i7 - 3820, of 3.6 GHz frequency, 32 GB memory, 4 cores and 8 threads. It uses a NVIDIA GM107 (GeForce GTX 750 Ti) GPU with 2 GB.

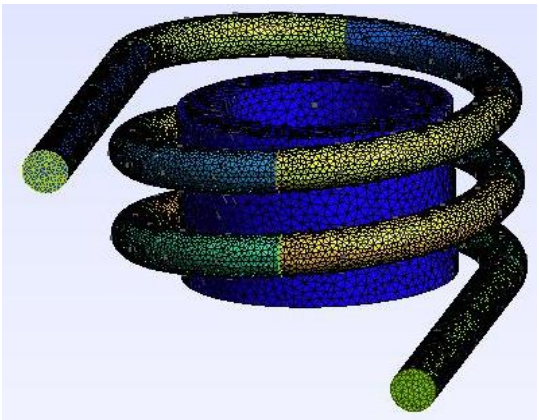

Fig. 1. 3D simulation of a coil with a cylindrical metallic core

Table 1. Speedup of parallel pre-processing algorithms using a serial algorithm as reference

Nodes	Lines Generator		Surfaces Generator		C Matrix		D Matrix	
	2 processors	3 processors	2 processors	3 processors	2 processors	3 processors	2 processors	3 processors
1,465	27.517%	46.309%	2.260%	27.119%	37.978%	56.459%	41.167%	62.002%
2,893	32.414%	39.310%	23.684%	43.421%	49.821%	64.686%	48.009%	64.760%
5,818	34.672%	43.066%	34.557%	49.541%	44.387%	61.708%	49.101%	64.949%
9,778	38.584%	49.855%	43.738%	51.963%	48.350%	63.015%	49.878%	65.741%
19,336	45.040%	48.922%	47.418%	54.484%	47.509%	65.197%	47.330%	64.751%
25,534	49.445%	50.264%	50.070%	54.008%	44.563%	67.330%	49.237%	65.467%
36,059	49.316%	51.882%	37.485%	52.337%	47.662%	63.823%	47.620%	63.904%

The 3D simulation of a coil model has been implemented in Gmsh, a three-dimensional finite element grid generator software with a build-in CAD engine and post-processor [3].

As it can be seen in Fig. 1, the simulation of the system includes a coil, a cylindrical metallic core and the surrounding air inside a cubic volume.

First time, this pre-processing was done following a serial philosophy. However, to improve the detail of the simulations, it is necessary to increase the number of points and physical elements in general. The number of nodes of the simulations varied from 1,465 to 36,059 nodes. This increase of elements leads to an exponential escalation of the pre-processing programs' execution times, from 10.423 seconds to 5329.586 seconds. As a result, a parallel adaption of the pre-processing algorithms was implemented using MPI (Message-Passing Interface) inside PETSc environment [4].

IV. Results

The objective of this study is to reduce the execution time of the pre-processing of a model by the parallelization of its algorithms.

The results of the parallel implementation of the 3D coil pre-processing algorithms can be seen in Table 1. The achieved speedup generally varies from 30% to 65%. Normally, speedup improves with the increase of the number of nodes and the number of processors.

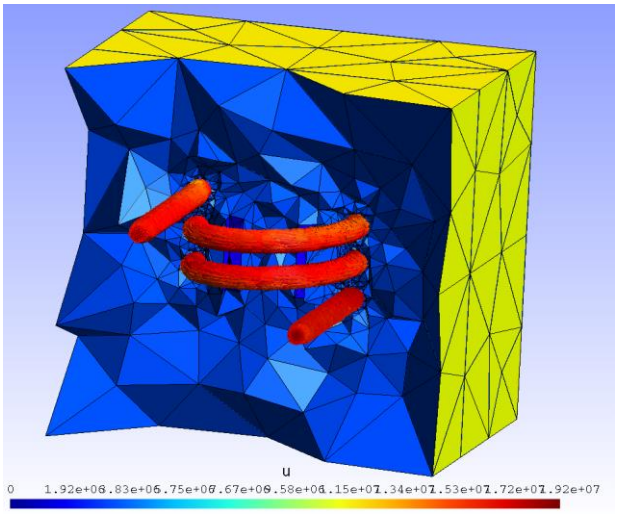

Fig. 2. Current density view obtained in post-processing

The final results of the simulations realized on the coil model can be seen in Fig. 2, a view of the current density along the coil when electric potential is applied to its extremes. This image is obtained from the post-processing of the model.

V. Conclusion

The parallelization of pre-processing algorithms for electromagnetic model simulations based on CM has been implemented.

This parallel philosophy has lead to speedups generally higher than 40%, up to 65%. This achievement allows simulations to progress in detail and in lower execution times. Future simulations on a computer with a higher number of processors will probably show even better results of parallelization. Further work will try to improve parallelization algorithms, what could reduce execution times even more.

References

[1] E. Tonti, "On the Geometrical Structure of Electromagnetism", Gravitation, Electromagnetism and Geometrical Structures, pp.281-308 1995.

[2] E. Tonti, "A Direct Discrete Formulation For The Wave Equation", International Conference on Theoretical and Computational Acoustics, May 1999.

[3] C. Geuzaine and J.-F. Remacle, " Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities", International Journal for Numerical Methods in Engineering 79(11), pp. 1309-1331, 2009.

[4] S. Balay, W. D. Gropp, L. C. McInnes and B. F. Smith, "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries", Modern Software Tools in Scientific Computing, pp 163-202, 1997.

[5] M. Repetto, and F. Trevisan, "Global formulation for 3D magneto-static using flux and gauged potential approaches", International Journal for Numerical Methods in Engineering, 2002.

[6] M. Bullo, F. Dughiero, M. Guarnieri, and E. Tittonel, "Nonlinear Coupled Thermo-Electromagnetic Problems With the Cell Method", IEEE Transactions On Magnetics, Vol. 42, No. 4, April 2006.

[7] M. Bullo, V. D'Ambrosio, F. Dughiero, and M. Guarnieri, "A 3-D Cell Method Formulation for Coupled Electric and Thermal Problems", IEEE Transactions On Magnetics, Vol. 43, No. 4, April 2007.

[8] S. McFee, and D. Ma, "Generalized h-p Triangles and Tetrahedra for Adaptive Finite Element Analysis in Parallel Processing Environments", IEEE Transactions On Magnetics, Vol. 40, No. 2, March 2004.

[9] C. Fu, "Parallel Finite Element Computation Based On Coarse-Fine Mesh Formulation", International Conference on Computer Science and Software Engineering, 2008.

Thermal Constitutive Matrix Applied to Asynchronous Electrical Machine Using the Cell Method

Pablo González Domínguez, Adrián de Pablo Sánchez, José Miguel Monzón-Verona, Leopoldo Simón Rodríguez
Departamento de Ingeniería Eléctrica
Instituto Universitario de Microelectrónica Aplicada
Las Palmas de Gran Canaria, Spain

***Abstract*— This work demonstrates the equivalence of two constitutive equations. One used in Fourier heat transmission equation, the other one used in electric conduction equation, both based on the numerical method of the Cell Method, using the Finite Formulation (FF-CM). A 3-D pure heat conduction model is proposed. The temperatures are in stay steady and there are not internal heat sources. The obtained results are compared with an equivalent model developed using the Finite Elements Method (FEM) in 2-D.**

Keywords—Finite Formulation; Cell Method; Thermal Constitutive Matrix; 3-D heat tansmission models; Finite Elements Method.

I. INTRODUCTION

The Heat transmission equation in FF-CM is expressed in (1) as:

$$[G]^T\{[M_\lambda][G][\tau]\} + \left[M_{Cp}\right] {}^{d\tau}/_{dt} = [W] \quad (1)$$

Two constitutive matrices are observed: $[M_\lambda]$ and $\left[M_{Cp}\right]$. The constitutive matrix $[M_\lambda]$ belongs to Fourier heat transmission. The constitutive matrix $\left[M_{Cp}\right]$ belongs to the Heat transmission equation with change of temperature.

Fourier heat transmission equation applied to stator-airgap-rotor is as followings:

$$[G]^T\{[M_\lambda][G][\tau]\} \quad (2)$$

The heat sources, corresponding to Joule effect and magnetic hysteresis and eddy currents, are represented as following:

$$[W] \quad (3)$$

The changes in the internal energy depending of the constructive materials of the asynchronous machines are represented in the following expression:

$$\left[M_{Cp}\right] {}^{d\tau}/_{dt} \quad (4)$$

We establish an analogy between the thermal conduction and electrical conduction as is indicated in (5).

$$-[G]^T\{-[M_\lambda][G][\tau]\}=\left[\tilde{D}\right][q_\lambda] \;\Leftrightarrow\; -[G]^T\{[M_\sigma][G][v]\}=\left[\tilde{D}\right][J] \quad (5)$$

Fourier heat transmission equation is treated in 2-D and 3-D. The cartesian absolute coordinates of the cell are converted to local standardized coordinates. The origin is one of the four vertex of the tetrahedron used as cell [1], [2], [3]. The dual variables are projected to the primal using geometrical methods, planar symmetries and axisymmetric symmetries are used and quadratic interpolations are applied to obtain distributions of temperature [1], [2], [3], [4], [5].

Assuming the analogy indicated in (5), thus:

$$[M_\lambda] \;\Leftrightarrow\; [M_\sigma] \quad (6)$$

Using the expression proposed by [6], the matrices exposed in (6) are topological equivalent:

$$\lambda\left(\frac{1}{V_t}\vec{\tilde{S}}_i\vec{\tilde{S}}_j\right) \Leftrightarrow \sigma\left(\frac{1}{V_t}\vec{\tilde{S}}_i\vec{\tilde{S}}_j\right) \quad (7)$$

La matriz $[M_\sigma]$ debe calcularse para los fenómenos electrotérmicos ocurridos en la máquina asíncrona. Entonces, ¿por qué hacer dos veces los mismos cálculos? Para confirmar esta suposición se diseñaron varios experimentos numéricos. Los resultados obtenidos se contrastaron con los resultados obtenidos del mismo modelo desarrollado con Métodos de Elementos Finitos (FEM) en 2-D.

II. NUMERICAL EXPERIMENTS

A. *Numerical experiments desciption*

In the numerical experiments, it is supposed there is a pure conduction of heat, there are not inner heat sources and the temperatures are stationaries:

$$[M_\lambda][G][\tau] = [q_\lambda] \quad (8)$$

Two boundary conditions, type Dirichlet, were established. Two temperatures, τ_1 and τ_2, were fixed in two faces of cylinder (rotor or stator). This faces are axial sides of de cylinder. The rest of de faces, top face and down face of cylinder, were considered as perfect heat insulators. It was considered different structural materials with different thermal conductivities coefficients.

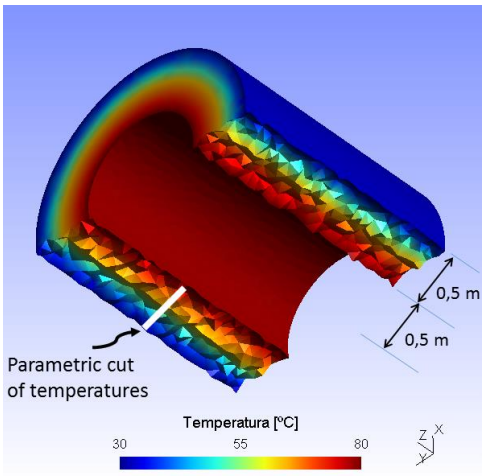

Fig. 1. Rotor with two different structural materials, also two different thermal conductivity coefficients.

B. Results

Different meshes are used in the model. The temperatures are calculated with FF-CM. The obtained temperatures are compared with the temperatures obtained with a FEM model using software FEMM ®.

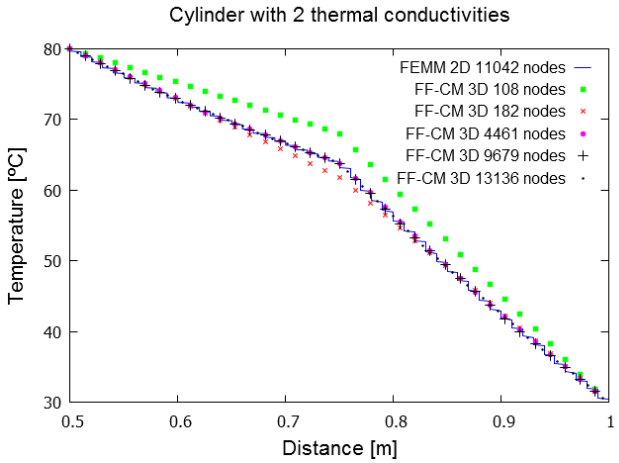

Fig. 2. Temperatures obtened in a cylinder with two thermal conductiviy coeficients. The results FEMM have been obtened using FEM. The results FF-MC have been obtened using Finite Formulation with Cell Method.

III. Conclusiones

The FF-CM is a strong method because we face a 3-D model in FF-CM, with a not very dense tetrahedral mesh, with an axisymmetric model calculated with FEM and very dense triangular mesh. Errors produced are not significant. The number of nodes is the number of the unknowns and equations to resolve. There is not a significant gain in precision increasing the density of the mesh. In the Fig. 2, with 9,679 nodes and 13,136 nodes the errors are similar.

References

[1] Bullo, M., D'Ambrosio, V., Dughiero, F. & Guarnieri, M., 2006a. Coupled electrical and thermal transient conduction problems with a quadratic interpolation cell method approach. Magnetics, IEEE Transactions on, April, 42(4), pp. 1003-1006

[2] Bullo, M., D'Ambrosio, V., Dughiero, F. & Guarnieri, M., 2006b. A 3D Cell Method Formulation for Coupled Electric and Thermal Problems b). Miami,FL, USA, IEEE, pp. 7-7.

[3] Bullo, M., D'Ambrosio, V., Dughiero, F. & Guarnieri, M., 2007. A 3-D Cell Method Formulation for Coupled Electric and Thermal Problems. Magnetics, IEEE Transactions on, April, 43(4), pp. 1197-1200.

[4] Tonti, E., 2000. Formulazione finita delle equazioni di campo: Il Metodo delle Celle. Atti del XIII Convegno Italiano di Meccanica Computazionale, Brescia, Italy.

[5] Tonti, E., 2001. A direct discrete formulation of field laws: The cell method. CMES- Computer Modeling in Engineering and Sciences, 2(2), pp. 237-258.

[6] Specogna, R. & Trevisan, F., 2005. Discrete constitutive equations in A-Chi geometric eddy-current formulation. IEEE Trans. Magn, 41(4), pp. 1259-1263.

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

Anexo 2. Códigos de programas

```
//PROGRAMA GENERA ARISTAS OPTIMIZADO


#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <fstream>
using namespace std;

 int main(int argc, char **args)
{
int indice,nodoA,nodoB,nodoC,nodoD,atrib;
int
filas_tetraedros,contador2,contador3,c1,c2,c,dato,volumenes,nudos,estimacion_aristas;
int A,B,C,D;

ifstream PARAMETER_read;//fichero de entrada
ofstream PARAMETER_write;//fichero de salida
ofstream PARAMETER1_write;
ifstream PARAMETER5_read;//fichero datos.ene

PARAMETER_read.open("tetra3D_01.txt",ios :: in);
PARAMETER_write.open("aris3D_01_1.txt",ios :: out);
PARAMETER5_read.open("datos_fisicos_msh.txt",ios::in);
PARAMETER5_read>>nudos>>volumenes>>dato>>dato>>dato;
PARAMETER5_read.close();


filas_tetraedros=volumenes;
int contador_matriz_apoyo;
contador_matriz_apoyo=0;

for(contador2=1;contador2<=filas_tetraedros;++contador2){
   PARAMETER_read>>nodoA>>nodoB>>nodoC>>nodoD>>atrib;
   A=nodoA;B=nodoB;C=nodoC;D=nodoD;


if (A<B) {c1=A;c2=B ;}
else {
         c1=B;c2=A;}

contador_matriz_apoyo=contador_matriz_apoyo+1;
PARAMETER_write<<c1<<" "<<c2<<endl;
c=contador_matriz_apoyo+1;


if (B<C) {   c1=B;c2=C;}
 else {
                        c1=C;c2=B;}

contador_matriz_apoyo=contador_matriz_apoyo+1;
PARAMETER_write<<c1<<" "<<c2<<endl;
c=contador_matriz_apoyo+1;


if (C<D) {   c1=C;c2=D;}
 else {
         c1=D;c2=C;}

contador_matriz_apoyo=contador_matriz_apoyo+1;
PARAMETER_write<<c1<<" "<<c2<<endl;
c=contador_matriz_apoyo+1;
```

```
if (D<A) {   c1=D;c2=A;}
 else {
         c1=A;c2=D;}

contador_matriz_apoyo=contador_matriz_apoyo+1;
PARAMETER_write<<c1<<" "<<c2<<endl;
c=contador_matriz_apoyo+1;


if (A<C) {   c1=A;c2=C;}
 else {
         c1=C;c2=A;}

contador_matriz_apoyo=contador_matriz_apoyo+1;
PARAMETER_write<<c1<<" "<<c2<<endl;
c=contador_matriz_apoyo+1;


c1=B;c2=D;
if (B<D) {   c1=B;c2=D;}
 else {
         c1=D;c2=B;}

contador_matriz_apoyo=contador_matriz_apoyo+1;
PARAMETER_write<<c1<<" "<<c2<<endl;
c=contador_matriz_apoyo+1;
}

PARAMETER_write.close();
system("cat aris3D_01_1.txt | sort | uniq >aris3D_01.txt"); //Se ordenan y eliminan los
elementos repetidos
system ("wc -l aris3D_01.txt > numero.aristas");//Se cuentan las aristas y se guarda el
nÃºmero en el fichero

return 1;
}
```

```
//PROGRAMA GENERA CARAS OPTIMIZADO


#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <fstream>

using namespace std;
int main(int argc, char **args)

{
int indice,nodoA,nodoB,nodoC,nodoD,atrib;
int filas_tetraedros,contador2,contador3,c,dato,volumenes,nudos,estimacion_caras;
int c1,c2,c3;
int A,B,C,D;

ifstream PARAMETER_read;//fichero de entrada
ofstream PARAMETER_write;//fichero de salida
ofstream PARAMETER1_write;
PARAMETER_read.open("tetra3D_01.txt",ios :: in);
PARAMETER_write.open("trian3D_01_1.txt",ios::out);
PARAMETER1_write.open("numero.triangulos",ios :: out);
ifstream PARAMETER5_read;//fichero datos.ene
PARAMETER5_read.open("datos_fisicos_msh.txt",ios::in);
PARAMETER5_read>>nudos>>volumenes>>dato>>dato>>dato;
PARAMETER5_read.close();

filas_tetraedros=volumenes;
estimacion_caras=nudos*12.5;
int contador_matriz_apoyo_tri;
int matriz_apoyo_tri_c1[estimacion_caras];//matriz de apoyo para triangulos
int matriz_apoyo_tri_c2[estimacion_caras];
int matriz_apoyo_tri_c3[estimacion_caras];
contador_matriz_apoyo_tri=0;

for(contador2=1;contador2<=filas_tetraedros;++contador2){
  PARAMETER_read>>nodoA>>nodoB>>nodoC>>nodoD>>atrib;
  A=nodoA;B=nodoB;C=nodoC;D=nodoD;
  int aux;

c1=A;c2=B;c3=C;

//Ordena de menor a mayor
if(c1>c2)
{
aux=c2;
c2=c1;
c1=aux;
}

if(c2>c3)
{
aux=c2;
c2=c3;
c3=aux;
}

if(c1>c2)
{
aux=c1;
c1=c2;
c2=aux;
}
```

```
PARAMETER_write<<c1<<" "<<c2<<" "<<c3<<endl;
contador_matriz_apoyo_tri=contador_matriz_apoyo_tri+1;
c=contador_matriz_apoyo_tri+1;

c1=B;c2=C;c3=D;

//Ordena de menor a mayor
if(c1>c2)
{
aux=c2;
c2=c1;
c1=aux;
}

if(c2>c3)
{
aux=c2;
c2=c3;
c3=aux;
}

if(c1>c2)
{
aux=c1;
c1=c2;
c2=aux;
}

PARAMETER_write<<c1<<" "<<c2<<" "<<c3<<endl;
contador_matriz_apoyo_tri=contador_matriz_apoyo_tri+1;
c=contador_matriz_apoyo_tri+1;

c1=C;c2=D;c3=A;

//Ordena de menor a mayor
if(c1>c2)
{
aux=c2;
c2=c1;
c1=aux;
}

if(c2>c3)
{
aux=c2;
c2=c3;
c3=aux;
}

if(c1>c2)
{
aux=c1;
c1=c2;
c2=aux;
}

PARAMETER_write<<c1<<" "<<c2<<" "<<c3<<endl;
contador_matriz_apoyo_tri=contador_matriz_apoyo_tri+1;
c=contador_matriz_apoyo_tri+1;

c1=B;c2=D;c3=A;

//Ordena de menor a mayor
if(c1>c2)
{
aux=c2;
```

```
c2=c1;
c1=aux;
}

if(c2>c3)
{
aux=c2;
c2=c3;
c3=aux;
}

if(c1>c2)
{
aux=c1;
c1=c2;
c2=aux;
}

PARAMETER_write<<c1<<" "<<c2<<" "<<c3<<endl;
contador_matriz_apoyo_tri=contador_matriz_apoyo_tri+1;
c=contador_matriz_apoyo_tri+1;
}

PARAMETER_write.close();
PARAMETER1_write<<contador3<<endl;
PARAMETER1_write.close();
system("cat trian3D_01_1.txt | sort | uniq >trian3D_01.txt"); //Se ordenan y eliminan
los elementos repetidos
system ("wc -l  trian3D_01.txt> numero.caras");//Se cuentan las caras y se guarda el
nÃºmero en el fichero

return 1;
}//Fin del main()
```

```
//PROGRAMA GENERA ARISTAS PARALELO


static char help[] = "";

#include <petscmat.h>
#undef __FUNCT__
#define __FUNCT__ "main"
 #include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <fstream>

using namespace std;
 int main(int argc, char **args)
{
//InicializaciÃ³n de PETSc
PetscErrorCode  ierr;
 int rank, size;
 ierr = PetscInitialize(&argc,&args,(char*)0,help);CHKERRQ(ierr);
   ierr = MPI_Comm_rank(PETSC_COMM_WORLD,&rank);CHKERRQ(ierr);
  ierr = MPI_Comm_size(PETSC_COMM_WORLD,&size);CHKERRQ(ierr);

 //ObtenciÃ³n de variables de inicio y fin
    int RPN(int rank, int size, int N,int *I,int *F);
 int I,F,N;


int indice,nodoA,nodoB,nodoC,nodoD,atrib;
int
filas_tetraedros,contador2,contador3,c1,c2,c,dato,volumenes,nudos,estimacion_aristas;
int A,B,C,D;

ifstream PARAMETER_read;//fichero de entrada
ofstream PARAMETER_write;//fichero de salida
ofstream PARAMETER1_write;
ifstream PARAMETER5_read;//fichero datos.ene

PARAMETER_read.open("tetra3D_01.txt",ios :: in);
PARAMETER_write.open("aris3D_01_1.txt",ios::out | ios :: trunc);
PARAMETER_write.close();
PARAMETER_write.open("aris3D_01_1.txt",ios :: out | ios :: app);
PARAMETER5_read.open("datos_fisicos_msh.txt",ios::in);
PARAMETER5_read>>nudos>>volumenes>>dato>>dato>>dato;
PARAMETER5_read.close();


filas_tetraedros=volumenes;
 N=filas_tetraedros;
RPN(rank, size,N ,&I,&F);
//Ajuste de variables de inicio y fin
I=I+1;
F=F+1;

//Ajuste de lectura de datos hasta la variable de inicio
for(contador2=0;contador2<I;++contador2){
PARAMETER_read>>nodoA>>nodoB>>nodoC>>nodoD>>atrib;
}

 for(contador2=I;contador2<=F;++contador2){
  PARAMETER_read>>nodoA>>nodoB>>nodoC>>nodoD>>atrib;


A=nodoA;B=nodoB;C=nodoC;D=nodoD;
```

```
if (A<B) {c1=A;c2=B ;}
    else {
         c1=B;c2=A;}
        PARAMETER_write<<c1<<" "<<c2<<endl;

if (B<C) {   c1=B;c2=C;}
 else {
         c1=C;c2=B;}
        PARAMETER_write<<c1<<" "<<c2<<endl;

if (C<D) {   c1=C;c2=D;}
 else {
         c1=D;c2=C;}
        PARAMETER_write<<c1<<" "<<c2<<endl;

if (D<A) {   c1=D;c2=A;}
 else {
         c1=A;c2=D;}
        PARAMETER_write<<c1<<" "<<c2<<endl;

if (A<C) {   c1=A;c2=C;}
 else {
         c1=C;c2=A;}
        PARAMETER_write<<c1<<" "<<c2<<endl;

c1=B;c2=D;

if (B<D) {   c1=B;c2=D;}
 else {
         c1=D;c2=B;}
        PARAMETER_write<<c1<<" "<<c2<<endl;


}//tetra

int j;
if(rank==0){
        PARAMETER_write.close();
system("awk '!seen[$0]++' aris3D_01_1.txt  > aris3D_01.txt");//Se ordenan y eliminan
los elementos repetidos
system ("wc -l aris3D_01.txt > numero.aristas");//Se cuentan las aristas y se guarda el
nÃºmero en el fichero
}
  ierr = PetscFinalize();
  return 0;
}


int RPN(int rank, int size, int N,int *I,int *F)
{
//*************** FUNCIONES PROPIAS DE ESTE PROGRAMA *********************
/*************************************************************************/
//
//Esta funciÃ³n divide intervalos segun rank size y N y dev. I, F
/*************************************************************************/

if (rank !=0) {*I=rank*int(floor(N/size))+1;}
else {*I=0;}
if (rank !=(size-1)){*F=(rank+1)*int(floor(N/size));}
else {*F=N-1 ;}
return 1;
} //FIN
```

```
//PROGRAMA GENERA CARAS PARALELO


static char help[] = "";
#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <petscmat.h>
#include <fstream>
#include <limits>

    using namespace std;
int main(int argc, char **args)
{
int indice,nodoA,nodoB,nodoC,nodoD,atrib;
int filas_tetraedros,contador2,contador3,c,dato,volumenes,nudos,estimacion_caras;
int c1,c2,c3;
int A,B,C,D;

//InicializaciÃ³n de PETSc
PetscErrorCode  ierr;
 int rank, size;
 ierr = PetscInitialize(&argc,&args,(char*)0,help);CHKERRQ(ierr);
   ierr = MPI_Comm_rank(PETSC_COMM_WORLD,&rank);CHKERRQ(ierr);
  ierr = MPI_Comm_size(PETSC_COMM_WORLD,&size);CHKERRQ(ierr);

  //ObtenciÃ³n de variables de inicio y fin
 int RPN(int rank, int size, int N,int *I,int *F);
 int I,F,N;

ifstream PARAMETER_read;//fichero de entrada
ofstream PARAMETER_write;//fichero de salida
PARAMETER_read.open("tetra3D_01.txt",ios :: in);
PARAMETER_write.open("trian3D_01_1.txt",ios::out | ios :: trunc);
PARAMETER_write.close();
PARAMETER_write.open("trian3D_01_1.txt",ios::out | ios :: app);
ifstream PARAMETER5_read;
PARAMETER5_read.open("datos_fisicos_msh.txt",ios::in);
PARAMETER5_read>>nudos>>volumenes>>dato>>dato>>dato;
PARAMETER5_read.close();
filas_tetraedros=volumenes;

 N=filas_tetraedros;
RPN(rank, size,N ,&I,&F);

//Ajuste de variables de inicio y fin
I=I+1;
F=F+1;
printf( "I= %d F= %d\n", I, F );

//Ajuste de lectura de datos hasta la variable de inicio
for(contador2=0;contador2<I;++contador2){
PARAMETER_read>>nodoA>>nodoB>>nodoC>>nodoD>>atrib;
}


for(contador2=I;contador2<=F;++contador2){
     PARAMETER_read>>nodoA>>nodoB>>nodoC>>nodoD>>atrib;
A=nodoA;B=nodoB;C=nodoC;D=nodoD;
int aux;
c1=A;c2=B;c3=C;

//Ordena de menor a mayor
if(c1>c2)
```

```
{
aux=c2;
c2=c1;
c1=aux;
}

if(c2>c3)
{
aux=c2;
c2=c3;
c3=aux;
}

if(c1>c2)
{
aux=c1;
c1=c2;
c2=aux;
}
    PARAMETER_write<<c1<<" "<<c2<<" "<<c3<<endl;

c1=B;c2=C;c3=D;

//Ordena de menor a mayor
if(c1>c2)
{
aux=c2;
c2=c1;
c1=aux;
}

if(c2>c3)
{
aux=c2;
c2=c3;
c3=aux;
}

if(c1>c2)
{
aux=c1;
c1=c2;
c2=aux;
}

    PARAMETER_write<<c1<<" "<<c2<<" "<<c3<<endl;

c1=C;c2=D;c3=A;


//Ordena de menor a mayor
if(c1>c2)
{
aux=c2;
c2=c1;
c1=aux;
}

if(c2>c3)
{
aux=c2;
c2=c3;
c3=aux;
}

if(c1>c2)
{
```

```
aux=c1;
c1=c2;
c2=aux;
}

    PARAMETER_write<<c1<<" "<<c2<<" "<<c3<<endl;

c1=B;c2=D;c3=A;

//Ordena de menor a mayor
if(c1>c2)
{
aux=c2;
c2=c1;
c1=aux;
}

if(c2>c3)
{
aux=c2;
c2=c3;
c3=aux;
}

if(c1>c2)
{
aux=c1;
c1=c2;
c2=aux;
}
    PARAMETER_write<<c1<<" "<<c2<<" "<<c3<<endl;
}

if(rank==0){
       PARAMETER_write.close();
system("awk '!seen[$0]++' trian3D_01_1.txt >trian3D_01.txt"); //Se ordenan y eliminan
los elementos repetidos
system ("wc -l  trian3D_01.txt> numero.triangulos"); //Se cuentan las caras y se guarda
el nÃºmero en el fichero
}
   ierr = PetscFinalize();
return 0;
}//Fin del main()

int RPN(int rank, int size, int N,int *I,int *F)
{
//*************** FUNCIONES PROPIAS DE ESTE PROGRAMA *********************
/*************************************************************************/
//
//Esta funciÃ³n divide intervalos segun rank size y N y dev. I, F
/*************************************************************************/

if (rank !=0) {*I=rank*int(floor(N/size))+1;}
else {*I=0;}
if (rank !=(size-1)){*F=(rank+1)*int(floor(N/size));}
else {*F=N-1 ;}
return 1;
} //FIN
```

```
//PROGRAMA MATRIZ G PARALELO


static char help[] = "";
#include <petscmat.h>
#undef __FUNCT__
#define __FUNCT__ "main"
#include<iostream>
#include<fstream>
 using namespace std;
int main(int argc, char **args)
{
//InicializaciÃ³n de PETSc
PetscErrorCode  ierr;
 int rank, size;
 ierr = PetscInitialize(&argc,&args,(char*)0,help);CHKERRQ(ierr);
   ierr = MPI_Comm_rank(PETSC_COMM_WORLD,&rank);CHKERRQ(ierr);
  ierr = MPI_Comm_size(PETSC_COMM_WORLD,&size);CHKERRQ(ierr);


 int i,j,nn,na,nc,nv,ari0,ari1;

  //Indica lo que estÃ¡ calculando
  cout<<"Leyendo datos ..."<<endl;

 //Lee el nÃºmero de: nudos, aristas,caras y volumenes. estÃ¡n contenidos en el archivo
NACV.TXT
 ifstream p1("nacv3D.txt");
 while (!p1.eof()) {p1 >> nn >> na >> nc >> nv;}
 p1.close();

  //ObtenciÃ³n y ajustes de variables de inicio y fin
  int RPN(int rank, int size, int N,int *I,int *F);
 int I,F,N;
 N=nn;
 RPN(rank, size,N ,&I,&F);
 if(rank==(size-1)) {F=F+1;}
  if(rank!=0) {I=I-1;}

 //La aristas estÃ¡n contenidas en el archivo ARIS_01.TXT
 ifstream p2("aris3D_01.txt");
ofstream PARAMETER_write;
PARAMETER_write.open("matriz_G.txt",ios::out | ios :: trunc);
PARAMETER_write.close();
PARAMETER_write.open("matriz_G.txt",ios :: out | ios :: app);


  //Indica lo que estÃ¡ calculando
  cout<<"Calculando matriz G ..."<<endl;

 //Bucle de aristas. El contador debe empezar en i=1 pues las matriz G comienza
 //por la fila 1, columna j=1.
 for(i=1;i<na+1;i++)
 {
  //Lee los nudos de las aristas
  p2>>ari0>>ari1;

  //Inicializa el contador de nudos en el nudo de la Ãºltima arista analizada
 j=ari0;

  for (j=I;j<F;j++) //Bucle de nudos
  {
   //Analiza el primer nudo de la arista i
   if (ari0==j)
   {
    //El nudo inicial de la arista es nudo incidente (incidencia positiva -1)
```

```
    PARAMETER_write << i << " " << j << " " << -1 <<endl;

    //El nudo final de la arista es nudo no incidente (incidencia negativa
+1)
        PARAMETER_write << i << " " << ari1 << " " << +1 <<endl;

        //No sigue examinando la arista i
        j=nn+10;

   }//Bucle if
  }//Fin bucle de nudos
 }//Fin bucle de aristas

 //Cierra el archivo MATRIZ_G.TXT
 PARAMETER_write.close();

 //Cierra el archivo ARIS3D_01.TXT
 p2.close();
  ierr = PetscFinalize();
  return 0;
}


 int RPN(int rank, int size, int N,int *I,int *F)

{
//*************** FUNCIONES PROPIAS DE ESTE PROGRAMA *********************
/*************************************************************************/
//
//Esta funciÃ³n divide intervalos segun rank size y N y dev. I, F
/*************************************************************************/

if (rank !=0) {*I=rank*int(floor(N/size))+1;}
else {*I=0;}
if (rank !=(size-1)){*F=(rank+1)*int(floor(N/size));}
else {*F=N-1 ;}
return 1;
} //FIN
```

```
//PROGRAMA MATRIZ C PARALELO


static char help[] = "";
#include <petscmat.h>
#include<iostream>
#include<fstream>
#undef __FUNCT__
#define __FUNCT__ "main"
using namespace std;

int IsMember(int **b, int, int, int, int, int, int *v, int *c);

int main(int argc, char **args)
{
//InicializaciÃ³n de PETSc
 PetscErrorCode  ierr;
 int rank, size;
 ierr = PetscInitialize(&argc,&args,(char*)0,help);CHKERRQ(ierr);
 ierr = MPI_Comm_rank(PETSC_COMM_WORLD,&rank);CHKERRQ(ierr);
 ierr = MPI_Comm_size(PETSC_COMM_WORLD,&size);CHKERRQ(ierr);

 int i,j,nn,na,nc,nv,tri0,tri1,tri2,v,c,k,n, NC;

 //Indica lo que estÃ¡ calculando
  cout<<"Leyendo datos ..."<<endl;

 //Lee el nÃºmero de: nudos, aristas,caras y volumenes. EstÃ¡n contenidos en el archivo
NACV.TXT
 ifstream p1("nacv3D.txt");
 while (!p1.eof()) {p1 >> nn >> na >> nc >> nv;}
 p1.close();

 //La aristas estÃ¡n contenidas en el archivo ARIS_01.TXT
 int **ari;

 //Establece el vector filas de la matriz ARI de aristas
 ari = new int*[na];

 //Establece la matriz ARI con columnas
 for(i=0;i<na;i++) ari[i]=new int[2];

 //ObtenciÃ³n y ajustes de variables de inicio y fin
 int RPN(int rank, int size, int N,int *I,int *F);
 int I,F,N;
 N=na;
 RPN(rank, size,N ,&I,&F);
 if(rank==(size-1)) {F=F+1;}
 if(rank!=0) {I=I-1;}

 //Carga la matriz ARI
 ifstream p2("aris3D_01.txt");
 i=0;
 while (!p2.eof())
 {
  for (i=0;i<na;i++) {p2>>ari[i][0]>>ari[i][1];}
 }
 p2.close();

 //Abre el archivo que contiene las caras o trÃÃ¡ngulos: TRIAN_01.TXT
 ifstream p3("trian3D_01.txt");

  //Indica lo que estÃ¡ calculando
  cout<<"Calculando matriz C ..."<<endl;

 //Crea el archivo de la matriz de incidencias CARAS-ARISTAS en forma
```

```
 //de matriz dispersa C[i,j,v]. i es el valor de la cara (fila de C),
 //j es el valor de arista (columna de C), v toma los valores {-1,0,1})
 //segÃºn sea la incidencia negativa, no existente o positiva, respectivamente.

ofstream PARAMETER_write;
PARAMETER_write.open("matriz_C.txt",ios::out | ios :: trunc);
PARAMETER_write.close();
PARAMETER_write.open("matriz_C.txt",ios :: out | ios :: app);

 i=0;
 N=nc;

 while (!p3.eof() && i<nc)
 {
  p3>>tri0>>tri1>>tri2; //Lectura de los nudos del triÃ¡ngulo

    //SÃ³lo imprime cuando haya incidencia, es decir: v!=0
    //El contador i+1 se incrementa e 1 porque las matrices empiezan en
    //el nÃºmero 1, mientras que los contadores en C++ empiezan en cero

        //Arista 1 orientaciÃ³n POSITIVA y NEGATIVA
    //Busca las aristas de la cara en la lista de aristas
    IsMember(ari,tri0,tri1,I,F,na,&v,&c);
    //if(v!=0){fs << i+1 << " " << c <<" " << v <<endl;}
    if(v!=0){PARAMETER_write<< i+1 << " " << c <<" " << v <<endl;}

    //Arista 2 orientaciÃ³n POSITIVA y NEGATIVA
    //Busca las aristas de la cara en la lista de
aristas
    IsMember(ari,tri1,tri2,I,F,na,&v,&c);
    if(v!=0){PARAMETER_write<< i+1 << " " << c <<" " << v <<endl;}

    //Arista 3 orientaciÃ³n POSITIVA y NEGATIVA
    //Busca las aristas de la cara en la lista de aristas
    IsMember(ari,tri2,tri0,I,F,na,&v,&c);
    if(v!=0){PARAMETER_write<< i+1 << " " << c <<" " << v <<endl;}

    //Incrementa el contador de caras
    i++;
 }//Fin bule WHILE

 //Cierra el archivo TRIAN_01.TXT
 p3.close();

 //Cierra el archivo MATRIZ_C.TXT
PARAMETER_write.close();
 //Libera la memoria de la matriz
 delete [] ari;

  ierr = PetscFinalize();
  return 0;
}

 //*************** FUNCIONES PROPIAS DE ESTE PROGRAMA *********************
int IsMember(int **b, int n1, int n2, int I, int F, int na, int *v, int *c)
/**************************************************************************/
/* Esta funciÃ³n localiza las aristas de una cara en el listado de aristas
/* ARI del mallado.
/*
/* Si la arista coincide, entonces v=1 y c=j, j corresponderÃ¡ a la posiciÃ³n
/* donde se encuentra la arista i en el listado de aristas ARI
/**************************************************************************/
{//Comienzo de la funciÃ³n

    int j;
    *v=0;
    *c=0;
```

```
 //Busca en la lista de elementos ARI (aristas)
  for(j=I;j<F;j++)
  {
   //La arista extraida del triangulo coincide con una arista de la lista
   //de aristas.
   if(n1==b[j][0] && n2==b[j][1])
   {
    //Se trata de una arista perteneciente a la cara y de orientaciÃ³n positiva
    //Se suma 1 a j porque las matrices empiezan en 1 y no en cero. Por lo tanto
    //c=j+1
    *v=1; *c=j+1;
   }

   //La arista extraida del triangulo coincide con una arista de la lista
   //de aristas, pero estÃ¡ orientada de forma inversa.
   if(n2==b[j][0] && n1==b[j][1])
   {
    //Se trata de una arista perteneciente a la cara y de orientaciÃ³n negativa
    //Se suma 1 a j porque las matrices empiezan en 1 y no en cero. Por lo tanto
    //c=j+1
    *v=-1; *c=j+1;

   }
  }//Fin bucle j
 return 0;
}//Fin de la funciÃ³n

int RPN(int rank, int size, int N,int *I,int *F)
{
//*************** FUNCIONES PROPIAS DE ESTE PROGRAMA *********************
/*************************************************************************/
//
//Esta funciÃ³n divide intervalos segun rank size y N y dev. I, F
/*************************************************************************/

if (rank !=0) {*I=rank*int(floor(N/size))+1;}
else {*I=0;}
if (rank !=(size-1)){*F=(rank+1)*int(floor(N/size));}
else {*F=N-1 ;}
return 1;

} //FIN
```

```
//PROGRAMA MATRIZ D PARALELO

static char help[] = "";
#include <petscmat.h>
#undef __FUNCT__
#define __FUNCT__ "main"
#include<iostream>
#include<fstream>

int IsMembTet(int **b, int, int, int, int, int, int, int *v, int *c);

using namespace std;
int main(int argc, char **args)
{
//InicializaciÃ³n de PETSc
 PetscErrorCode  ierr;
 int rank, size;
 ierr = PetscInitialize(&argc,&args,(char*)0,help);CHKERRQ(ierr);
 ierr = MPI_Comm_rank(PETSC_COMM_WORLD,&rank);CHKERRQ(ierr);
 ierr = MPI_Comm_size(PETSC_COMM_WORLD,&size);CHKERRQ(ierr);

 int i,j,nn,na,nc,nv,tet0,tet1,tet2,tet3,pm,v,c;

 //Indica lo que estÃ¡ calculando
 cout<<"Leyendo datos ..."<<endl;

 //Lee el nÃºmero de: nudos, aristas,caras y volumenes. EstÃ¡n contenidos en el archivo
NACV.TXT
 ifstream p1("nacv3D.txt");
 while (!p1.eof()) {p1 >> nn >> na >> nc >> nv;}
 p1.close();

 //La caras estÃ¡n contenidas en el archivo TRIAN_01.TXT
 int **tri;

 //Establece el vector filas de la matriz TRI de caras
 tri = new int*[nc];

 //Establece la matriz TRI con columnas
 for(i=0;i<nc;i++) tri[i]=new int[3];

 //ObtenciÃ³n y ajuste de variables de inicio y fin
 int RPN(int rank, int size, int N,int *I,int *F);
 int I,F,N;
 N=nc;
 RPN(rank, size,N ,&I,&F);
 if(rank==(size-1)) {F=F+1;}
 if(rank!=0) {I=I-1;}
 printf( "I= %d F= %d\n", I, F );

 //Carga la matriz TRI
 ifstream p2("trian3D_01.txt");
 i=0;
 while (!p2.eof())
 {
  for (i=0;i<nc;i++) {p2>>tri[i][0]>>tri[i][1]>>tri[i][2];}
 }
 p2.close();

 //Indica lo que estÃ¡ calculando
 cout<<"Calculando matriz D ..."<<endl;

 //Abre el archivo que contiene las volÃºmenes o tetraedros: TETRA_01.TXT
 ifstream p3("tetra3D_01.txt");

 //Crea el archivo de la matriz de incidencias VOLUMENES-CARAS en forma
```

```
 //de matriz dispersa D[i,j,v]. i es el valor del volumen (fila de C),
 //j es el valor de la cara (columna de C), v toma los valores {-1,0,1})
 //segÃºn sea la incidencia negativa, no existente o positiva, respectivamente.
 //ofstream fs("matriz_D.txt");

ofstream PARAMETER_write;
PARAMETER_write.open("matriz_D.txt",ios::out | ios :: trunc);
PARAMETER_write.close();
PARAMETER_write.open("matriz_D.txt",ios :: out | ios :: app);


 //Lee los nudos del tetraedro desde el archivo TETRA_01.TXT
 i=0;
 while (!p3.eof() && i<nv)
 {
  //Lectura de los nudos del tetraedro. La variable PM se refiere a propiedades del
material.
  p3>>tet0>>tet1>>tet2>>tet3>>pm;

  //SÃ³lo imprime cuando haya incidencia, es decir: v!=0
  //El contador i+1 se incrementa e 1 porque las matrices empiezan en
  //el nÃºmero 1, mientras que los contadores en C++ empiezan en cero

    //Cara 1 orientaciÃ³n POSITIVA y NEGATIVA
    //Busca las caras del tetraedro en la lista de caras TRI
    IsMembTet(tri,tet0,tet1,tet3,I,F,nc,&v,&c);
    if(v!=0){PARAMETER_write << i+1 << " " << c <<" " << v <<endl;}

    //Cara 2 orientaciÃ³n POSITIVA y NEGATIVA
    //Busca las caras del tetraedro en la lista de caras TRI
    IsMembTet(tri,tet0,tet2,tet1,I,F,nc,&v,&c);
    if(v!=0){PARAMETER_write << i+1 << " " << c <<" " << v <<endl;}

    //Cara 3 orientaciÃ³n POSITIVA y NEGATIVA
    //Busca las caras del tetraedro en la lista de caras TRI
    IsMembTet(tri,tet0,tet3,tet2,I,F,nc,&v,&c);
    if(v!=0){PARAMETER_write << i+1 << " " << c <<" " << v <<endl;}

    //Cara 4 orientaciÃ³n POSITIVA y NEGATIVA
    //Busca las caras del tetraedro en la lista de caras TRI
    IsMembTet(tri,tet1,tet2,tet3,I,F,nc,&v,&c);
    if(v!=0){PARAMETER_write << i+1 << " " << c <<" " << v <<endl;}

    //Incrementa el contador de volumenes (tetraedros)
    i++;

 }//Fin bule WHILE

 //Cierra el archivo TETRA_01.TXT
 p3.close();

 //Cierra el archivo MATRIZ_D.TXT
 PARAMETER_write.close();

 //Libera la memoria de la matriz
 delete [] tri;

  ierr = PetscFinalize();
  return 0;
}

//*************** FUNCIONES PROPIAS DE ESTE PROGRAMA *********************
int IsMembTet(int **b, int n1, int n2, int n3, int I, int F, int nc, int *v, int *c)
/*****************************************************************************/
/* Esta funciÃ³n localiza las caras de un tetraedro en el listado de caras
/* TRI del mallado.
/*
```

```
/* Si la cara coincide, entonces v=1 y c=j, j corresponderÃ¡ a la posiciÃ³n
/* donde se encuentra la cara i en el listado de caras TRI
/*********************************************************************/

{//Comienzo de la funciÃ³n
 int j,flag;

  //Iniicaliza los contadores de incidencias v,c
  *v=0; *c=0;

  //Busca en la lista de elementos TRI (caras)
  for(j=I;j<F;j++)
  {
   //Control de comparaciÃ³n
   flag=0;

   //La cara extraida del tetraedro coincide con una cara de la lista de TRI.
   if((n1==b[j][0] && n2==b[j][1] && n3==b[j][2]) ||
      (n2==b[j][0] && n3==b[j][1] && n1==b[j][2]) ||
      (n3==b[j][0] && n1==b[j][1] && n2==b[j][2]))
   {
    //Se trata de una cara perteneciente al tetraedro y de orientaciÃ³n positiva
    //Se suma 1 a j porque las matrices empiezan en 1 y no en cero. Por lo tanto
    //c=j+1
    *v=1; *c=j+1; flag=1;
   }

   //La cara extraida del tetraedro coincide con una cara de la lista TRI
   //de caras, pero estÃ¡ orientada de forma inversa.
   if(flag==0)
   {
              if((n1==b[j][0] && n3==b[j][1] && n2==b[j][2]) ||
              (n2==b[j][0] && n1==b[j][1] && n3==b[j][2]) ||
              (n3==b[j][0] && n2==b[j][1] && n1==b[j][2]))
              {
              //Se trata de una cara perteneciente al tetraedro y de orientaciÃ³n
negativa
              //Se suma 1 a j porque las matrices empiezan en 1 y no en cero. Por lo
tanto
              //c=j+1
                *v=-1; *c=j+1;
              }
   }//Fin flag==0
  }//Fin bucle j

 return 0;
 }//Fin de la funciÃ³n

 int RPN(int rank, int size, int N,int *I,int *F)
{
//*************** FUNCIONES PROPIAS DE ESTE PROGRAMA *********************
/*********************************************************************/
//
//Esta funciÃ³n divide intervalos segun rank size y N y dev. I, F
/*********************************************************************/

if (rank !=0) {*I=rank*int(floor(N/size))+1;}
else {*I=0;}
if (rank !=(size-1)){*F=(rank+1)*int(floor(N/size));}
else {*F=N-1 ;}
return 1;

} //FIN
```