UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

COLEGIO DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN MODELACIÓN MATEMÁTICA

Métodos de Diferencias Finitas y Elemento finito: "Análisis comparativo con programación Paralela"

T E S I S

QUE PARA OPTAR POR EL TÍTULO DE

LICENCIADO EN MODELACIÓN MATEMÁTICA

P R E S E N T A N :

JAVIER PÉREZ RAMÍREZ
CÉSAR EDUARDO ROMERO BAUTISTA

D I R E C T O R

MTRO. ENRIQUE CRUZ MARTÍNEZ

Ciudad de México, junio de 2021.

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

CA Coordinación Académica

SISTEMA BIBLIOTECARIO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO
COORDINACIÓN ACADÉMICA

RESTRICCIONES DE USO PARA LAS TESIS DIGITALES

DERECHOS RESERVADOS©

Índice general

Índice de figuras

Índice de cuadros

Introducción

¿Qué pasaría si se pudiera encontrar la manera de predecir qué es lo que va a suceder en un futuro con respecto a una problemática basada en fenómenos físicos? Dar respuesta a esta cuestión es en lo que está basado este trabajo; uno de los objetivos es desarrollar el panorama de las personas interesadas en el tema y ampliar el material de apoyo.

En este libro se detallarán los algoritmos para la aplicación de los métodos enunciados en el título, así como un contexto de los mismos para introducir al lector de manera amigable con el tema. Se detallará la teoría general de los métodos y se aplicarán a ejercicios en particular para ejemplificar con mayor detalle el uso de las técnicas.

El trabajo será abordado desde una perspectiva matemática, con un enfoque fundamentado en el lenguaje de programación numérica "Fortran".

Agradecimientos

En estas líneas queremos agradecer a nuestros padres, por siempre apoyarnos, motivarnos y por sus sabios consejos, son la base para concluir nuestras metas. A nuestros hermanos por animarnos cada día para terminar este trabajo. A nuestros familiares que creyeron en nosotros, amigos que estuvieron al pendiente y siempre dándonos ánimos, a cada uno de los profesores por apoyarnos a lo largo de nuestra carrera, a nuestros lectores por darse el tiempo de ayudarnos y aportar su conocimiento para que el trabajo fuera de excelencia, a nuestro director por siempre motivarnos, ayudárnos y nunca dejarnos solos, por ser nuestro guía para culminar nuestra carrera.

Le agradecemos al Laboratorio de Cómputo para la Enseñanza de las Ciencias (LACECI) por abrirnos las puertas. Al maestro Fernando Robles Morales del Laboratorio Nacional de Supercómputo (LNS) por dar acceso a una cuenta en el clúster de la Benemérita Universidad Autónoma de Puebla (BUAP).

Agradecemos a la Universidad Autónoma de la Ciudad de México por el apoyo otorgado para el empastado e impresión de este trabajo.

Objetivo

El objetivo de esta tesis es explicar las técnicas numéricas más usadas para resolver ecuaciones diferencias parciales (EDP) utilizando las técnicas numéricas usuales como Diferencias Finitas (DF) y Elemento Finito (EF) así como ilustrar su implementación para resolverse usando procesamiento paralelo con **OpenMP** principalmente.

Prólogo

A lo largo de nuestra estadía en la universidad siempre nos preguntaban: "¿Qué es modelación matemática?", nosotros al principio no sabíamos que significaba, no teníamos una idea clara de lo que trataría nuestra carrera, para qué nos serviría, en que nos desarrollaríamos. No sabíamos nada, pero en cada curso comprendíamos más el concepto que venía planteando nuestra carrera, poco a poco encontrábamos el camino que la universidad nos brindaba, y ahora con seguridad podemos responder esta simple, pero a la vez compleja pregunta: La modelación matemática es la aplicación de las matemáticas en los problemas del mundo real e investigar preguntas importantes que surgen. Usando las herramientas matemáticas correctas, el problema del mundo real se transforma en una cuestión matemática, que imita al original. Se obtiene una solución, la cual se interpreta en el lenguaje del problema inicial para hacer predicciones de este. Por problemas del mundo real, nos referimos a dificultades en el campo de la biología, química, ingeniería, ecología, medio ambiente, física, ciencias sociales, estadísticas, manejo de vida silvestre, etc. La modelación matemática se describe como una tarea que permite a un matemático colaborar con biólogos, químicos, economistas, etcétera, dependiendo del problema en el que se esté trabajando. El objetivo principal de un modelador es realizar experimentos sobre la representación matemática de un problema del mundo real, en lugar de realizar experimentos en él. El verdadero desafío es el modelado matemático. Se debe evitar producir el modelo descriptivo más completo, para producir el modelo más simple posible que incorpore las características principales del fenómeno de interés.

Si un modelo matemático puede imitar el comportamiento de una situación de la vida real, obtenemos una comprensión mejor del sistema a través del análisis adecuado del modelo utilizando herramientas matemáti-

cas apropiadas. Además, en el proceso de construcción de cualquier modelo, encontramos factores que sobresalen del sistema, factores que son más importantes para el sistema y que revelan el comportamiento del mismo. La importancia del modelado matemático en física, química, biología, economía e incluso en la industria no puede ser ignorada. El modelado matemático en ciencias básicas está ganando popularidad, principalmente en ciencias biológicas, economía y problemas industriales. Por ejemplo, si consideramos el modelado matemático en la industria del acero, muchos aspectos en su fabricación, desde la minería hasta su distribución, son susceptibles al modelado matemático. Cabe mencionar que distintos tipos de empresas se han incorporado en aplicaciones de la matemática, descubriendo que pueden solucionar varios problemas apoyándose en modelos matemáticos: problemas que implican el control de la refrigeración de lingotes, transferencia de calor y masa en altos hornos, mecánica de laminado en caliente, soldadura por fricción, enfriamiento por pulverización y contracción en la solidificación de lingotes, por mencionar algunos. Para obtener información física, se utilizan técnicas analíticas. Sin embargo, para tratar problemas más complejos, los enfoques numéricos son bastante útiles. Siempre es aconsejable y útil formular un sistema complejo con un modelo simple cuya ecuación dé una solución analítica. Luego, el modelo se puede modificar a uno más realista que se pueda resolver numéricamente. Junto con los resultados analíticos para modelos más simples y la solución numérica de modelos más realistas, se puede obtener una visión máxima del problema.

Capítulo 1

Ecuaciones Diferenciales

1.1. Introducción a las Ecuaciones Diferenciales

Las ecuaciones diferenciales son herramientas matemáticas que nos ayudan a describir el comportamiento de fenómenos tanto naturales como físicos. Dependiendo de la complejidad de la ecuación diferencial esta puede incluso llegar a describir procesos muy complejos.

Nos ayudan a "predecir el futuro", al resolver problemas que puedan ser modelados. Utilizando las reglas que conocemos del comportamiento que gobierna a un proceso, y también sabiendo que los cambios con respecto al tiempo pueden ser medidos con la derivada; usar estos conocimientos en conjunto para describir cómo se va a comportar el proceso es de lo que tratan las ecuaciones diferenciales.

El uso de las ecuaciones diferenciales tiene una gran cantidad de aplicaciones, como modelar fenómenos un tanto típicos como la propagación del calor o del sonido, mecánica de fluidos, electromagnetismo, etcétera; hasta expresiones de mecánica cuántica o matemática moderna.

Podemos dividir a las ecuaciones diferenciales en dos grandes grupos:

Ordinarias: Este tipo de ecuaciones contienen derivadas de una o más variables dependientes con respecto de una variable independiente.

Parciales: Generalmente abreviadas (EDP) o en inglés (DPE). Este tipo de ecuaciones involucran derivadas parciales de una o más variables dependientes de una o más variables independientes.

En matemáticas una ecuación en derivadas parciales es aquella ecuación diferencial cuyas incógnitas son funciones de diversas variables independientes, con la peculiaridad de que en dicha ecuación figuran no solo las propias funciones sino también sus derivadas. Tienen que existir funciones de por lo menos dos variables independientes. O bien una ecuación que involucre una función matemática de varias variables independientes y las derivadas parciales de respecto de esas variables. Las ecuaciones en derivadas parciales se emplean en la formulación matemática de procesos de la física y otras ciencias que suelen estar distribuidos en el espacio y el tiempo. Problemas típicos son la propagación del sonido o del calor, la electrostática, la electrodinámica, la dinámica de fluidos, la elasticidad, la mecánica cuántica y muchos otros. Se les conoce también como ecuaciones diferenciales parciales.

1.2. Historia

Las expresiones matemáticas de las que hablamos comienzan con el nacimiento del cálculo de Issac Newton y Gottfried Wilhelm Leibniz, quien en 1684, publicó su trabajo en la revista "*Acta Eruditorum*". Contenía una definición de la diferencial, y donde aparecieron pequeñas reglas para el cálculo en sumas, productos, cocientes, potencias y raíces. Leibinz añadió pequeñas aplicaciones a problemas de tangentes y puntos críticos.

La primera vez que el término "ecuación diferencial" (aequatio differentialis) se utilizó, fue en 1676 por Leibniz para expresar la relación entre las diferenciales dx y dy, en dos variables x y y. Es importante mencionar que las "Ecuaciones Diferenciales Ordinarias" nacen con la aparición del Cálculo en la famosa nota de Newton a Leibniz, donde Newton manda el siguiente anagrama:

6a cc d ae 13e ff 7i el 9n 4o 4q rr 4s 9t 12v x,

La primera clasificación de las Ecuaciones Diferenciales Ordinarias de primer orden fue dada por Newton. Las de primer orden estaban compuestas por aquellas ecuaciones en las cuales dos **fluxiones** *x'*, *y'*, y un **fluentes** x o y están relacionados, por ejemplo:

$$\frac{x'}{y'} = f(x)$$

o bien

$$\frac{dx}{dy} = f(x)$$

$$\frac{dy}{dx} = f(y)$$

el segundo orden abarcó aquellas ecuaciones que involucran dos **fluxiones** y dos **fluentes**

$$\frac{x'}{y'} = f(x, y)$$

y

$$\frac{dy}{dx} = f(x, y).$$

Finalmente, el tercer orden abarcó a ecuaciones que contienen más de dos **fluxiones**, las cuales en la actualidad nos llevan a ecuaciones diferenciales parciales.

Es conveniente resaltar dos de las características más importantes del momento histórico de Newton-Leibniz:

1. En esta época los problemas seguían resolviéndose con geometría euclidiana. Tanto Leibniz como Newton, elaboraron sus conceptos matemáticos en términos geométricos en los que se representan las propiedades y conceptos. "Esto es una consecuencia de lo restringido que se encontraba el concepto de función en el siglo XVII" [8]. La percepción de función iba de la mano con la idea de curva geométrica. En este sentido, obviamente el concepto de tangente era el euclidiano. La noción que se manejaba de recta tangente era puramente intuitiva.

2. El cálculo de Newton y de Leibniz trataba de variables. En el cálculo de Leibniz las variables eran una sucesión de valores infinitamente próximos; en el de Newton, las cantidades variaban con el tiempo. El primero ve el

espacio geométrico formado por segmentos infinitesimales. El segundo tiene una idea intuitiva de movimiento continuo cercana al concepto de límite.

Por otro lado los hermanos James y Johan Bernoulli introdujeron términos como el de "integrar" una ecuación diferencial y el proceso "separación de variables" de una ecuación diferencial en la última década del siglo XVII.

Johan Bernoulli encontró un método diferente; alrededor de 1692, lo utilizó en una serie de problemas, "multiplicación por un factor integrante". Para resolver ecuaciones en los cuales el método de separación de variables no se podía aplicar, por ejemplo:

$$xdy - ydx = 0$$

recordemos que en esta época no se conocía que

$$\frac{dx}{x} = lnx.$$

Sin embargo, los métodos eran incompletos y la teoría general de las ecuaciones diferenciales (a comienzos del siglo XVII) no podía ser propuesta.

Hablaremos de algunas aportaciones a lo largo de la historia de las ecuaciones diferenciales.

El matemático italiano J. F. Riccati (1676-1754), en 1724 analizó la ecuación

$$\frac{dy}{dx} + ay^2 = bx$$

determinando la integrabilidad en funciones derivadas de ésta y se extiende a todas las ecuaciones del tipo:

$$y' = P(x)y2 + Q(x)y + R(x)$$

donde P, Q y R son funciones continuas. La investigación de esta ecuación la llevaron a cabo muchos matemáticos: Leibniz, Ch. Goldbach, Johan I, Nicolás I y Daniel Bernoulli entre otros.

Euler hizo la primera sistematización de los trabajos para resolver las ecuaciones diferenciales y puede decirse que es la primera teoría de las ecuaciones diferenciales ordinarias (1768-1770). Donde podemos encontrar métodos para resolver ecuaciones diferenciales de primer orden, segundo orden y

de orden superior, así como el método de series de potencias. Euler tenia una forma de ver las ecuaciones diferenciales ordinarias distintas a las de hoy en día, la expresión:

$$\frac{dy}{dx}$$

significaba para él un cociente entre diferenciales y no la derivada como la conocemos, otro ejemplo en una ecuación de segundo orden aparece en los diferenciales

$$ddy,\ dx^2$$

en lugar de la segunda derivada y".

Muchos matemáticos, entre ellos Clairaut y Euler (los más importantes) siguieron desarrollando el método de factor integrante. Entre 1768 y 1769, Euler analizó las clases de ecuaciones diferenciales que tienen factor integrante de un tipo y quiso extender esta investigación a ecuaciones de orden superior. Lagrange (1736-1813), demostró que la solución general de una ecuación diferencial lineal homogénea de orden n con coeficientes constantes es:

$$"y = c_1y_1 + c_2y_2 + ... + c_ny_n$$

donde

$$y_1, y_2, ..., y_n$$

son un conjunto de soluciones linealmente independientes y

$$c_1, c_2, ..., c_n$$

son constantes arbitrarias" [8], desarrolló en su forma general el método de variación de parámetros en 1774.

La Ecuación de Riccati hizo un cambio a la tradición algebraica: una ecuación relativamente sencilla que en la mayoría de los casos no puede integrarse en cuadraturas. En segundo lugar, este rompimiento es más fuerte si puntualizamos que una de las razones por las cuales es más fácil resolver una ecuación diferencial lineal que una no lineal (aparte de la propia naturaleza de esta última que puede impedir tal propósito) es la existencia del Principio de Superposición ya mencionado. Este principio, es la forma usual de expresar la solución general como una función de un número finito de soluciones particulares. La Ecuación de Riccati es una ecuación no lineal que posee una solución general que satisface la fórmula:

$$\frac{((y_1 - y_2)(y_2 - y_4))}{((y_1 - y_4)(y_2 - y_3))} = \frac{((\propto_1 - \propto_2)(\propto_2 - \propto_4))}{((\propto_1 - \propto_4)(\propto_2 - \propto_3))} \tag{1.1}$$

para cuatro valores diferentes de " $\propto$ ", y cualesquiera tres soluciones particulares

$$y_1, y_2, ..., y_3,$$

o sea, una solución general de una ecuación no lineal que se expresa en términos de otras particulares [8].

De esta manera, el trabajo era dar la solución de ecuaciones específicas. Así entonces fueron creados los postulados para la creación de las bases para la teoría general, con varios conceptos fundamentales.

En el año 1743, surgieron los conceptos de integral particular y general, encontradas por Euler previamente en 1739. Ellos fueron publicados en la memoria donde se trata de un único algoritmo de resolución de ecuaciones lineales de orden n con coeficientes constantes.

En este contexto, veamos el protocolo de rigor que imperaba a finales del siglo XVIII:

- Cada concepto matemático debía ser explícitamente definido en términos de otros conceptos cuya naturaleza era suficientemente conocida.
- Las pruebas de los teoremas debían ser completamente justificadas en cada una de sus etapas, o bien por un teorema anteriormente probado, por una definición o por un axioma explícitamente establecido.
- Las definiciones y axiomas escogidos debían ser lo suficientemente amplios para que pudiesen cubrir los resultados ya existentes.
- La intuición (geométrica o física) no era un criterio válido para desarrollar una prueba matemática.

Las dos primeras caracterizaciones han permanecido más o menos estables desde la época de Euclides. Las otras dos, son un pronunciamiento en contra de concepciones matemáticas muy comunes hasta el siglo XVIII. En

esta época, casi todos los problemas del cálculo, surgían de la necesidad de mate-matizar algún fenómeno de índole físico. Dado a esto los resultados matemáticos cobraban una importancia sobresaliente.

Por otra parte, este cálculo encontraba cada día nuevas aplicaciones en la mecánica y la astronomía, convirtiéndose, poco a poco, en la parte central y más productiva del conocimiento matemático.

Una teoría que todavía se encuentra en el estado de búsqueda de leyes fundamentales de desarrollo y de definición precisa de sus conceptos principales y no puede ser fundamentada lógicamente. Además, para poder hablar de una fundamentación filosófica, los conceptos fundamentales deben ser suficientemente generales y a la vez bien determinados. La fundamentación lógica y filosófica del cálculo diferencial e integral era objetivamente imposible sobre la base de los conceptos sobre los cuales aparecieron y por eso los esfuerzos de Newton, Leibniz, Lagrange y otros, que hasta los comienzos del siglo XIX, terminaron en el fracaso. Señalemos las principales insuficiencias:

Incorrecta comprensión del concepto de diferencial: En Leibniz, L'Hopital, Euler y otros matemáticos del siglo XVIII el concepto de diferencial se confundía. Una aproximación suficientemente correcta del concepto de diferencial fue dada sólo por Lagrange (1765).

Insuficiente comprensión del concepto de función: De hecho hasta fines del siglo XIX los matemáticos partiendo de la intuición mecánica y geométrica, entendieron por fundamentación sólo las funciones analíticas representadas por una determinada fórmula (en algunos casos infinita como es el caso de las consideraciones de Fourier ligadas con su teoría del calor). Sólo con la aparición de las funciones discontinuas en problemas prácticos, los matemáticos prestaron atención a la formación lógica del concepto de función.

Ausencia de un concepto claro de límite: Los seguidores de Newton: Maclaurin, Taylor, Wallis y otros, mantuvieron una larga discusión sobre el hecho de que si la variable alcanza o no el límite. Este problema no era fácil, precisamente, porque no había una definición precisa de límite y sólo se determinaba por razonamientos mecánicos y geométricos. Esta insuficiencia permaneció hasta Cauchy (1823).

El concepto de continuidad funcional era intuitivo: Esto se explica porque

los matemáticos del siglo XVIII consideraban todas las funciones continuas y por eso no tenían la necesidad de precisar este concepto. Sólo a principios del siglo XIX se comenzó a pensar en este problema (otros detalles los puede encontrar en la última sección de esta conferencia).

Concepto difuso de integral definida: Relacionado ante todo con la ausencia de un teorema de existencia. Se consideraba por ejemplo, que la fórmula de Newton-Leibniz tenía un significado universal, es decir, que era válida para todas las funciones y en todas las condiciones. Los esfuerzos en la precisión del concepto hechos por Lacroix, Poisson y Cauchy pusieron en primer plano el concepto de límite y de continuidad. Pero el problema de la integral definida sólo halló una respuesta completa hasta fines del siglo XIX en los trabajos de Lebesgue.

Se necesitaba tener una clara comprensión de lo que era un sistema numérico: En particular, la estructura del sistema de los números reales, lo que no sucederá sino con las investigaciones de Dedekind y Cantor, entre otros; otra de las concepciones básicas relacionadas con este tópico, era el concepto mismo de número (aquí, nuevamente debemos mencionar a los matemáticos del siglo XIX y a Frege en especial, para seguir con Russel, y otros).

Así, el movimiento del análisis matemático en el siglo XVIII hacia su fundamentación, puede describirse completamente en el sistema "teoría-práctica", esto es, como interrelación dialéctica entre estos momentos. La necesidad del cálculo de áreas y volúmenes y del hallazgo de máximos y mínimos entre otros problemas concretos, conllevó a la creación del algoritmo del cálculo diferencial e integral. La aplicación de estos algoritmos a nuevos problemas inevitablemente conllevó a la generalización y precisión de los algoritmos. En última instancia, el análisis se formalizó como lógicamente no-contradictorio, como un sistema relativamente cerrado y completo.

1.3. Modelos

Un modelo básicamente es una expresión matemática que representa o describe un fenómeno y como se relaciona con ciertas variables. La acción de modelar a grandes rasgos es convertir las reglas inherentes de la evolución de un suceso en una ecuación diferencial.

En la ciencia y en la ingeniería se desarrollan modelos matemáticos para comprender mejor eventos físicos o naturales. Con frecuencia estos modelos producen una ecuación que contienen algunas derivadas de una función incógnita, a esto se le conoce como ecuación diferencial.

1.3.1. Deducción de un modelo

Para ampliar un poco más el conocimiento acerca de los modelos, en esta sección vamos a deducir un sistema físico sencillo y llegar a una ecuación matemática que describa su comportamiento conforme pasa el tiempo.

Considérese un bloque de masa "m" sujeto a un extremo de un resorte con constante de elasticidad "k", mientras que el otro extremo está fijo, no existe fricción sobre la superficie (fig 1.1).

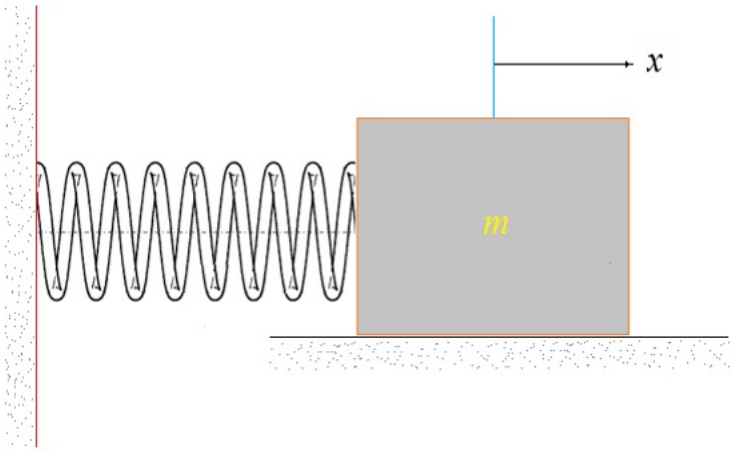

Figura 1.1: Sistema masa-resorte.

Este sistema se mueve en una sola dirección, supongamos el eje "x", cuando el sistema está en equilibrio, el resorte no está ni comprimido ni estirado y por lo tanto la masa está en reposo.

Al estirar o comprimir el resorte este comenzara a moverse con (movimiento armónico simple) y como se mencionó antes no hay ninguna fuerza externa que se relaciones con el sistema, por lo cual el movimiento será perpetuo una vez que se altere la posición de reposo.

Usando la segunda ley de Newton para describir las fuerzas que gobiernan en el sistema, la ecuación se ve de la siguiente manera:

$$m\frac{d^2x}{dt^2} = -kx$$

Para simplificar un poco la ecuación podemos dividir entre la masa "m" y después dejando el lado derecho igualado a "0" la ecuación diferencial se ve la siguiente manera:

$$\frac{d^2x}{dt^2} + \frac{kx}{m} = 0 \tag{1.2}$$

Si hacemos

$$\omega_0^2 = k/m$$

se obtiene una ecuación homogénea y se va a resolver de manera analítica, usando el polinomio auxiliar y proponiendo la solución:

$$x = e^{mt}$$

Derivamos la solución dos veces para encontrar una expresión:

$$\dot{x} = me^{mt}$$

y

$$\ddot{x} = m^2e^{mt}$$

sustituyendo en la ecuación diferencial (ec. 1.2):

$$m^2e^{mt} + \omega_0^2e^{mt}$$

factorizando

$$(m^2 + w_0^2)e^{mt} = 0$$

dividiendo entre e^{mt}, el polinomio auxiliar queda de la siguiente manera:

$$m^2 + \omega_0^2 = 0 \tag{1.3}$$

El siguiente paso es encontrar las raíces del polinomio (ec. 1.3) despejando m:

$$m = \sqrt{-\omega_0^2}$$

$$m = \pm i\omega_0$$

Las soluciones son:

$$x_1(t) = e^{i\omega_0} + e^{-i\omega_0}$$

y

$$x_2(t) = e^{i\omega_0} - e^{-i\omega_0}$$

Usando la fórmula de Euler podemos ver las soluciones de la siguiente manera:

$$x_1(t) = \frac{e^{i\omega_0} + e^{-i\omega_0}}{2} = cos(\omega_0 t)$$

$$x_2(t) = \frac{e^{i\omega_0} + e^{-i\omega_0}}{2i} = sen(\omega_0 t)$$

La solución (ec. 1.4) a la ecuación diferencial homogénea será la suma de las soluciones anteriores:

$$x(t) = Acos(\omega_0 t) + Bsen(\omega_0 t) \tag{1.4}$$

donde A y B son constantes que van a quedar determinadas por problemas con condiciones iniciales.

1.4. Importancia de las Ecuaciones Diferenciales Parciales en la actualidad

Ya dimos una pequeña introducción de lo que son las ecuaciones diferenciales parciales, en esta sección vamos a dar énfasis a la relevancia de dichas ecuaciones en la actualidad. Las EDP son tan esenciales en la vida moderna que no hay rama de las ciencias que no las utilicen, por mencionar algunos ejemplos:

- Ecuación de Dirac:

$$(\alpha_0 mc^2 + \sum_{j=1}^{3} \alpha_j p_j c)\Psi(x,t) = i\hbar \frac{\partial \Psi}{\partial t}(x,t).$$

- Ecuación de Schrödinger:

$$i\hbar \frac{\partial}{\partial t}\Psi(r,t) = \hat{H}\Psi(r,t).$$

- Ecuación de calor:

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0.$$

- Ecuación de Laplace:

$$\Delta \varphi = 0.$$

- Ecuación de onda:

$$\frac{\partial^2 \xi}{\partial t^2} = v^2 \frac{\partial^2 \xi}{\partial x^2}.$$

Estas ecuaciones tienen aplicaciones en una gran cantidad de áreas de las ciencias porque modelan muchos fenómenos, tanto biológicos, químicos, físicos, como de ingeniería de economía, etcétera.

1.5. Solución de la ecuación de calor en 1-D

La ecuación de calor (ec. 1.5) en una dimensión es un caso sencillo de la solución analítica de una ecuación diferencial. A continuación presentamos el problema con condiciones de frontera en la ecuación de calor y su solución exacta.

Consideremos una varilla de longitud L (fig. 1.2), la cual en sus extremos la temperatura va a permanecer constante (0). La incógnita en esta ecuación es la función u y esta va a representar la temperatura en cada instante de tiempo. La ecuación se ve de la siguiente manera:

$$u_t = \beta^2 u_{xx}$$

$$\text{x}\in (0, L), \quad t > 0, \quad u(x, 0) = f(x), \quad u(0, t) = u(L, t) = 0 \qquad (1.5)$$

Figura 1.2: Varilla de longitud L.

Resolveremos la ecuación por el método de variables separables, para ello propongamos la solución:

$$u(x, t) = X(x)T(t).$$

Calculamos las derivadas parciales para sustituir en la ecuación, para ello necesitamos la parcial de u con respecto de t y la segunda derivada parcial de u con respecto de x:

$$u_t = X(x)T'(t)$$

y

$$u_{xx} = X''(x)T(t)$$

De esta forma para que $u(x,t)$ sea solución tiene satisfacer la igualdad:

$$X(x)T'(t) = \beta^2 X''(x)T(t).$$

Separando las variables la expresión se ve de la siguiente manera:

$$\frac{T'(t)}{\beta^2 T(t)} = \frac{X''(x)}{X(x)}$$

Como x y t son variables independientes y cada lado solo depende de una sola de las variables, deben ser iguales a una constante, adoptémosla como , es decir:

$$X''(x) = -(x)$$

y

$$T'(t) = -(t)$$

Para encontrar las soluciones de las nuevas ecuaciones recurriremos al polinomio característico de cada una:

$$m^2 + \lambda = 0.$$

Entonces se tiene que

$$m = +\sqrt{-\lambda}$$

descartando las soluciones triviales, las soluciones son de la forma:

$$X(x) = a\cos\sqrt{\lambda}x + b\,\mathrm{sen}\,\sqrt{\lambda}x$$

Aplicando la condición de que $X(0) = 0$, implica que $a = 0$ y X(L)=0 implica que:

$$b\,\mathrm{sen}\,\sqrt{\lambda}L = 0$$

sin embargo, esta última igualdad es cierta si

$$\sqrt{\lambda} = n$$

con $n \in N$, si tomamos $\lambda = n^2$ entonces:

$$X_n(x) = b_n\,\mathrm{sen}\,nx$$

con $n \in N$.

Utilizando el método de separación de variables para hallar la solución de la ecuación

$$T'(t) = -(t),$$

obtenemos que:

$$T(t) = ce^{-\lambda t}.$$

Como anteriormente definimos $\lambda = n^2$ la solución se ve como:

$$T_n(t) = c_n e^{-n^2 t}.$$

Combinando las soluciones se obtiene:

$$b_n e^{-n^2 t}\,\mathrm{sen}\,(nx)$$

con $n \in N$.

Por el teorema de superposición $\forall\ k \in N$ la solución es de la forma:

$$u(x,t) = \sum_{n=1}^{\infty} [b_n e^{-n^2 t} \operatorname{sen}(nx)].$$

[18].

1.6. Solución de la ecuación de onda en 1-D

Ahora resolveremos la ecuación de onda (ec. 1.6) en una dimensión de la misma manera que resolvimos la ecuación de calor, usando el método separación de variables. La ecuación de onda es un problema clásico para ejemplificar la solución analítica de una ecuación.

$$U_{tt} = c^2 U_{xx}$$

Con las condiciones de frontera: U(0,t) = 0 = U(L,t) (1.6)

Asumiremos que $U(x,t) = X(x)T(t)$ ahora sustituyendo en la ecuación 1.6:

$$X(x)T''(t) = c^2 X''(x)T(t)$$

dividimos $X(x)T(t)$ en ambos lados:

$$\frac{T''}{T} = c^2 \frac{X''}{X} = -\omega^2 \ (constante)$$

Despejando

$$X'' = -k^2 X$$

con

$$k = \frac{\omega}{c}$$

donde:

$$X = A \operatorname{sen} kx + B \operatorname{sen} kx$$

Con las condiciones de frontera:

$$X(0) = B = 0$$

y

$$X(L) = A \operatorname{sen} kL = 0$$

$\Rightarrow$ k=$\mathrm{k}_n = n\frac{\pi}{L},\ con\ n = 0, 1, ...$

Así la solución de $X(x)$ es:

$$X(x) = \sum_n a_n \operatorname{sen} \frac{n\pi x}{L}$$

De manera similar si despejamos $T'' = -\omega_n^2 T$ donde $\omega_n = ck_n$

encontrando la solución general:

$$T(t) = c \operatorname{sen} \omega_n t + D \cos \omega_n t$$

Entonces la solución de la ecuación de onda es efectivamente:

$$U(x,t) = X(x)T(t)$$

$$= \sum_n [a_n \operatorname{sen} \omega_n t + b_n \cos \omega_n t] \operatorname{sen} k_n x$$

Donde a_n y b_n están dadas por las condiciones iniciales:

$$U(x,0) = U_0(x)$$
$$\frac{\partial U}{\partial t}(x,0) = V_0(x)$$

$$\Rightarrow U_0 = \sum_n b_n \operatorname{sen} k_n x$$

$$\Rightarrow V_0 = \sum_n a_n \omega_n \operatorname{sen} k_n x$$

Usando la ortogonalidad de la función seno:

$$\int_0^L \operatorname{sen} k_n x \operatorname{sen} k_m x dx = \delta_{nm}$$

$$\Rightarrow b_m = \frac{2}{L} \int_0^L U_0(x) \operatorname{sen} k_m x dx$$

$$\Rightarrow a_m = \frac{2}{\omega_m L} \int_0^L V_0(x) \operatorname{sen} k_m x dx$$

Donde $k_n = \frac{n\pi}{L}$ y $k_m = \frac{m\pi}{L}$ [4].

Capítulo 2

Métodos Numéricos

2.1. Métodos de resolución de ecuaciones diferenciales

En el siglo XIX Augustin Louis Cauchy (1789-1857) se preocupó por dar solución a las ecuaciones diferenciales. Demostró por primera vez la solubilidad de un problema con condición inicial (hoy en día se conocen como condiciones de Cauchy), pero fue hasta 1868 que Rudolf Lipschitz (1832-1903) demostró la existencia y unicidad bajo la exigencia de continuidad y la condición de Lipschitz, a este resultado se le conoce como "Teorema de Cauchy-Lipschitz".

Existen tres técnicas básicas para resolver las ecuaciones diferenciales.

Las técnicas analíticas: Estas implican encontrar fórmulas para los valores futuros de la cantidad, es decir, encontrar una solución exacta a la ecuación (si es que esta existe).

Los métodos cualitativos: Estos se apoyan en un esbozo de la gráfica de la ecuación como función del tiempo, y en la descripción de su comportamiento a largo plazo.

Las técnicas numéricas: Requieren que efectuemos cálculos aritméticos

que den aproximaciones de los valores futuros de la cantidad, los cálculos pueden llegar a ser tantos y tan complejos que es necesario recurrir a un ordenador para poder llegar a encontrar una solución aproximada.

El estudio de los fenómenos ya sean naturales o de distinta índole resulta interesante, ya que en muchos casos encontrar una solución analítica a dicho problema es una tarea muy laboriosa y en algunos casos hasta imposible; es por ello que surge la idea de estudiar la solución de una ecuación diferencial desde otra perspectiva la cual da lugar a la Teoría Cualitativa de Ecuaciones Diferenciales.

2.1.1. Método de diferencias finitas

Las diferencias finitas son expresiones matemáticas que nos permiten "aproximar" las derivadas de las funciones evaluadas en diferentes tiempos. La técnica de diferencias finitas se emplea a menudo en análisis numérico, particularmente en ecuaciones diferenciales, ecuaciones diferenciales ordinarias numéricas y ecuaciones en derivadas parciales. Los métodos resultantes reciben el nombre de métodos de diferencias finitas.

Las aplicaciones más comunes de los métodos de diferencias finitas son en los campos de la computación y áreas de la ingeniería como ingeniería térmica o mecánica de fluidos y en general en las ramas donde se utilicen las matemáticas.

La técnica para aplicar el método de diferencias finitas es aproximar las derivadas con una expresión, describiremos esas aproximaciones a continuación:

Nos interesa conocer la derivada de la función f

Diferencias finitas hacia adelante:

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \varepsilon(h) \tag{2.1}$$

Diferencias finitas hacia atrás:

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \varepsilon(h) \tag{2.2}$$

Diferencias finitas centradas:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \varepsilon(h^2) \tag{2.3}$$

donde h es el tamaño del espaciado entre un tiempo y otro, mientras que $\varepsilon(h)$ y $\varepsilon(h^2)$ representan el error de la aproximación de las derivadas conocidos como error de primer orden y error de segundo orden respectivamente.

Para derivadas de orden mayor, por ejemplo, tomemos el espaciado de h^2 y usemos las diferencias centrales (ec. 2.3) para $f'(x+h)$ y $f'(x-h)$ y

aplicando la definición a la primera derivada de f' en x, se obtiene la segunda derivada de f:

$$f''(x) = \frac{f(x+h) - f(x-h) - 2f(x)}{h^2} + \varepsilon(h^2)$$

2.1.2. Elemento finito

El método de elemento finito consiste a grandes rasgos en la discretización del dominio en el que se está trabajando, es decir, dividir el dominio de la ecuación diferencial en subdominios "más sencillos" que sean más fáciles de estudiar.

Para ejemplificar este método vamos a aproximar el perímetro de una circunferencia mediante segmentos de líneas de los cuáles conocemos su longitud exacta. Entre más pequeños hagamos los segmentos de línea, más exacta va a ser la aproximación de la circunferencia.

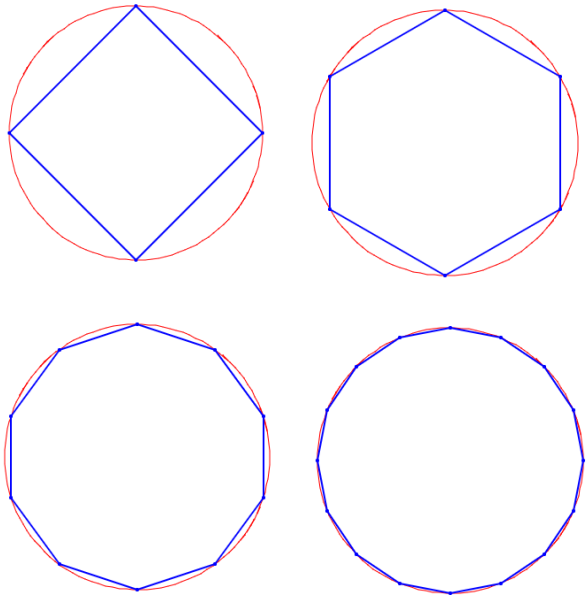

Figura 2.1: Aproximación de una circunferencia

En el panel (fig. 2.1) se aproxima la circunferencia con segmentos de línea grandes que se van haciendo pequeños en cada figura:

Este ejemplo sirve como idea intuitiva para introducir el concepto de discretización de un dominio; el dominio en este caso es la circunferencia, mientras que los segmentos de línea son los "elementos" o subdominios y las conexiones que unen los elementos los llamaremos "nodos".

A los elementos de la discretización les llamaremos "malla". Dependiendo la manera de discretizar el dominio, por ejemplo, en la circunferencia; si los fragmentos de línea son todos de la misma longitud diremos que es una malla uniforme, de lo contrario será una "malla" no uniforme.

2.2. Ecuación de Poisson resuelta con diferencias finitas

En esta sección ejemplificaremos la manera de resolver una ecuación diferencial utilizando el método de diferencias finitas, específicamente la ecuación de Poisson (ec. 2.4).

$$U_{xx} + U_{yy} = f \tag{2.4}$$

Discretizamos en las direcciones de x y y

Cuando discretizamos en la dirección x con el espacio en la cuadrícula

$$\triangle x = \frac{a}{(m+1)}$$

y en la dirección y con el tamaño de paso

$$\triangle y = \frac{b}{(n+1)}$$

y después

$$x_k = k \triangle x,$$

donde

$$0 \leq k \leq m+1$$

y

$$y_j = j \triangle y,$$

donde

$$0 \leq j \leq n+1.$$

Dejamos

$$U_{kj} = U(x_k, y_j)$$

y

$$f_{kj} = f(x_k, y_j)$$

ahora resolviendo la ecuación de Poisson con las siguientes condiciones iniciales:

$$U(0, y) = U_{0,j} = g_{0,j}(y_j)$$

$$U(a, y) = U_{m+1,j} = g_{m+1,j}(y_j)$$

$$U(x, 0) = U_{k,0} = g_{k,0}y(x_k)$$

$$U(x, b) = U_{k,n+1} = g_{k,n+1}y(x_k)$$

Ahora usando diferencias centradas (ec. 2.3) aproximaremos U_{xx} y U_{yy}, entonces las diferencias finitas para la aproximación de la ecuación de Poisson son:

$$\frac{\partial^2 U}{\partial x^2} = U_{xx}(x_k, y_j) = \frac{U_{k+1,j} - 2U_{k,j} + U_{k-1,j}}{\triangle x^2}$$

$$\frac{\partial^2 U}{\partial y^2} = U_{yy}(x_k, y_j) = \frac{U_{k,j+1} - 2U_{k,j} + U_{k,j-1}}{\triangle y^2}$$

Nuestra discretización de la ecuación de Poisson se convierte en (si $\triangle x = \triangle y = h$):

$$\frac{U_{k+1,j} - 2U_{k,j} + U_{k-1,j}}{h^2} + \frac{U_{k,j+1} - 2U_{k,j} + U_{k,j-1}}{h^2} = f_{k,j}$$

ó

$$U_{k+1,j} - 4U_{k,j} + U_{k-1,j} + U_{k,j+1} + U_{k,j-1} = h^2 f_{k,j}$$

Dado que tenemos condiciones de frontera, los límites exteriores de la región que estamos resolviendo son:

$$U_{0,j}$$

$$U_{m+1,j}$$

$$U_{k,0}$$

$$U_{k,n+1}$$

y necesitaremos los valores interiores: $U_{k,j}$ para $1 \leq k \leq m$ y $1 \leq j \leq n$.

Por ejemplo $m = 3$ y $n = 3$:

Resolveremos los valores interiores (fig. 2.2), ya que los valores de las fronteras ya están definidos. Dejamos que el vector de valores interiores que estamos resolviendo se defina como:

$$\vec{U} = \begin{pmatrix} U_{11} \\ U_{12} \\ U_{13} \\ U_{21} \\ U_{22} \\ U_{23} \\ U_{31} \\ U_{32} \\ U_{33} \end{pmatrix}, \textit{ vector de valores internos.}$$

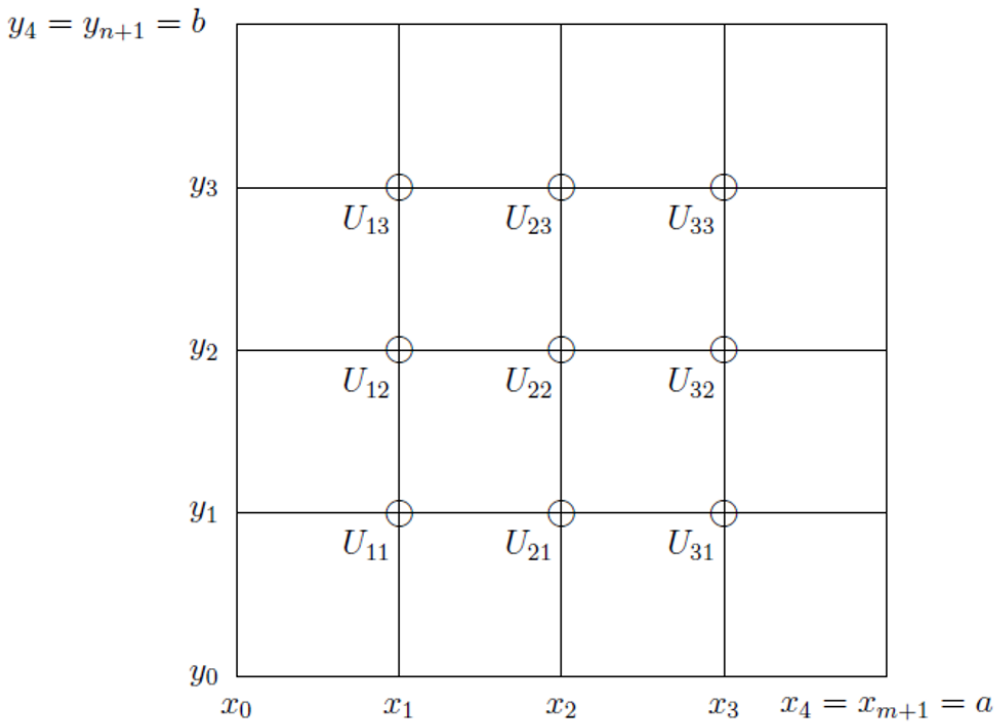

Figura 2.2: Valores internos de la malla.

De esta manera el sistema matricial $\vec{U}$ utilizándolo en la ecuación anterior:

$$U_{k+1,j} - 4U_{k,j} + U_{k-1,j} + U_{k,j+1} + U_{k,j-1} = h^2 f_{k,j}$$

se convierte en:

$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} U_{11} \\ U_{12} \\ U_{13} \\ U_{21} \\ U_{22} \\ U_{23} \\ U_{31} \\ U_{32} \\ U_{33} \end{pmatrix} =$$

$$= \begin{pmatrix} h^2 f_{11} - g_{10} - g_{01} \\ h^2 f_{12} - g_{02} \\ h^2 f_{13} - g_{14} - g_{03} \\ h^2 f_{21} - g_{20} \\ h^2 f_{22} \\ h^2 f_{23} - g_{24} \\ h^2 f_{31} - g_{30} - g_{41} \\ h^2 f_{32} - g_{42} \\ h^2 f_{33} - g_{34} - g_{43} \end{pmatrix}$$

Al final de todo el desarrollo, lo que se tiene que resolver es el sistemas de ecuaciones resultante.

$$A\bar{U} = \bar{f}$$

2.3. Ecuación de Poisson resuelta con elemento finito

$$-U_{xx} = q \qquad 0 \leq x \leq L \tag{2.5}$$

Consideremos las siguientes condiciones de frontera:

$$U(0) = U(L) = 0$$

Para aplicar el método de los elementos a la ecuación de Poisson (ec. 2.5), propongamos una función de prueba, continua ϕ_x, que cumpla las mismas condiciones de frontera que la función U, es decir:

$$\phi(0) = \phi(L) = 0$$

y reescribamos el problema de condiciones de frontera en forma variacional (sección 4.1.1).

$$\int_0^L U_{xx}(x)\phi(x)dx + \int_0^L q(x)\phi(x)dx = 0 \tag{2.6}$$

Resolviendo la primera integral:

$$\int_0^L U_{xx}(x)\phi(x)dx = [U_{xx}(x)\phi_x(x)]_0^L - \int_0^L U_x(x)\phi_x(x)dx = -\int_0^L U_x(x)\phi_x(x)dx$$

Ahora sustituimos la ecuación anterior en la ecuación 2.6

$$-\int_0^L U_x(x)\phi_x(x)dx + \int_0^L q(x)\phi(x)dx = 0$$

Multiplicamos por menos y despejamos:

$$\int_0^L U_x(x)\phi_x(x)dx = \int_0^L q(x)\phi(x)dx \tag{2.7}$$

La ecuación anterior es continua por partes para todo $\phi(x)$ y satisface las condiciones de frontera. Introducimos una malla en el intervalo $[0, L]$, con puntos

$$x_j = j \triangle x, \quad j = 0, \quad ..., n+1$$

donde

$$\triangle x = \frac{L}{n+1}.$$

Elegimos una base para $\phi(x)$, tomaremos las funciones "sombrero", $\phi_j(x)$.

Resolveremos la ecuación utilizando lo siguiente:

$$\phi(x) = \sum_{j=1}^{n} a_j \phi_j(x)$$

Donde:

$$\phi_j(x) = \begin{cases} 0 & \text{para } o \leq x \leq x_{j-1} \\ \frac{1}{\triangle x}(x - x_{j-1}) & \text{para } x_{j-1} \leq x \leq x_j \\ 1 - \frac{1}{\triangle x}(x - x_j) & \text{para } x_j \leq x \leq x_{j+1} \\ 0 & \text{para } x \geq x_{j+1} \end{cases} \tag{2.8}$$

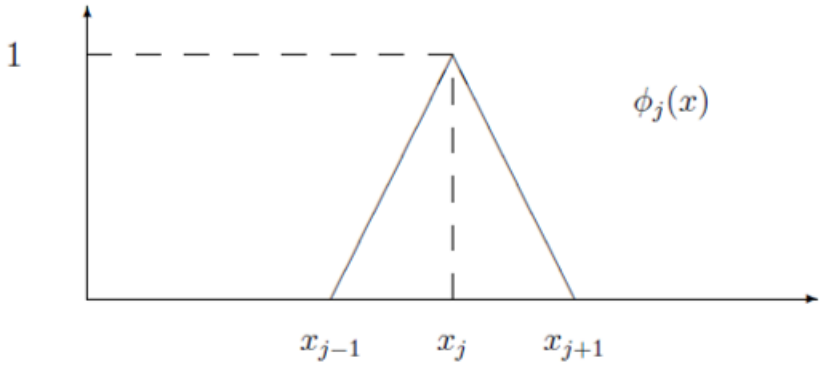

Figura 2.3: Gráfica de la función sombrero.

Con esta construcción: $\phi_j(x_i) = \delta_{ij}$ y

$$\phi_j'(x) = \frac{\partial \phi_j}{\partial x} = \begin{cases} 0 & \text{para } o < x < x_{j-1} \\ \frac{1}{\triangle x} & \text{para } x_{j-1} < x < x_j \\ -\frac{1}{\triangle x} & \text{para } x_j < x < x_{j+1} \\ 0 & \text{para } x > x_{j+1} \end{cases} \tag{2.9}$$

Sea

$$\phi(x) = \sum_{j=1}^{n} a_j \phi_j(x)$$

y

$$\phi(x_i) = a_i \; para \; i = 1, ..., n$$

y

$$\phi(0) = \phi_1(0) = 0$$

y

$$\phi(L) = \phi_n(L) = 0,$$

para que $\phi(x)$ satisfaga las condiciones de frontera.

Las funciones sombrero (ec. 2.8) nos ayudan mucho como base, porque son ortonormales, es decir:

$$\int_0^L \phi_j(x)\phi_k(x)dx = 0 \quad donde \quad j - k > 1.$$

Usando E.F. buscamos una solución aproximada que satisfaga las funciones base $\phi_i(x)$ para $i = 1, ..., n$:

$$\int_0^L U_x(x)\phi_x(x)dx = \int_0^L q(x)\phi(x)dx$$

Para $\phi = \phi_i$, donde $i = 1, ..., n$. Expandimos la solución $U(x)$ utilizando las funciones sombrero ϕ_i como base.

$$U(x) \approx U_h(x) = \sum_{j=1}^{n} b_j\phi_j(x)$$

$$\Rightarrow U_h'(x) = \sum_{j=1}^{n} b_j\phi_j'(x)$$

Esto simplifica la ecuación 2.7 y resolvemos para $\phi = \phi_i, \quad i = 1, ..., n$

$$\int_0^L U_h'(x)\phi_i'(x)dx = \int_0^L q(x)\phi_i(x)dx, \quad para \quad i = 1, ..., n$$

Donde

$$f'(x) = \frac{\partial f}{\partial x}$$

$$LI = \int_0^L U_h'(x)\phi_i'(x)dx = \int_0^L \sum_{j=1}^{n} b_j\phi_j'(x)\phi_i'(x)dx = \sum_{j=1}^{n} C_{i,j}b_j$$

Donde

$$C_{i,j} = \int_0^L \phi_j'(x)\phi_i'(x)dx$$

es la matriz conocida como de rigidez.

Para encontrar los coeficientes b_j que son partes de la solución $U(x)$ debemos resolver las n ecuaciones lineales.

$$LI = \sum_{j=1}^{n} C_{i,j}b_j = LD = \int_0^L q(x)\phi_i(x)dx = q_i \tag{2.10}$$

Para $i = 1, ..., n$ con $q_i = \int_0^L q(x)\phi_i(x)dx$.

Aproximaremos la solución usando una combinación lineal de las funciones sombrero:

$$U(x) \approx \sum_{j=1}^{n} b_j\phi_j(x)$$

por lo tanto solamente necesitamos conocer los coeficientes de b_j para definir nuestra solución $U(x)$ y usando E.F. resolveremos la siguiente ecuación para el vector $\overline{b} = (b_1, b_2, ..., b_n)$:

$$\sum_{j=1}^{n} b_j \int_0^L \phi_{j,x}(x)\phi_{i,x}(x)dx = \int_0^L q(x)\phi_i(x)dx$$

o

$$\sum_{j=1}^{n} C_{i,j}b_j = q_i$$

Para $i = 1, ..., n$

Mostremos que la matriz C es tridiagonal, para esto tendremos que resolver la integral por partes:

$$C_{ij} = \int_0^L \phi_{j,x}(x)\phi_{i,x}(x)dx =$$

$$\int_0^{j-1} \phi_{j,x}(x)\phi_{i,x}(x)dx + \int_{j-1}^{j} \phi_{j,x}(x)\phi_{i,x}(x)dx + \int_{j}^{j+1} \phi_{j,x}(x)\phi_{i,x}(x)dx + \int_{j+1}^{L} \phi_{j,x}(x)\phi_{i,x}(x)dx = INT$$

Cuando $i = j - 1 \Rightarrow j = i + 1$, tomamos los valores de $\phi_{j,x}(x)$ en la ecuación 2.10 y sustituimos:

$$INT = \int_{j-1=i}^{j=i+1} \frac{1}{\triangle x}(-\frac{1}{\triangle x})dx$$

$$= -\frac{1}{\triangle x^2} x \mid_{j-1}^{j} = -\frac{1}{\triangle x^2}(x_j - x_{j-1})$$

$$= -\frac{1}{\triangle x^2}(j \triangle x - (j-1) \triangle x) = -\frac{1}{\triangle x^2}(\triangle x(j + 1 - j)) = -\frac{1}{\triangle x}$$

Nos damos cuenta que el único intervalo que es diferente de cero en INT es $[j - 1, j]$

Por otro lado si $i = j$, utilizamos los valores de $\phi_{j,x}(x)$ en la ecuación 2.10 y sustituimos :

$$INT = \int_{j-1}^{j} \frac{1}{\triangle x}(\frac{1}{\triangle x})dx + \int_{j}^{j+1} -\frac{1}{\triangle x}(-\frac{1}{\triangle x})dx$$

$$= \frac{1}{\triangle x^2} x \mid_{j-1}^{j} + \frac{1}{\triangle x^2} x \mid_{j}^{j+1} = \frac{1}{\triangle x^2}(x_j - x_{j-1}) + \frac{1}{\triangle x^2}(x_{j+1} - x_j)$$

$$= \frac{1}{\triangle x^2}(j \triangle x - (j-1) \triangle x) + \frac{1}{\triangle x^2}((j+1) \triangle x - j \triangle x)$$
$$= \frac{1}{\triangle x^2}(\triangle x(j - j + 1)) + \frac{1}{\triangle x^2}(\triangle x(j + 1 - j))$$
$$= \frac{1}{\triangle x} + \frac{1}{\triangle x} = \frac{2}{\triangle x}$$

Nos damos cuenta que hay dos intervalos que son diferentes de cero en INT y son: $[j-1, j]$ y $[j, j+1]$

Por último si $i = j + 1 \Rightarrow j = i - 1$, retomamos los valores de $\phi_{j,x}(x)$ en la ecuación (2.10) y sustituimos:

$$INT = \int_{j=i-1}^{j+1=i} -\frac{1}{\triangle x}(\frac{1}{\triangle x})dx$$
$$= -\frac{1}{\triangle x^2} x \mid_j^{j+1} = -\frac{1}{\triangle x^2}(x_{j+1} - x_j)$$
$$= -\frac{1}{\triangle x^2}((j+1) \triangle x - j \triangle x)$$
$$= -\frac{1}{\triangle x^2}(\triangle x(j + 1 - j)) = -\frac{1}{\triangle x}$$

Nos damos cuenta que el único intervalo que es diferente de cero en INT es $[j, j+1]$

$$\Rightarrow \; C_{ij} = \int_0^L \phi_{j,x}(x)\phi_{i,x}(x)dx = \begin{cases} -\frac{1}{\triangle x} & \text{para } i = j-1 \\ \frac{2}{\triangle x} & \text{para } i = j \\ -\frac{1}{\triangle x} & \text{para } i = j+1 \\ 0 & \text{e.o.c} \end{cases} \tag{2.11}$$

Aproximamos q_i usando:

$$q_i = \int_0^L q(x)\phi_i(x)dx \approx q(x_i) \int_0^L \phi_i(x)dx$$

$$= q(x_i)(\int_{x_{j-1}}^{x_j} \frac{1}{\triangle x}(x - x_{j-1})dx + \int_{x_j}^{x_{j+1}} (1 - \frac{1}{\triangle x}(x - x_j)dx)$$

$$= q(x_i)(\int_{x_{j-1}}^{x_j} \frac{1}{\triangle x}xdx - \int_{x_{j-1}}^{x_j} \frac{1}{\triangle x}x_{j-1}dx + \int_{x_j}^{x_{j+1}} 1dx + \int_{x_j}^{x_{j+1}} -\frac{1}{\triangle x}xdx + \int_{x_j}^{x_{j+1}} \frac{1}{\triangle x}x_j dx)$$

$$= q(x_i)(\frac{1}{\triangle x}\frac{x^2}{2} \mid_{x_{j-1}}^{x_j} - \frac{1}{\triangle x}x_{j-1}(x_j - x_{j-1}) + (x_{j+1} - x_j) - \frac{1}{\triangle x}\frac{x^2}{2} \mid_{x_j}^{x_{j+1}} + \frac{1}{\triangle x}x_j(x_{j+1} - x_j)$$

$$= q(x_i)(\frac{1}{\triangle x}\frac{(x_j - x_{j-1})^2}{2} - \frac{1}{\triangle x}(j-1) \triangle x(\triangle x(j - j + 1)) + (\triangle x(j + 1 - j)) -$$

$$\frac{1}{\triangle x}\frac{(x_{j+1} - x_j)^2}{2} + \frac{1}{\triangle x}j \triangle x(\triangle x(j + 1 - j)))$$

$$= q(x_i)(\frac{1}{\triangle x}\frac{(\triangle x(j - j + 1))^2}{2} - (j-1)\triangle x + \triangle x - \frac{1}{\triangle x}\frac{\triangle x(j + 1 - j))^2}{2} + j\triangle x)$$

$$= q(x_i)(\frac{\triangle x}{2} - j \triangle x + \triangle x + \triangle x - \frac{\triangle x}{2} + j \triangle x)$$

$$= q(x_i)(2 \triangle x)$$

Si sustituimos lo anterior en la ecuación 2.10, llegamos a:

$$C\overline{b} = \triangle x \overline{q} o \frac{1}{\triangle x} C\overline{b} = \overline{q}$$

$$\begin{pmatrix} \frac{2}{\triangle x^2} & -\frac{1}{\triangle x^2} & 0 & \cdots \\ -\frac{1}{\triangle x^2} & \frac{2}{\triangle x^2} & -\frac{1}{\triangle x^2} & \ddots \\ 0 & \ddots & \ddots & \ddots \\ \vdots & \ddots & -\frac{1}{\triangle x^2} & \frac{2}{\triangle x^2} \\ & & & \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \\ \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \\ \end{pmatrix}$$

Por lo tanto el sistema matricial resultante es exactamente el mismo que se obtendría por el método de la inversa de C:

$$\overline{b} = C^{-1} \triangle x\overline{q}$$

Una vez encontrado el vector $\overline{b}$, la solución U del problema es la aproximación:

$$U(x) \approx \sum_{j=1}^{n} b_j \phi_j(x)$$

Con esto se resuelve el problema de valores en la frontera (ec. 2.5)

Capítulo 3

Programación paralela

3.1. Introducción

La programación paralela, a grandes rasgos, es una forma de realizar cálculos o procesos computacionales en el que varios procesos son ejecutados simultáneamente. La computación paralela surgió con la necesidad de resolver problemas que involucraban un gran número de cálculos. El interés por la computación paralela se remonta a finales de los años 50 del siglo XX. Tal interés se vio reflejado en supercomputadoras, que surgieron entre 1960 y 1970. A mediados de 1980, un nuevo tipo de computadoras paralelas fue creado cuando el proyecto "Concurrent Computation" del Caltech construyó una supercomputadora para aplicaciones científicas. Demostraron que se podía lograr un gran rendimiento y a un bajo costo, utilizando microprocesadores convencionales. Fue así como los Clusters se postularon como los mejores candidatos para el cálculo masivo. Un Cluster es un tipo de computadora paralela, construido a partir de múltiples computadoras, todas ellas conectadas a una misma red. En la década de los 90 surgió un tipo de arquitectura (Application Programming Interface) orientada al manejo de Clusters que utilizaban lenguaje de programación C, C++ y Fortran, llamada MPI (Message Passing Interface); a su vez, para los microprocesadores que utilizaban memoria compartida, surgió OpenMP (Open Multi-Processing).

Desde que comenzaron los problemas relacionados con el tiempo de cálcu-

lo de los ordenadores, el objetivo de poder realizar el mayor número de operaciones posibles por segundo ha sido una necesidad para los usuarios, ya que por un lado al ser más rápidas, pueden resolver ciertos problemas y descifrar patrones que hubieran sido imposibles descubrir a mano. Por otro lado, los científicos están generando nuevas preguntas y modelos más complejos que se adaptan fácilmente a las capacidades del poder computacional.

La ciencia siempre ha apegado a seguir los pasos del método científico, pero algunas teorías son difíciles de probar por el costo que tiene implementar un experimento para poder demostrar dichas teorías. Sin embargo, gracias al poder computacional estos experimentos son mucho más baratos porque realizan simulaciones que ayudan a probar las teorías. Esto no solo pasa con nuevas teorías, sino también en la práctica con nuevos prototipos en la industria, transporte, ingeniería, etc. Hoy en día es posible analizar fenómenos con suficiente certidumbre en la vida real, por ejemplo el clima, la evolución del universo, las epidemias, etcétera.

Aunque pensamos que la velocidad actual de las computadoras es suficiente hay muchos casos donde un problema "sencillo" puede necesitar una gran cantidad de operaciones por unidad de tiempo. Además de la necesidad de velocidad computacional, también es necesario considerar el problema del almacenamiento de grandes cantidades de información. Si logramos adquirir la velocidad deseada, pero en una computadora que solamente tenemos acceso a algunos gigabytes de la RAM es inútil porque no tendremos la capacidad de almacenamiento requerido para dicha velocidad.

La forma más sencilla de obtener más poder computacional es esperar que las tecnologías existentes sigan evolucionando con la ley de Moore. Está claro que la ley de Moore tiene un límite, es decir, llegará a un límite físico en donde no va a ser posible añadir más transistores a un microprocesador. Una solución para este problema se puede ver en un ejemplo sencillo. Consideremos un agricultor, quien únicamente puede cultivar 100 metros cuadrados de tierra por día, pero se requiere por situaciones emergentes, que se cultiven 1000 metros cuadrados de tierra por día; la solución más lógica es contratar otros 9 trabajadores que puedan cultivar la misma cantidad de tierra por día, con esto, se pueden cultivar los 1000 metros de tierra que se requieren, esto incrementa la eficacia del trabajo de 1 a 10.

Utilizando esta analogía, nuestro agricultor es el procesador y los 1000

metros cuadrados son la cantidad de trabajo que se requiere realizar, por lo tanto, se necesitan de varias computadoras interconectadas realizando el mismo trabajo para aumentar el rendimiento de solución de problemas, a esto se le conoce como "computadora paralela".

3.2. Conceptos básicos

En tiempos atrás los algoritmos computaciones se solían escribir para su cómputo en serie, es decir, se ejecutaban en un solo procesador. Cuando se intenta resolver un problema con un algoritmo computacional, la unidad de procesamiento de la computadora, ejecuta una por una las instrucciones del algoritmo, dando tiempo a cada instrucción de completarse antes de que comience a ejecutarse la siguiente. Algo muy diferente ocurre con la programación paralela, este tipo de programación puede ejecutar diferentes procesos al mismo tiempo; esto se logra particionando el algoritmo de tal manera que las instrucciones puedan ejecutarse de forma simultánea con los demás elementos de la partición.

3.2.1. Tareas

Un problema "grande" se puede dividir en "subproblemas" más pequeños a los cuales les llamaremos TAREAS (fig. 3.1). Una tarea la podemos pensar como un conjunto de instrucciones que va a ejecutar el procesador. Cabe mencionar que las tareas no tienen por qué ser iguales, en cuestión de tamaño y de instrucciones.

3.2.2. Hilos

En inglés THREAD, un hilo (fig. 3.2), también llamado: proceso, hebra, proceso ligero o sub proceso, es una lista de tareas secuenciales, en otras palabras un hilo es una tarea que puede ser ejecutada simultáneamente con otra tarea.

Figura 3.1: Tareas.

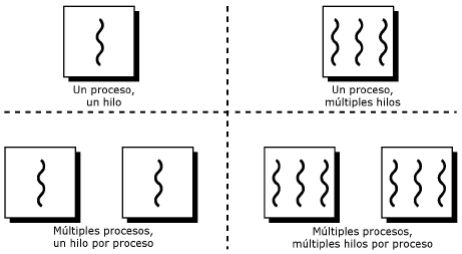

Figura 3.2: Hilos.

3.2.3. Granularidad

Hace referencia al tamaño de las tareas, teniendo en cuenta la independencia entre tareas. Existen dos tipos (fig. 3.3):

- Gruesa: Se refiere a una tarea de "gran tamaño", con una alta independencia entre tareas y muy poca necesidad de sincronización.
- Fina: Poco trabajo entre tareas, poca independencia entre las tareas, pero considerando una alta necesidad de sincronización.

3.2.4. Independencia

La independencia entre datos es básicamente mantener los datos separados o agrupados, de tal manera que las tareas no puedan utilizar el mismo

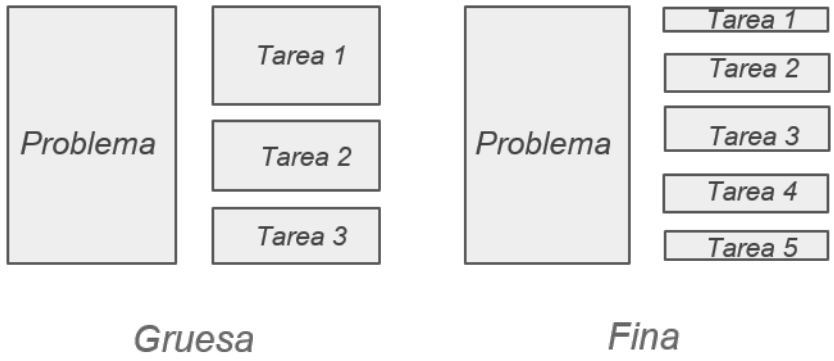

Figura 3.3: Granularidad.

conjunto de datos al mismo tiempo. Esta independencia asegura que los datos no puedan ser modificados por una tarea, ajena a la que está trabajando en ese momento con los datos.

3.2.5. Sincronización

Por definición, se refiere al ajuste temporal entre eventos. Es decir, la coordinación entre tareas e hilos. Un programa escrito en paralelo necesita la correcta ejecución temporal de las tareas e hilos. La coordinación de los eventos, está fuertemente correlacionada con la arquitectura del hardware, es decir, la manera en que las tareas e hilos comparten la información.

3.2.6. Ley de Moore

Una idea propuesta por el cofundador de Intel, que básicamente dice que el número de transistores que un microprocesador puede tener, se duplica aproximadamente cada dos años. Comparemos los procesadores intel 4004 lanzado en 1971 que contenía 2300 transistores, con el procesador 80286, puesto al mercado en febrero de 1982, contenía 134000 transistores y a su vez comparado con el procesador Pentium II lanzado en mayo de 1997 tenía 7.5 millones de transistores [24].

La ley de Moore claramente tiene una limitación, y es que llegará un momento en que la tecnología actual no permita seguir miniaturizando los transistores, es decir, no podrán meterse más de cierta cantidad de transistores en una determinada área de chip.

3.2.7. Ley de Amdahl y Ley de Gustafson

Idealmente la aceleración a partir de la paralelización es lineal, doblar el número de elementos de procesamiento debe reducir a la mitad el tiempo de ejecución y doblarlo por segunda vez debe nuevamente reducir el tiempo a la mitad. Sin embargo la práctica nos sugiere que no sucede así, muy pocos algoritmos logran una aceleración óptima. La rapidez potencial de un algoritmo en una plataforma de cómputo en paralelo está dada por la ley de Amdahl (formulada originalmente por Gene Amdahl en la década de 1960).

La fórmula es la siguiente:

$$F = F_a((1 - F_m) + \frac{F_m}{A_m})$$

Donde F tiempo de ejecución mejorado, F_m fracción de tiempo que utiliza el sistema del subsistema mejorado, F_a el tiempo de ejecución antiguo y A_m el factor de mejora que se ha introducido en el subsistema mejorado [19].

La idea básica de este postulado radica en que el algoritmo es el que decide la mejora de velocidad y no el número de procesadores. Finalmente llega un punto en que el algoritmo ya no se puede paralelizar más.

3.3. Sistemas de memoria compartida

A grandes rasgos un sistema de Multiprocesadores de Memoria Compartida (MMC), son múltiples procesadores que comparten la información de la memoria principal; esto quiere decir que todos los procesadores están conectados a la misma memoria principal, a través de distintos métodos de

interconexión. El acceso a los módulos por parte de los procesadores se realiza en paralelo, pero cada módulo solamente puede atender una petición en cada instante de tiempo. A este tipo de arquitectura se le conoce como UMA (Uniform Memory Access). Características de los MMC:

- Tiempos de acceso a memoria uniformes, ya que todos los procesadores se encuentran igualmente comunicados con la memoria principal.
- Las lecturas y escrituras de cada uno de los procesadores tienen exactamente las mismas latencias (es la suma de retardos temporales dentro de una red).
- La programación es mucho más fácil que en los MMD, debido a que la gestión de la memoria de cada módulo es transparente para el programador.
- Al acceder simultáneamente a la memoria se producen colisiones y esperas, lo que es un problema.
- Debido a la organización de la arquitectura, es poco escalable en número de procesadores, debido a que puede surgir un cuello de botella si se aumenta el número de CPU's.

3.4. Sistema de memoria distribuida

Un procesador con memoria distribuida (MMD) posee su propia memoria local. De esta manera, cada procesador puede acceder tanto a su memoria local, como a la memoria remota de cualquiera del resto de procesadores. Este tipo de arquitectura se denomina NUMA (Non-Uniform Memory Access). Contrastando los procesadores de memoria compartida, el usuario tiene que mover los datos o distribuirlos cuando se necesite.

Ventajas:

- Las computadoras con memoria distribuida son fáciles de escalar, es decir, se puede ir agregando más memoria y/o más procesadores.

Desventajas:

- El acceso remoto a memoria es lento.
- La programación puede ser complicada.

3.5. Open MP

Ante la necesidad de una potencia de cálculo cada vez mayor, los desarrolladores de sistemas informáticos comenzaron a pensar en utilizar varias de sus computadoras existentes de manera conjunta. Lo cual originó el surgimiento de la programación paralela y el comienzo de un nuevo campo. Un desafío en las máquinas paralelas es el desarrollo de códigos capaces de utilizar las capacidades del hardware al 100 %, para resolver problemas más grandes en menos tiempo. En los últimos años se ha creado un nuevo estándar de la industria con el objetivo de servir como una buena base para el desarrollo de programas paralelos en máquinas de memoria compartida: OpenMP

Las máquinas de memoria compartida existen en la actualidad. Tiempo atrás, cada empresa desarrollaba su propio estándar de instrucciones para compilación y llamado de bibliotecas, lo que permitía a un programador hacer uso de las capacidades de una máquina paralela específica. Un esfuerzo de estandarización anterior fue el lenguaje ANSI X3H5 el cual nunca se adoptó formalmente, ya que, por un lado, no existía un fuerte apoyo de las empresas y, por otro lado, las máquinas de memoria distribuida, con sus propias bibliotecas de paso de mensajes más estándar PVM y MPI, parecían una buena alternativa. Pero en 1996-1997, apareció un nuevo interés en una interfaz estándar de programación de memoria compartida, principalmente debido a:

1. Un nuevo interés por parte de las empresas en las arquitecturas de memoria compartida.

2. La opinión de una parte de las empresas, que la paralelización de los programas que utilizan interfaces de paso de mensajes es complicada y larga, sería deseable una interfaz de programación más abstracta.

OpenMP es el resultado de un gran acuerdo entre empresas de hardware y desarrolladores de compiladores, se considera un estándar de la industria: se pueden utilizar para especificar el paralelismo de memoria compartida en Fortran y Programas C / C ++.

OpenMP consolida todo esto en una única sintaxis y semántica y finalmente ofrece portabilidad de fuente única para el paralelismo de memoria compartida.

Participantes en el desarrollo:

Las especificaciones de OpenMP son propiedad, escrita y mantenida por el OpenMP Architecture Review Board, que es una unión de las compañías que participan activamente en el desarrollo de la interfaz estándar de programación de memoria compartida. En el año 2000, los miembros permanentes de OpenMP ARB fueron:

- Departamento de Energía de EE. UU., a través de su programa ASCI
- Compaq Computer Corp.
- Fujitsu
- Compañía Hewlett-Packard
- Intel Corp.
- Máquinas de negocios internacionales
- Kuck Associates, Inc.
- Incorporar Silicon Graphics
- Sun Microsystems

Además de OpenMP ARB, un gran número de compañías contribuyen al desarrollo de OpenMP al usarlo en sus programas y compiladores o al informar problemas, comentarios y sugerencias a OpenMP ARB.

3.5.1. Principales instrucciones

Las instrucciones más importantes de OpenMP son las encargadas de definir las llamadas regiones paralelas. Esta es un bloque de código que será ejecutado por múltiples subprocesos que se ejecutan en paralelo. Como una región paralela necesita ser "created/opened and destroyed/closed" (creada/abierta y destruida/cerrada), son necesarios dos instrucciones, formando un llamado:

!$OMP PARALLEL/!$OMP END PARALLEL.

Un ejemplo sería:

```
!$OMP PARALLEL

write(*,*) "Hola"

!$OMP END PARALLEL
```

Dado que el código encerrado entre las dos instrucciones es ejecutado por cada subproceso, el mensaje *Hola* aparece en la pantalla tantas veces como se usan subprocesos en la región paralela.

Antes y después de la región paralela, el código es ejecutado por un solo subproceso, que es el comportamiento normal en los programas en serie. Por lo tanto, se dice que en el programa también hay las llamadas regiones seriales.

Cuando un hilo que ejecuta una región en serie encuentra una región paralela, crea un equipo de hilos y se convierte en el hilo maestro del equipo. El hilo maestro también es miembro del equipo y participa en los cálculos. Cada hilo dentro de la región paralela obtiene un número de hilo único que varía desde cero, para el hilo maestro, hasta N_{p-1} , donde N_p es el número total de hilos dentro del equipo. En la figura 3.4, el ejemplo anterior se representa de forma gráfica para aclarar las ideas detrás del constructor de la región paralela.

Al comienzo de la región paralela, es posible imponer cláusulas que fijen ciertos aspectos de la forma en que funcionará la región paralela: por ejemplo, el alcance de las variables, el número de hilos, los tratamientos especiales de

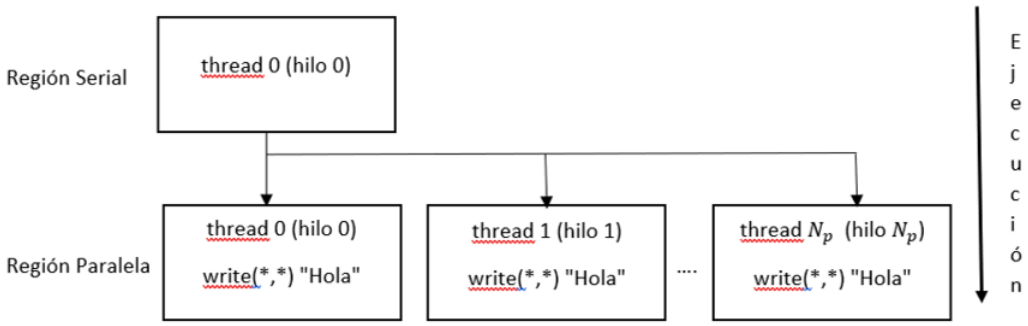

Figura 3.4: Región Paralela.

algunas variables, etc. La sintaxis utilizar es el siguiente:

!$OMP PARALLEL clause1 clause2 ...

...

!$OMP END PARALLEL

La instrucción !$ OMP END PARALLEL denota el final de la región paralela. Llegando a ese punto, todas las variables declaradas como locales para cada subproceso (PRIVATE) se borran y todos los subprocesos se eliminan, excepto el subproceso maestro, que continúa la ejecución más allá del final de la región paralela. Es necesario que el hilo maestro espere todos los otros hilos para terminar su trabajo antes de cerrar la región paralela; de lo contrario, se perdería información y/o no se realizaría el trabajo. De hecho, esta espera no es más que una sincronización entre los hilos en ejecución paralelos. ¡Por lo tanto, se dice que la instrucción !$ OMP END PARALLEL tiene una sincronización implícita. Cuando se incluye una región paralela en un código, es necesario cumplir dos condiciones para garantizar que el programa resultante cumpla con la especificación OpenMP:

1. Las instrucciones !$ OMP PARALLEL /! $ OMP END PARALLEL debe aparecer en la misma rutina del programa.

2. El código encerrado en una región paralela debe ser un bloque de código estructurado. Esto significa que no está permitido entrar o salir de la región paralela, por ejemplo, usando la instrucción GOTO.

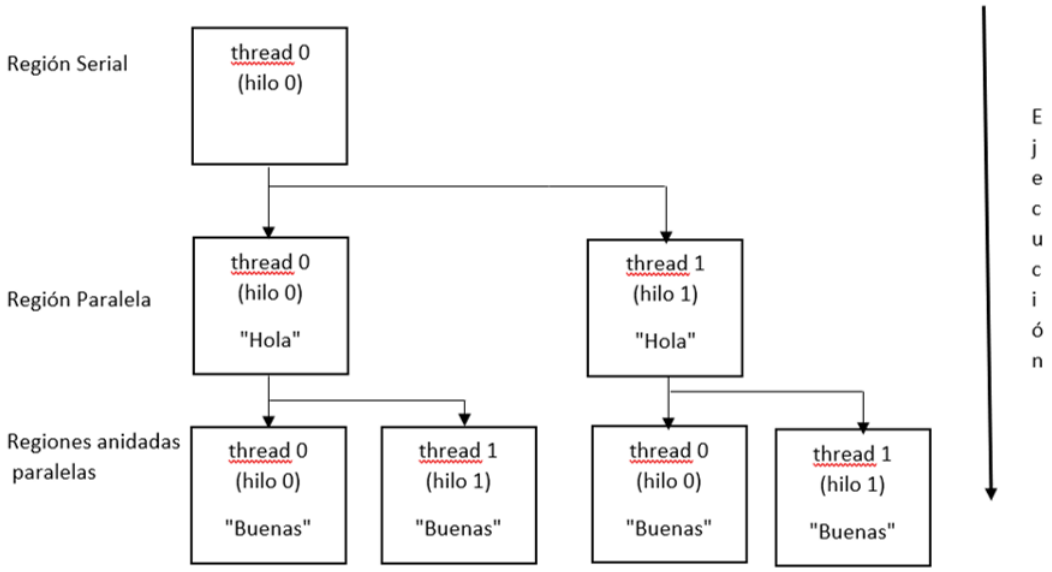

Figura 3.5: Regiones anidadas paralelas.

A pesar de estas dos reglas, no hay más restricciones a tener en cuenta al crear regiones paralelas. Sin embargo, es necesario tener cuidado al usarlas, ya que es fácil lograr programas de trabajo incorrectos, incluso cuando se consideran las restricciones anteriores.

Es posible anidar regiones paralelas dentro de regiones paralelas (fig. 3.5). Por ejemplo, si un hilo en un proceso paralelo encuentra una nueva región paralela, entonces crea un nuevo proceso y se convierte en el hilo maestro del nuevo proceso. Esta segunda región paralela se llama región paralela anidada. Un ejemplo de una región paralela anidada sería:

```
!$OMP PARALLEL

write(*,*) "Hello"

!$OMP PARALLEL

write(*,*) "Hi"

!$OMP END PARALLEL

!$OMP END PARALLEL
```

Si en ambas regiones paralelas se usa el mismo número de hilos N_p, se imprimirá en la pantalla un número total de mensajes $N_p^2 + N_p$. La estructura de árbol resultante se representa gráficamente en la figura 3.5, para el caso de usar dos hilos en cada nivel de anidamiento, $N_p = 2$. También se muestra en la misma figura la salida de pantalla asociada a cada uno de los hilos.

3.6. MPI

Por sus siglas en inglés "Message Passing Interface" que significan, interface de paso de mensajes. Es una biblioteca de funciones diseñada para programas que utilicen múltiples procesadores, utilizada en plataformas como C, C++ y Fortran, MPI es empleada en programación paralela para la sincronización entre procesos y permitir la exclusión mutua. Su característica principal es que no necesita una memoria compartida, por lo tanto, es excelente en sistemas de memoria distribuida. Cabe mencionar que la biblioteca MPI tuvo un desarrollo paulatino, mencionaremos un poco de su historia:

- Antes de 1992: Se desarrollaron varias bibliotecas de paso de mensajes, PVM, P4, ...
- 1992: En SC92 (super computing event), varios desarrolladores de bibliotecas de paso de mensajes acordaron desarrollar un estándar para pasar mensajes.
- 1994: Se publica el estándar MPI-1.0
- 1997: El estándar MPI-2.0 agrega creación y gestión de procesos, comunicación unilateral, comunicación colectiva extendida, interfaces externas y entradas y salidas paralelas.
- 2008: MPI-2.1 combina MPI-1.3 y MPI-2.0.
- 2009: MPI-2.2 correcciones y aclaraciones con extensiones menores.
- 2012: MPI-3.0 colectivos no bloqueantes, nuevas operaciones unilaterales, Fortran 2008 fijaciones.

- 2015: Entrada y salida colectiva sin bloqueo MPI-3.1, versión actual del estándar.

La mensajería de MPI se lleva a cabo mediante distintos programas seriales que se pueden comunicar mediante llamadas de funciones de bibliotecas. Las principales funciones podemos agruparlas en 4 grupos:

- Funciones para iniciar, manejar y terminar comunicaciones.
- Funciones para comunicar procesos.
- Funciones para realizar comunicaciones entre conjuntos de procesos.
- Funciones para crear tipos de datos arbitrarios.

3.6.1. Compilación y ejecución de programas MPI

La biblioteca MPI no prescinde de un método exacto con el cual un programa en paralelo sea ejecutado, esto depende de la arquitectura que se esté utilizando. Por lo general se emplea la instrucción *mpirun*.

MPICH.

Para compilar con mpich se utiliza la instrucción mpif90, mientras que para ejecutar se usa mpirun.

Sin embargo se pueden usar las siguientes sentencias: $< hostname >$, $<procs >$, $< progname >$, $[< login >]$. *hostname* es el nombre de la computadora, *procs* es el número de procesos que serán ejecutados, *progname* es el programa a ejecutar y *login* es el acceso del usuario a la computadora.

3.6.2. Principales sentencias

Estructura MPI INIT: Inicializa la estructura de comunicación de MPI entre los procesos. Ninguna función de comunicación MPI funcionará sin esta instrucción al inicio.

Estructura MPI Comm Rank: Determina el rango (identificador) del proceso que lo llama dentro del comunicador seleccionado.

Estructura MPI Comm Size: Determina el tamaño del comunicador seleccionado, es decir, el número de procesos que están actualmente asociados a este.

Estructura MPI Finalize: Termina el entorno de ejecución de MPI.

3.7. Banderas de Optimización

Las banderas de optimización realizan un análisis completo del código y activan funciones predefinidas del compilador. El objetivo del compilador es reducir el costo de compilación y hacer de que la depuración produzca los resultados esperados reduciendo el tiempo de ejecución. Por ejemplo, si declaramos una variable, pero ésta es constante el compilador quita la memoria reservada para esa variable y solamente le asigna el valor correspondiente.

Al activar las banderas de optimización el compilador comienza a mejorar el rendimiento y/o el tamaño del código, esto trae como resultado un aumento al tiempo de compilación del programa. El compilador realiza la optimización basándose en el conocimiento que tiene del programa.

El compilador de gfortran tiene 3 tipos de banderas básicas:

- -O/-O1: El compilador intenta reducir el tamaño del código y el tiempo de ejecución, sin realizar ninguna optimización que requiera una gran cantidad de tiempo de compilación.
- -O2: El compilador realiza casi todas las optimizaciones admitidas que no implican un incremento entre la velocidad y el tamaño. En comparación con -O, esta opción aumenta tanto el tiempo de compilación como el rendimiento del código generado.
- -O3: Optimización agresiva, esto significa que esta bandera modifica el código fuente y los resultados pueden ser alterados. Esta bandera no se recomienda mucho.

- -openmp: El compilador analiza el programa, detecta las rutinas y regiones del código que pueden transformarse en tareas que se van a ejecutar en paralelo. Esta bandera ayuda a disminuir los tiempos de ejecución.

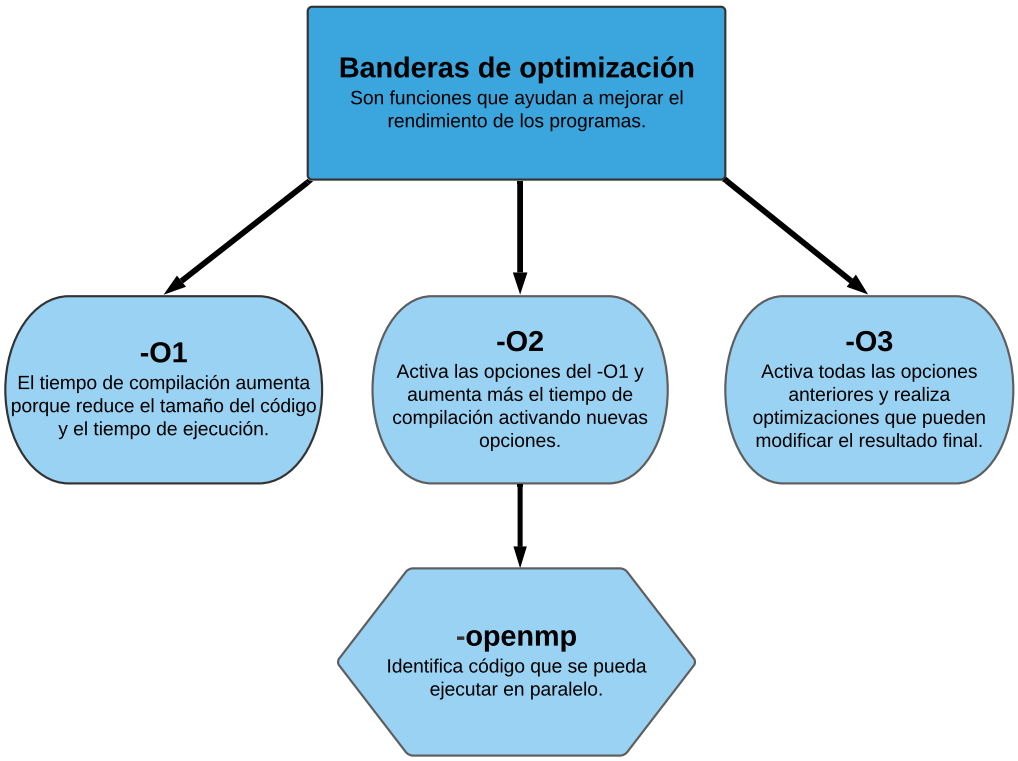

Figura 3.6: Banderas de optimización.

3.8. Importancia del Cómputo Paralelo

Desde los inicios de las primeras máquinas paralelas el cómputo en paralelo fue un problema abierto a resolver en estas arquitecturas. El multiprocesamiento siempre ha estado en la mente de los físicos y matemáticos como una aproximación a resolver problemas cuyos modelos matemáticos son muy

complejos o requieren particiones o mallas muy finas que dan lugar a matrices muy grandes del orden de millones o cientos de millones de elementos.

Diversos modelos de programación paralela que incluyen lenguajes de programación y bibliotecas especializadas se han desarrollado durante varias décadas, para esto los investigadores en las ciencias de la computación desarrollaron los paradigmas de paralelismo de datos, memoria compartida, envío de mensajes y microtasking. Lenguajes tales como HPF, Parallel Pascal, C y más recientemente Fortran 90, usaron algún paradigma de programación. Bibliotecas especializadas en envío de mensajes; también se desarrollaron tales como P4, Parmacs, PVM, MPL de IBM entre otras, también aprovecharon el cada vez más extendido uso de supercomputadoras de 4 o más unidades de computo (CPUs) seriales o vectoriales.

Con el surgimiento de las granjas de PCs o clústers que usaban principalmente software libre (OS por sus siglas en inglés) surgió otra revolución en la construcción de equipos de decenas de PCs hasta miles o cientos de miles usando tecnología de escritorio. Proyectos tales como el foro de MPI (Interface de Envío de Mensajes por sus siglas en inglés) que se unieron al movimiento de software libre, aceleraron el desarrollo de diversas aplicaciones. Después surgió el consorcio OpenMP basado en el modelo de memoria compartida y tuvo una aceptación inmediata ya que había antecedentes sobre el uso de estas directivas como por ejemplo en CRAY.

Actualmente las técnicas son cada vez más sofisticadas y se utilizan mezclas de modelos de programación llamadas híbridas, usando OpenMP/MPI con implementaciones de compiladores de código abierto como la suite de GCC y Open MPI. Otra tendencia está tomando relevancia y es la programación de procesadores gráficos o GPUs usando OpenCL o CUDA basados en modelos de programación de memoria compartida. El desarrollo de nuevos procesadores para móviles está permitiendo portar aplicaciones que se ejecutaban en grandes centros de datos a infraestructuras más pequeñas pero no menos veloces. En resumen cada vez es más común la ejecución paralela de una gran diversidad de aplicaciones científicas y de propósito general.

3.9. Clúster KNL

Durante el desarrollo y ejecución de los códigos realizados en esta tesis, se tuvo acceso al clúster KNL (fig. 3.8) que forma parte de la infraestructura del Laboratorio Nacional de Supercómputo (NLS) de la Benemérica Universidad Autónoma de Puebla (BUAP). El cual cuenta con las características señaladas en la figura (3.7).

Cluster KNL	
cantidad de nodos	45
cores x nodo	68
memoria RAM x nodo (GB)	192
Almacenamiento (TB)	230
Total de cores	3060
Total de memoria (GB)	13056

Descripción
1 nodo de control (2x Xeon E5-2630 10C, 128GB RAM DDR4)
1 nodo de login (2x Xeon E5-2630 10C, 128GB RAM DDR4)
4 nodos MDS, OSS (2x Xeon E5-2630 10C, 128GB RAM DDR4)
45 nodos cálculo (1x XeonPhi 7250 68C, 192GB RAM DDR4)
Red de Inteconexión OmniPath 100Gbps
Red administracion 1Gbps Ethernet
Arreglo de discos (120 DD 4TB SAS)
Sistema de Archivos ZFS
Linux Centos 7.3
Sitema de Colas Torque 6.x/Maui

Figura 3.7: Características Clúster KNL.

Mientras que la figura (3.9) es una fotografía real.

Figura 3.8: Imagen Clúster KNL.

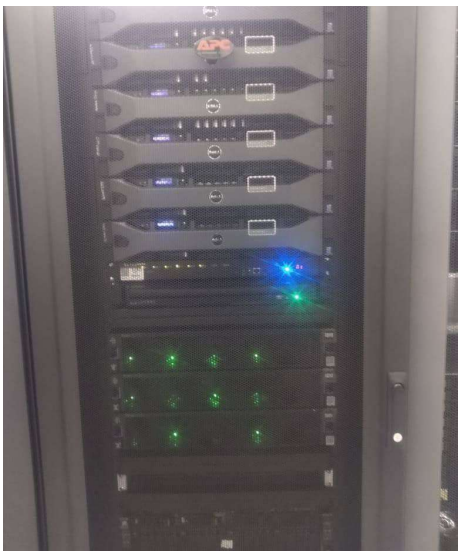

Figura 3.9: Fotografía del Clúster KNL del Laboratorio Nacional de Super Cómputo.

Capítulo 4

Análisis Matemático

4.1. Planteamiento matemático de los ejemplos tratados

En este capítulo vamos a introducir el Método de los Elementos Finitos, desde el punto de vista matemático. Para ello usemos como ejemplo la siguiente ecuación.

$$\Delta u + f = 0 \quad \textit{con las condiciones de borde} : \quad u = 0 \textit{ en } \Gamma \tag{4.1}$$

La solución de la ecuación de Poisson consiste en encontrar una función u tal que cumpla la ecuación en cada instante de tiempo, las condiciones de frontera, y que satisfaga los criterios del orden de la derivada. Esto es a lo que se le conoce como solución fuerte del problema.

Sin embargo con el método del los elementos finitos, la solución que se busca podemos encontrarla haciendo más flexibles algunos criterios de la solución fuerte; por ejemplo, que la función no cumpla la ecuación punto a punto, si no en un promedio de puntos, o que el orden de la derivada sea de un grado menor. Esto es la formulación débil del problema.

4.1.1. Formulación Variacional

Los métodos variacionales usualmente vienen de problemas de la física matemática. Cuando el problema que nos interesa resolver se puede escribir en su forma variacional, es equivalente a resolver la formulación débil.

Para plantear un problema en su forma variacional, se puede considerar el siguiente algoritmo:

- Definir un espacio de funciones, tal que, la solución se encuentre en ese espacio.
- Según la teoría del principio variacional, la solución del problema es la función en el espacio propuesto, que hace mínimo, máximo, o estacionario un funcional.

Una vez establecido el principio variacional que se aplicará.

- Proponemos un subespacio de funciones de dimensión finita, del espacio en donde está la solución.
- Aproximamos la solución con una combinación lineal de las funciones base del subespacio.
- Se encuentran los coeficientes de tal manera que la solución minimice, maximice o haga estacionario el funcional correspondiente.

Lo anterior en esencia es el método de Rayleigh-Ritz.

La ventaja de los métodos variacionales es que brindan la posibilidad de encontrar soluciones aproximadas de manera sencilla; lo complicado de este método es encontrar el subespacio de funciones adecuado al problema en cuestión. El método de elemento finito es básicamente el método de Ritz pero con una manera metódica para encontrar el subespacio de funciones.

4.1.2. Funciones Base

También llamadas funciones de prueba. Son utilizadas para aproximar funciones; dichas expresiones tienen que cumplir algunas características, tales como cumplir exáctamente con las condiciones de frontera. Si somos capaces de encontrar una función ψ de tal manera que $\psi|_\Gamma = u|_\Gamma$, donde u es la solución exacta de la ecuación diferencial, y además las funciones base propuestas $(\phi_m, m = 1, 2, ..., M)$ tales que $\phi_m|_\Gamma = 0$, para cada m, entonces se puede aproximar la función como:

$$u \simeq \hat{u} = \psi + \sum_{m=1}^{M} a_m \phi_m \tag{4.2}$$

Estas funciones tienen que elegirse de manera que al aumentar M se garantice una mejor aproximación en (4.2) o usar diferentes funciones base.

Resulta interesante la construcción de una función base para un problema de más de una dimensión.

En esta sección describiremos la metodología utilizada en el proceso de la creación de las funciones base que fueron empleadas en el algoritmo computacional que respalda los datos de esta investigación:

1) Elegir el tipo de elementos con los cuales queremos discretizar el dominio de trabajo, ya sean cuadrados o triángulos. La decisión está en función del problema en concreto con el que se está trabajando; resulta más sencillo en algunos casos utilizar elementos de tipo triangular que de tipo cuadrado y viceversa.

2) Una vez asignado el tipo de elemento a utilizar, se tiene que determinar el número de nodos que van a formar cada elemento; en nuestro caso el número de nodos optado fue de 3. Cabe mencionar que existe la posibilidad de tomar 6 nodos o más, esto es con base en la función de interpolación empleada.

3) Necesitamos una función de interpolación, la cual servirá para representar el dominio en términos de las funciones base. De la misma manera que los pasos anteriores, no existe una función de interpolación única, todo

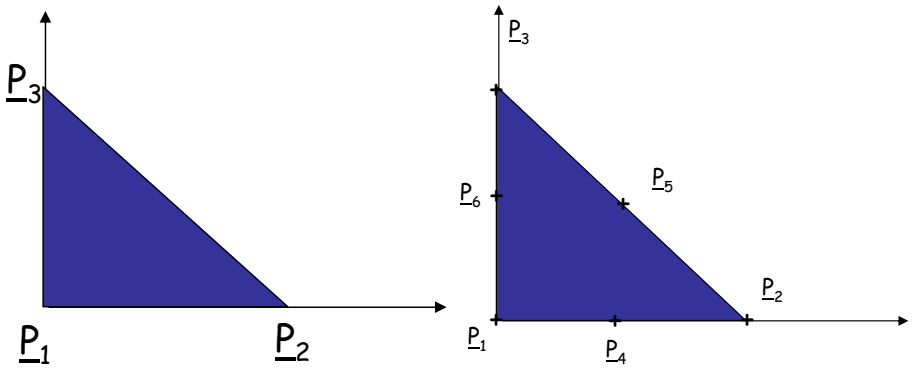

Figura 4.1: Elemento triangular de 3 nodos y 6 nodos.

se elige en función de la ecuación diferencial y del dominio en el que se está trabajando. La función propuesta es la siguiente:

$$u(x, y) = c_1 + c_2 x + c_3 y \tag{4.3}$$

Donde x, y representan las coordenadas de cada uno de los nodos que forman el elemento.

4) Con la función de interpolación y con las coordenadas de los nodos de cada uno de los elementos, se puede construir una matriz de coeficientes, que al resolver el sistema, quedan únicamente las funciones base ya evaluadas en cada uno de los nodos.

4.1.3. Método de Residuos Ponderados

Para englobar la teoría general de los elementos finitos es necesario explicar el método de residuos ponderados, específicamente el método de Galerkin.

El procedimiento para encontrar los coeficientes para la solución aproximada propuesta, es lo que se conoce como método de residuos ponderados. Se define el error de la solución en el dominio como:

$\mathrm{R}_\omega = u - \hat{u}$

Es evidente que la definición de error únicamente aplica cuando se conoce la solución exacta de la ecuación (u).

La definición de método variacional permite reescribir la ecuación diferencial en un término integral.

De hacer el error más pequeño, surgen expresiones del tipo:

$$\int_\Omega W_l(u - \hat{u})\ d\Omega \equiv \int_\Omega W_l R_\Omega\ d\Omega = 0 \tag{4.4}$$

con $l = 1, 2, ..., M$

W_l son las funciones de peso. Para que la convergencia esté garantizada, es decir, que $\hat{u} \longrightarrow u$ cuando $M \longrightarrow \infty$ se puede expresar mediante la ecuación (4.4) haciendo que se satisfaga para todo l para $M \longrightarrow \infty$. Dicha convergencia solo puede ser cierta si $R_\Omega \longrightarrow 0$ en todo el dominio .

Si sustituimos la ecuación (4.2) en (4.4) como resultado se obtiene un sistema de ecuaciones:

$$Ka = f \tag{4.5}$$

dicha expresión permite calcular los coeficientes a_m de la ecuación (4.2).

Ahora bien, el método de Galerkin es uno de los métodos de residuos ponderados más utilizados ya que las funciones de peso se eligen como las mismas funciones de prueba. Este métodoha sido empleado en el desarrollo del presente trabajo.

4.2. Casos de estudio

En la sección 1.6 se trabajó el método de elemento finito aplicado a una ecuación diferencial de una sola variable. En este apartado, describiremos con

detalle, cómo obtener la solución de una ecuación de dos dimensiones bajo el mismo principio.

$$U_{xx} + U_{yy} = -q \tag{4.6}$$

$$con\ (x, y)\ en\ \Omega$$

$$y$$

$$u_\Gamma = cte$$

$$donde\ \Gamma\ es\ la\ frontera\ de\ la\ región\ \Omega$$

.

Como primer paso, discretizamos el dominio de trabajo (fig. 4.2), para ello elegimos el tipo de elemento que utilizaremos, en este caso, se eligieron elementos triangulares, con 3 nodos cada uno (fig. 4.1).

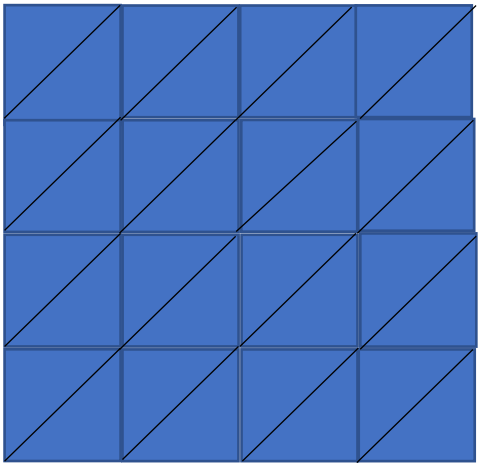

Figura 4.2: Dominio discretizado.

Una vez establecido el criterio para discretizar el dominio, se crea una base de datos en donde se van a guardar las coordenadas de cada uno de los vértices de los elementos.

Necesitamos una función de interpolación que describa el comportamiento de la ecuación en cada uno de los elementos, utilizamos la función (4.3). Debido a que la función de interpolación elegida es de tipo lineal, su derivada será constante, lo cual simplifica los cálculos.

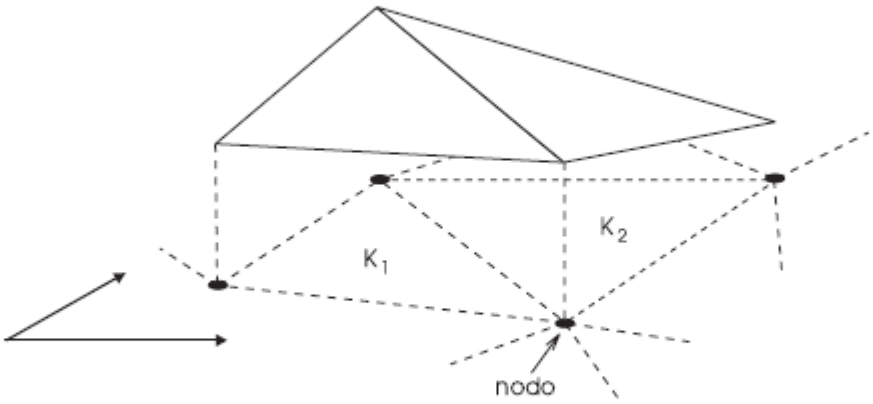

Figura 4.3: Funciones lineales aplicadas en cada elemento.

Se evalúa la función de interpolación en la base de datos, es decir, en cada uno de los nodos. El resultado serán las constantes c_i. Con una rutina que calcula la inversa de una matriz de 3x3, encontramos la forma explícita de las funciones base, dicha matriz tiene esta dimensión, ya que elegimos los elementos que contienen 3 nodos y son 3 incognitas ($c_1,\ c_2,\ y\ c_3$).

De la misma manera que en el problema unidimensional, se requiere multiplicar la ecuación diferencial por las funciones base, e integrar en cada uno de los elementos. Originalmente la expresión a trabajar es del tipo (ec. 2.6),

pero como la ecuación que se está trabajando es de dos dimensiones, al integrar, se tiene que utilizar el teorema de Green, y dado que las funciones base son del tipo lineales, se obtiene una expresión de la siguiente forma:

$$\int\int \phi_i(x,y)'\phi_j(x,y)'dydx \tag{4.7}$$

Ahora bien, al derivar las funciones base:

$$c_{2,i}c_{2,j}\int\int dydx \tag{4.8}$$

Como la geometría de los elementos es triangular, resulta que la integral de cada uno de los elementos, es en realidad el área que ocupa cada uno, esto reduce el tiempo de cálculo computacional.

El lado derecho del sistema está definido como la expresión (2.6), teniendo en cuenta que se está trabajando con una ecuación de dos variables, la expresión resultante es:

$$-\int\int \phi(x,y)q(x,y)dxdy \tag{4.9}$$

El método utilizado para resolver la expresión (4.9), es una rutina de integración numérica (Simspon 3/4). Como resultado obtenemos una matriz tridiagonal, la cual solucionaremos con una subrutina que resuelve sistemas de ecuaciones de tipo tridiagonal. Esta rutina fue seleccionada, ya que al compararla con algoritmos semejantes, como Gauss, o cálculo de la inversa, resulta óptimo, computacionalmente hablando, ya que el tiempo de ejecución se reduce drásticamente al omitir trabajar con una matriz de tipo densa, esto ya que los algoritmos como Gauss, tienen que analizar cada uno de los elementos de la matriz, y como gran parte de ellos son nulos (es decir, su valor es 0), emplea tiempo en hacer cálculos innecesarios.

El resultado será la aproximación de la solución que necesitábamos encontrar.

Capítulo 5

Resultados

Con la finalidad de respaldar la información planteada en el desarrollo del presente trabajo, se realizaron un par programas computacionales, los cuales llevaran a cabo cada uno de los procesos anteriormente definidos. Desde la división del dominio en una malla uniforme, hasta la construcción de las funciones base, para el caso de elemento finito, mientras que en diferencias finitas se crea de manera manual la matriz de rigidez y el vector de peso, con cálculos efectuados anteriormente.

Los resultados mostrados en este apartado ilustran las diferencias en los tiempos de ejecución de dichos programas, haciendo énfasis en la diferencia entre una computadora convencional y un clúster llamado KNL, situado en la Benemérita Universidad Autónoma de Puebla, para hacer dichas comparaciones en la velocidad de ejecución.

Mas adelante se detallan los resultados con las correspondientes gráficas.

5.1. Tiempos de ejecución Diferencias Finitas

En la figura (5.1) se muestran las diferencias entre los tiempos de ejecución del programa del Apéndice B. La trascendencia de los tiempos es que se ejecutan utilizando una variable de ambiente que cambia el número de hilos

con los cuales se ejecuta el programa. Para el siguiente caso se utilizaron como parámetros de entrada del programa $n = 99$ y $f = x$.

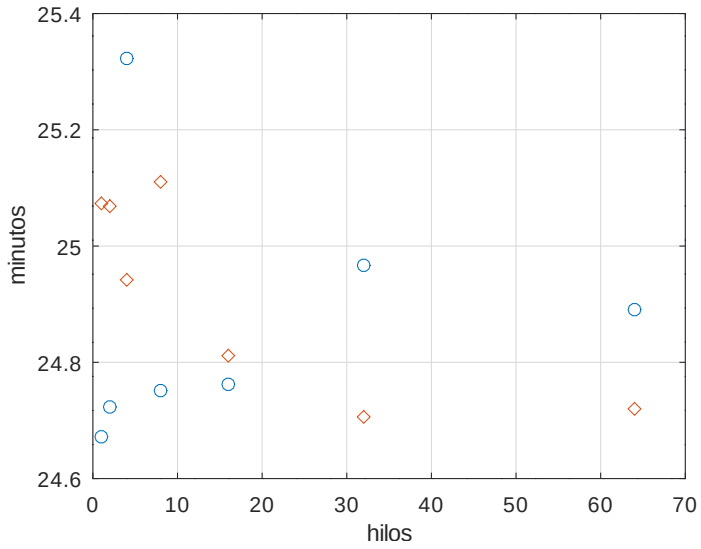

Figura 5.1: Gráfica de tiempos de ejecución del programa de Diferencias Finitas cambiando el número de hilos en cada ejecución. Donde ♢ es el tiempo con bandera de optimización y ◯ es el tiempo sin bandera.

5.2. Tiempos de ejecución Elemento Finito

En la figura (5.2) se muestran la diferencias entre los tiempos de ejecución del programa del Apéndice C. La relevancia entre los tiempos es que se ejecutan utilizando una variable de ambiente que cambia el número de hilos con los cuales se ejecuta el programa. Para el siguiente caso se utilizaron como parámetros de entrada del programa $n = 70$ y $q = 1$.

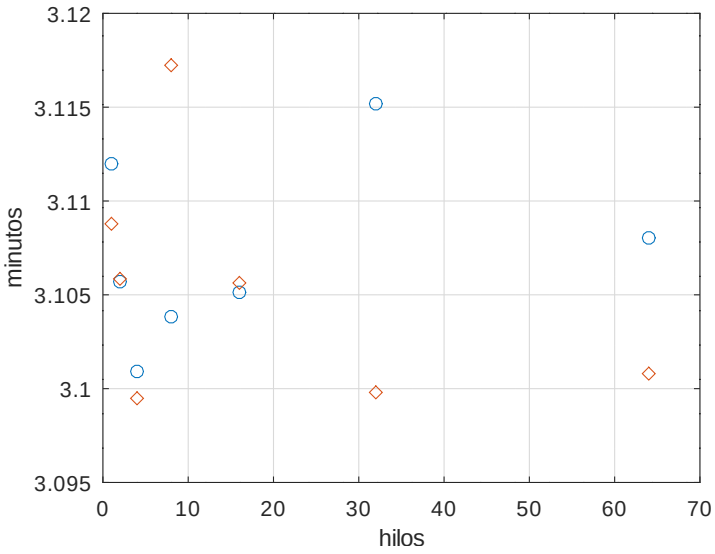

Figura 5.2: Gráfica de tiempos de ejecución del programa de Elemento Finito cambiando el número de hilos en cada ejecución. Donde ♢ es el tiempo con bandera de optimización y ◯ es el tiempo sin bandera.

5.3. Comparación de tiempos de ejecución Elemento Finito y Diferencias Finitas

La siguiente sección se enfocará en comparar los tiempos de ejecución entre el método de Diferencias Finitas y el método de Elemento Finito, utilizando el Clúster KNL de Puebla. Se utilizarán como parámetros de entrada diferentes valores de n y un valor predeterminado de 8 hilos al ejecutar el programa. En el cuadro (5.1), se muestran los tiempos de ejecución del programa que contiene el método de Diferencias Finitas.

n	tiempo
3	0.0001 min
15	0.000783 min
71	3.375716 min
142	216.737016 min

Cuadro 5.1: Comparación paralela Diferencias Finitas con 8 hilos.

En el cuadro (5.2) registran los tiempos realizados al ejecutar el programa de Elemento finito del Apéndice C. Considerando un número de hilos predeterminado de 8.

n	tiempo
2	0.00035 min
10	0.0002 min
50	0.4106 min
100	26.3 min

Cuadro 5.2: Comparación paralela Elemento Finito con 8 hilos.

De la construcción de los programas sabemos que el valor de n está directamente relacionado con el tamaño de la matriz, en la siguiente gráfica se muestran los tiempos de ejecución de ambos programas con una cantidad fija de 8 hilos y variando el valor de n, de tal manera que, el número de elementos de matriz está dado como uno de los valores de referencia de la figura (5.3), mientras que la escala vertical simboliza el tiempo, calculado en minutos.

5.4. Comparación de tiempos de ejecución en un Clúster

En las tablas del cuadro (**??**) se demuestran las diferencias en los tiempos de ejecución del programa que resuelve una ecuación diferencial por el método de Elemento Finito y Diferencias Finitas de los Apéndices B y C.

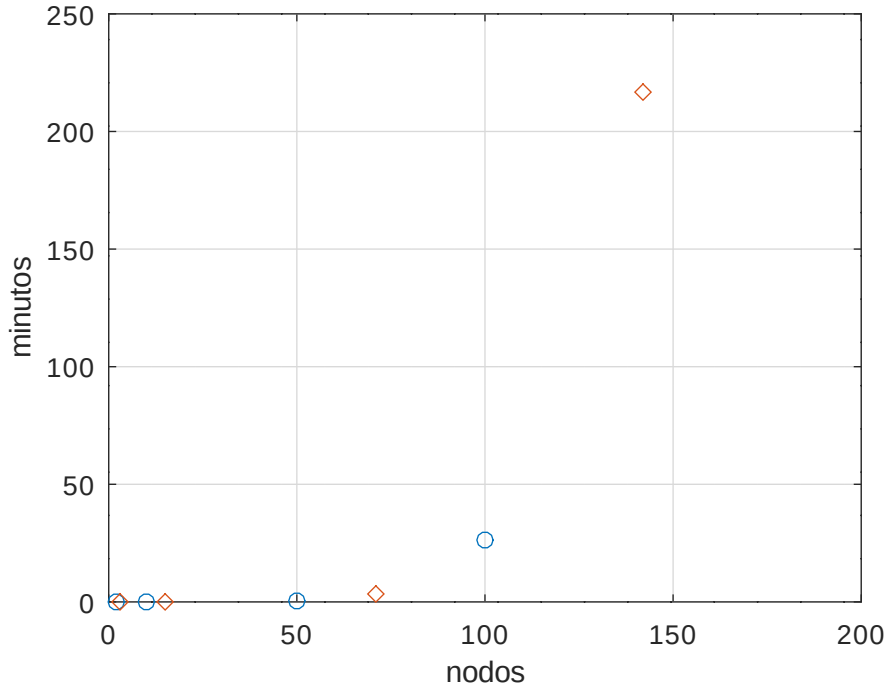

Figura 5.3: Gráfica de tiempos de ejecución del programa de Elemento Finito vs Diferencias Finitas. Donde ♢ es Diferencias Finitas y ◯ Elemento Finito.

Divisiones del mallado	Clúster KNL	Laptop Hp 6910p
n=2	0.00013 min	0.0003 min
n=10	0.00065 min	0.000583 min
n=50	3.35583 min	6.16236 min
n=100	208.782583 min	386.71245 min

Divisiones del mallado	Clúster KNL	Laptop Hp 6910p
n=3	0.000116 min	0.000083 min
n=15	0.000783 min	0.00075 min
n=71	3.3535 min	6.52026 min
n=142	220.639916 min	415.56453 min

Cuadro 5.3: Tabla de tiempos de ejecución del programa de Elemento Finito y Diferencias Finitas.

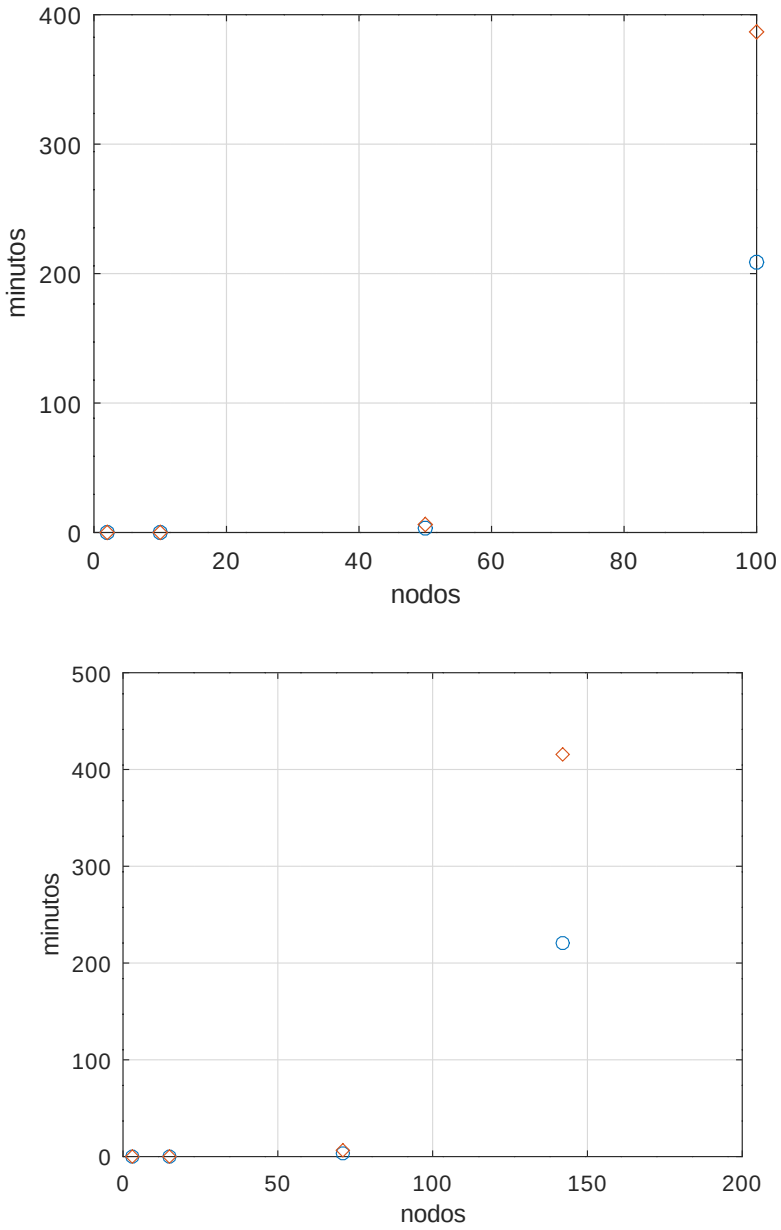

Figura 5.4: Gráfica de tiempos de ejecución del programa de Elemento Finito y Diferencias Finitas. Donde ♢ es la Laptop y ◯ es el Clúster.

Es importante resaltar el hecho de que cambiar la arquitectura de trabajo de una comptadora convecional a un Clúster tiene que repercutir directamente en los tiempos de ejecución de los programas, ya que estos tienen que reducirse drásticamente. Dicho análisis puede contemplarse en las tablas del cuadro (**??**).

En las gráficas de la figura (5.4) se demuestran las diferencias en los tiempos de ejecución del programa que resuelve una ecuación diferencial por el método de Diferencias Finitas del Apéndice B y el Método de Elemento Finito del Apéndice C.

5.5. Gráficas de resultados de programas

En este apartado se ilustran los resultados arrojados por los programas de los Apéndices B y C. Graficados con gnuplot, las gráficas corresponden a una entrada de $n = 71$, pero con las diferencias, de que en cada una de las simulaciones, la función de entrada q y f, cambian en cada ejecución.

En el panel (5.5), se muestran las soluciónes de la ecuación diferencial de Poisson en 2 dimensiones, resuelta con el método de Diferencias Finitas, con ayuda del programa del Apéndice B.

Como parámetros de entrada de utilizaron $n = 71$ y la función f se cambió en cada simulación, se utilizarón las funciones $f = 1$, $f = x$ y $f = x^2$.

En el panel (5.6) se pueden observar las soluciones de la ecuación diferencial de Poisson en 2 dimensiones, resuelta con el método de Elemento Finito, con ayuda del programa del Apéndice C.

Como parámetros de entrada de utilizaron $n = 50$ y la función f se cambió en cada simulación, se utilizarón las funciones $f = 1$, $f = x$ y $f = x^2$.

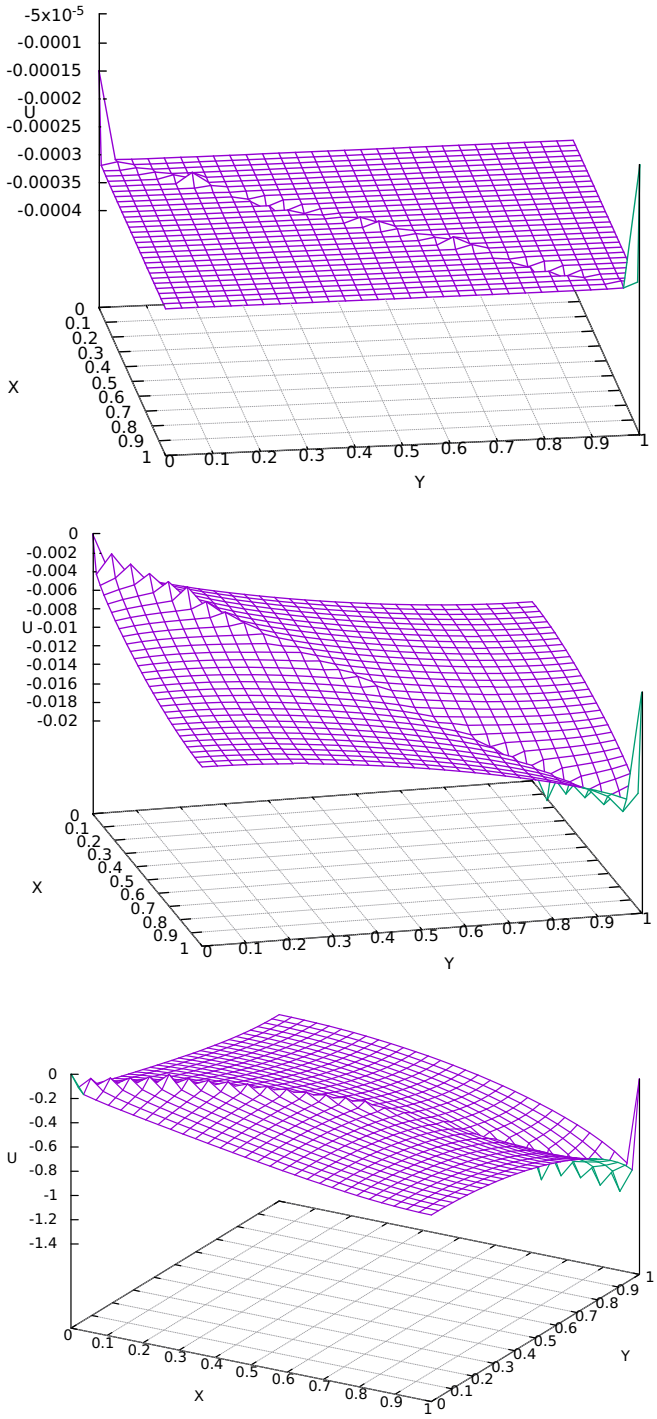

Figura 5.5: Soluciones de la Ecuación Diferencial de Poisson en 2D.

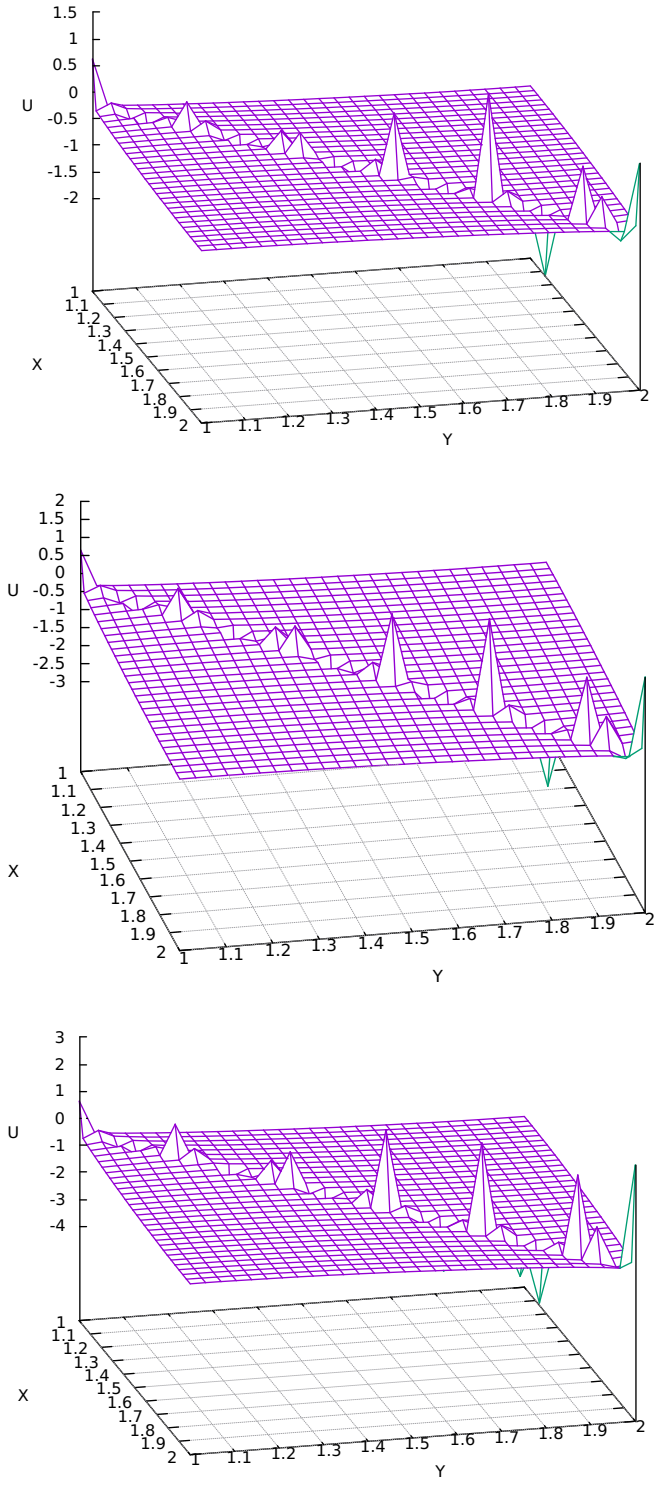

Figura 5.6: Soluciones de la Ecuación Diferencial de Poisson en 2D.

Conclusiones

- Se compararon los dos métodos numéricos más usuales de cálculo numérico para ecuaciones diferenciales parciales (EDP), tanto en desempeño como en eficacia.

- El método de Diferencias finitas al ser más sencillo y simple debido a que las derivadas en cada punto de la malla del dominio, es fácil de implementar, sobre todo en dominios regulares.

- El método de Elemento Finito al ser una aproximación con funciones base en cada elemento del dominio, permite que se pueda trabajar en dominios no regulares. Su implementación es más complicada pero la técnica es más poderosa.

- En la comparación de los tiempos de ejecución de los programas seriales entre la computadora personal y el clúster, resulta ser notoriamente más rápido en el clúster, como se puede ver el cuadro (5.3) donde se observa que para mallas grandes ($N >= 100$) hay una reducción del cerca del 50 por ciento o más en el tiempo de ejecución. La tecnología de clustering permite bajar drásticamente los tiempos de ejecución y al mismo tiempo aborda problemas más complejos de CFD y una gran diversidad de problemas de Ciencia e Ingeniería.

- La creación de los códigos en lenguaje fuente (Fortran u otro) es más eficiente realizarla desde las bases en lugar de utilizar programas realizados por otras personas o grupos, porque se comprende a profundidad el funcionamiento de los algoritmos. De esta manera se puede manipular, modificar, optimizar y saber dónde y porque puede fallar el programa con datos erróneos. Usar código abierto también es otra opción pero

sólo si se combina con buenas prácticas de programación y no se usa como caja negra.

- Un programa escrito de manera serial va a resultar ser más lento que uno escrito en paralelo, aunque hay una observación importante. Por ejemplo cuando se trabaja con un problema sencillo o pequeño (pocos datos y/o pocas instrucciones), el programa serial no gasta tiempo en la distribución de tareas ni en la comunicación entre los nodos. Por lo tanto el cómputo paralelo está orientado a cálculos más complejos o más grandes donde la distribución de tareas entre varios procesadores reduce aún más el tiempo de ejecución de la aplicación.

Bibliografía

[1] Beltrán Francisco. *Teoría General del Método de los Elementos Finitos.* Departamento de Mecánica Estructural y Construcciones Industriales - ETS Ingenieros Industriales Madrid Curso de Doctorado. 1998-99.

[2] Chapman Stephen J. *Fortran for Scientists and Engineers.* Fourth Edition. 2018. 1024 páginas.

[3] Johnson Claes. *Teoría General del Método de los Elementos Finitos.* Cambridge University Press.

[4] Louise Olsen-Kettle. *Numerical solution of partial diferential equations.* School of Earth Sciences Centre for Geoscience Computing.

[5] Alejandro T. Brewer, G. Flores Fernando. *Notas de clase de Método de los elementos finitos.* Universidad Nacional de Córdoba, Facultad de Ciencias Exactas, Físicas y Naturales. 2002

[6] Shoichiro Nakamura. *Métodos numéricos aplicados con software.* 1992

[7] Emiliano López. *Generación de mallas de elementos finitos en paralelo Universidad de Mendoza.* 2007

[8] Paul Blanchard and Glen R. Hall. *Ecuaciones diferenciales.*

[9] Dennis G. Zill and Michael R. Cullen. *Ecuaciones diferenciales con problemas con valores en la frontera.*

[10] Paul L. DeVries. *A First Course in Computational Physics.* 1994

[11] Anders Logg. *Automated Solution of Differential Equation by The Finite Element Method.* The FEniCS Book. 2011

[12] Gerardo Aragón. *Aspectos físicos y matemáticos del método del elemento finito.* Revista Mexicana de Física. 1998

[13] Guillermo Castaño Ochoa. *Aplicación del Método de los Elementos Finitos a la ecuación de Helmholtz.*

[14] Tomás Norberto Martínez Pérez. *El Método del elemento finito aplicado en la solución de problemas de Mecánica de materiales en una dimensión.* Tesis Universidad Autónoma de Nuevo León. 1998

[15] Gabriel López Garza. *Ecuaciones Diferenciales Parciales.* Tesis Universidad Autónoma Metropolitana. 2013

[16] William H. Press. *Numerical Recipes in Fortran 77.* The Art of Scientific Computing. 1992

[17] Jose Luis Gordillo Ruiz. *Algoritmo paralelo de entrenamiento de redes neuronales artificiales para sistemas de control.* Facultad de Ingeniería. 1998

[18] David Andrés Paloma Cruz. *Soluciones Clásicas a la Ecuación de Calor Lineal.* Universidad Distrital Francisco José de Caldas Bogotá D.C. 2017.

[19] Ortega J., Anguita, M., y Prieto A. *Arquitectura de Computadores.* España: Ed. Paraninfo S.A. 2005.

[20] Alberto Cardona. *Introducción al Método de los Elementos Finitos.* Centro Internacional de Métodos Computacionales en Ingeniería. 2004

[21] Foro MPI *https://www.mpi-forum.org.*

[22] Foro OpenMP *https://www.openmp.org.*

[23] Sitio Oficial MPI *https://www.open-mpi.org.*

[24] Proyecto suite de compiladores GNU *https://gcc.gnu.org*.

[25] Microprocesadores *https://www.duiops.net/hardware/micros/microshis.htm*.

Apéndice A. Diagrama de flujo de los códigos.

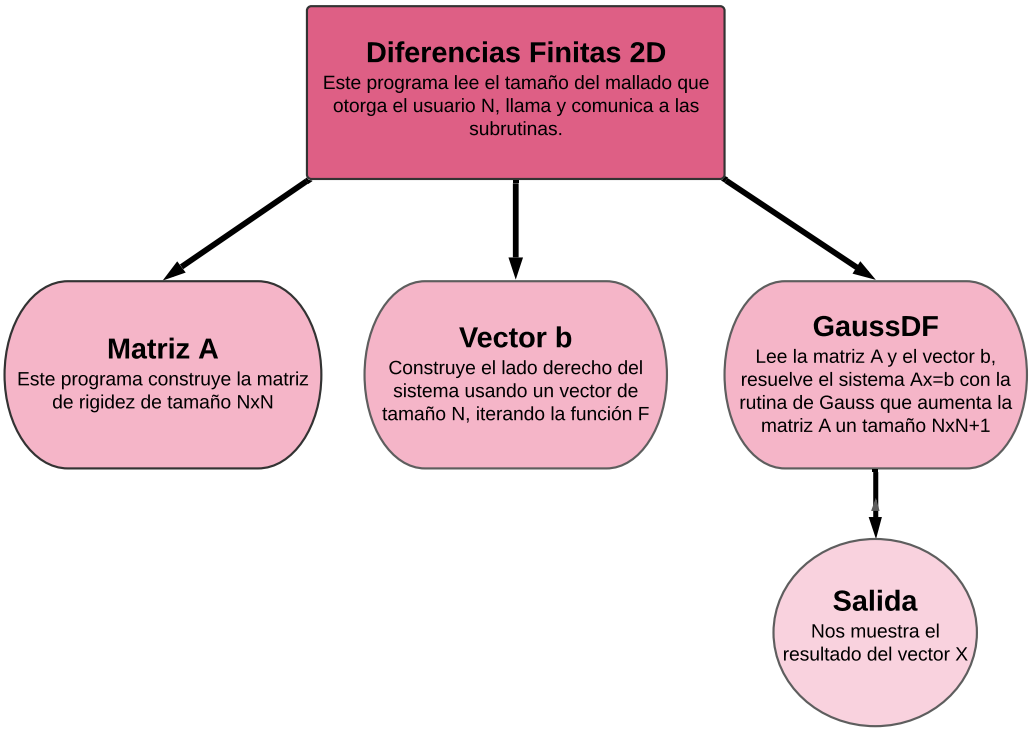

Figura 7: Diagrama de bloques Diferencias Finitas-2D.

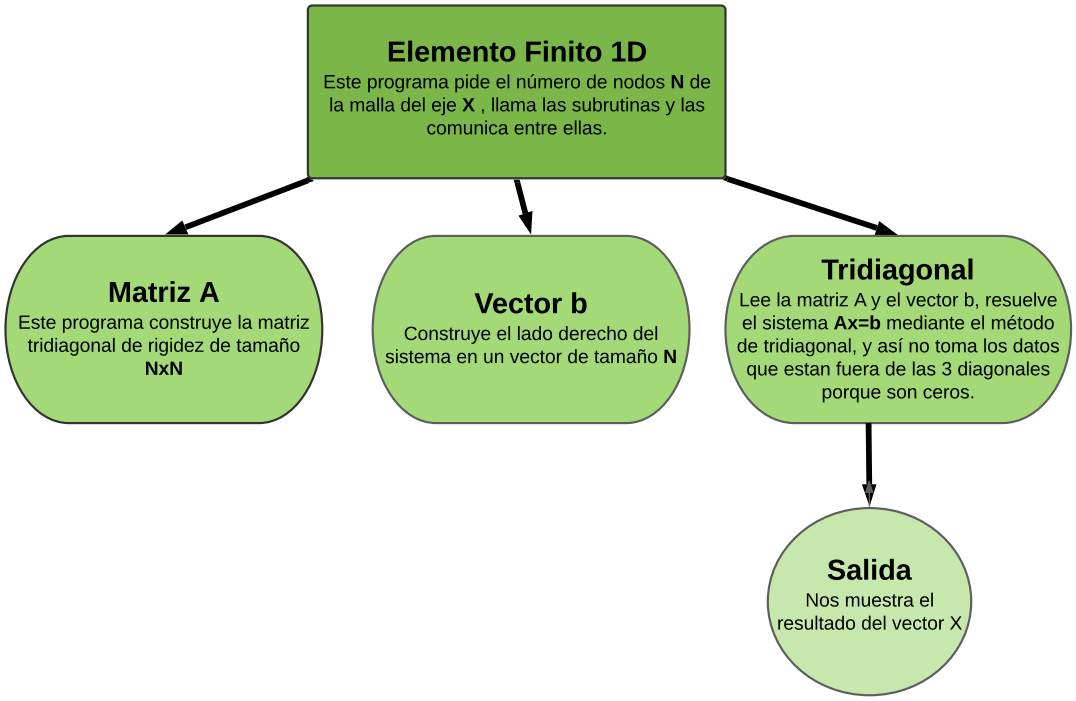

Figura 8: Diagrama de bloques Elemento Finito.

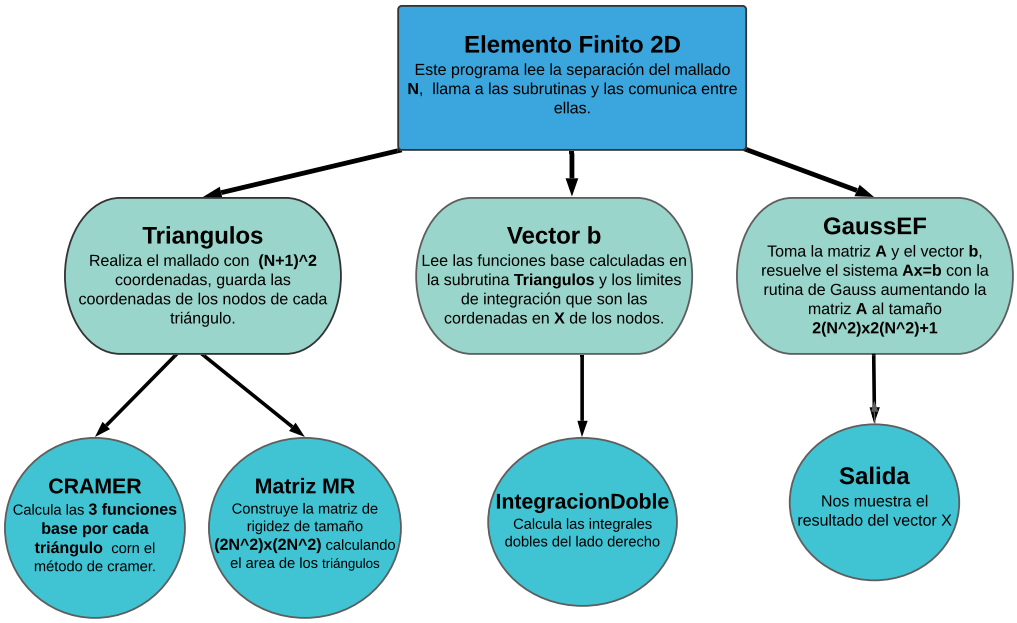

Figura 9: Diagrama de bloques Elemento Finito-2D.

Apéndice B. Código de Diferencias Finitas

```
!Autores: César Eduardo Romero Bautista y
!Javier Pérez Ramírez
!Este programa resuelve una ecuación
!diferencial con el método de
!diferencias finitas
!El programa principal es el que nos
!ayuda a llamar y unir todos los programas
PROGRAM diferencias_f
USE estructura
implicit none
integer::t,tI

print*, 'En cuantos subintervalos quieres dividir la region?'
Read(*,*) t

tI=t*t

CALL matrizDF (t,A)
CALL vecBD(tI,t,b)
CALL gaussEF(tI,b,A,x)


END PROGRAM

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
MODULE estructura
implicit none
Integer :: i,j
real::h
real,allocatable:: b(:)
real,allocatable::A(:,:)
real, allocatable:: x(:)


CONTAINS
SUBROUTINE MATRIZDF(t,A)
implicit none
INTEGER, INTENT(IN) :: t
integer::n,i,j
real::h
real, allocatable,intent(out):: A(:,:)

!INICIALIZAMOS LA MATRIZ EN ZEROS
N=t*t
h=1/t+2
Allocate(A(n,n),x(n),b(n))
A=0.0

DO I=1,N
A(I,I)=-4
END DO

DO I=1,N-1
 A(I,I+1)=1
 A(I+1,I)=1
END DO

DO I=3,N-1,3
 A(I,I+1)=0
 A(I+1,I)=0
END DO
```

```
DO I=1,N-3
 A(I,I+3)=1
 A(I+3,I)=1
END DO


 END SUBROUTINE

end module estructura

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

SUBROUTINE vecBD(tI,t,b)
INTEGER, INTENT(IN) :: tI,t
integer::n,i
real::h,k
real, dimension(tI):: b

h=1./(t+2)

do i=1,tI
  b(i)=(h*h)*(i/(t+2.))      !f=x
 !b(i)=(h*h)*(i/t+2)*(i/t+2)  !f=x²
 !b(i)=(h*h)*(sin(i/t+2))     !f=sin(x)
end do
end subroutine

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine gaussEF(tI,b,A,x)
implicit none
integer,intent(in) :: tI
real,dimension(tI),intent(in) :: b
real,dimension(tI,tI),intent(in) :: A
real,dimension(tI+1,tI)::AI
real,dimension(tI),intent(out) :: x
real,dimension(tI+1)::P
```

```
real,dimension(tI)::vecX
real :: m,z,aux2,aux,YY
integer:: i,n,j,k
n=tI
vecX=0.
do i=1,n
 do j=1,n
  AI(i,j)=A(i,j)
  AI(n+1,i)=b(i)
 end do
end do
DO j=1,N-1
 aux=AI(j,j)
 DO k=1,N+1,1
  P(k)=AI(k,j)/aux
 END DO
 DO i=j+1,N,1
  aux2=AI(j,i)
  DO k=1,N+1,1
   AI(k,i)=AI(k,i)-P(k)*aux2
  END DO
 END DO
END DO
DO I=1,N,1
 x(I)=0
END DO
 x(N)=AI(N+1,N)/AI(N,N)
DO I=N-1,1,-1
 Z=0;
 DO K=1,N,1
  Z=Z+x(K)*AI(k,I)
 END DO
 x(I)=(AI(N+1,I)-Z)/AI(I,I)
END DO
do i=1,n,1
 YY=i
 vecX(i)=YY/n
end do
```

```
DO I=1,N,1
 write(*,*) vecX(i),vecX(i), x(i)
END DO
END subroutine
```

Apéndice C. Códigos Elemento Finito

```
!Autores: César Eduardo Romero Bautista y
!Javier Pérez Ramírez
!ESTE ES EL PROGRAMA PRINCIPAL QUE RESUELVE
!UNA ECUACIÓN DIFERENCIAL UNIDIMENCIONAL CON
!EL MÉTODO DE ELEMENTO FINITO
PROGRAM ELEFIN
implicit none
integer, parameter :: N=10
INTEGER :: I,J
REAL :: H,B1,A1,X0,CP,DP,M
REAL*8, DIMENSION (N):: L,B,D,X
REAL*8, DIMENSION (N-1):: C,S
REAL*8, DIMENSION (N,N):: A
EXTERNAL MATRIZA
EXTERNAL vectorb
EXTERNAL trig

PRINT*, 'DAME EL VALOR DE LA DIAGONAL PRINCIPAL'
READ(*,*) L

h=1./N
Print*, H

L=(1./h)*L
print*, "EL VECTOR L"
```

```
PRINT*, L

PRINT*, 'DAME EL VALOR DE LA DIAGONAL SUPERIOR E INFERIOR'
READ(*,*) S
S=(1./h)*S
print*,"EL VECTOR S"
PRINT*, S

C=S
PRINT*,"EL VECTOR C"
PRINT*,C

X0=0

CALL MATRIZA(L,S,C,N,A)
CALL vectorb(X,X0,h,b,n)

Print*,"Valor de B"
print*, B

CALL solve_tridiag(C,L,S,B,D,N)

print*,"EL VECTOR SOLUCION D"
DO i=1,n
print*,D(i)
END DO

END PROGRAM ELEFIN

!SUBRUTINA QUE LEE LA MATRIZ DE RIGIDEZ
!GUARDAREMOS LAS DIAGONALES EN VECTORES
!DONDE L ES LA DIAGONAL
!S LA DIAGONAL SUPERIOR
!C LA DIAGONAL INFERIOR

SUBROUTINE MATRIZA(L,S,C,N,A)
INTEGER :: N
REAL*8, DIMENSION (N), INTENT(IN) :: L
```

```
REAL*8, DIMENSION (N-1), INTENT(IN) :: S,C
REAL*8, DIMENSION (N,N), INTENT(OUT) :: A
REAL*8 :: h,B1,A1
!INICIALIZAMOS LA MATRIZ EN CEROS

A=0.0
A1=0
B1=1
h=(B1-A1)/N


DO I=1,N
A(I,I)=L(I)
END DO

DO I=1,N-1
A(I,I+1)=S(I)
A(I+1,I)=C(I)
END DO

A=(1/h)*A

DO I=1,N
PRINT*,( A(I,J), j=1,n)
END DO


END SUBROUTINE


!Esta subrutina contruye el vector de carga B
!implicit none
subroutine vectorb(X,X0,h,b,n)
integer::i,j
real::h,B1,A1
integer ::n,X0
real,dimension(n) ::X
real,dimension(n) ::b
```

```
A1=0.
B1=1.
h=(B1-A1)/N


do i=1,n
X(i)=X0+(i*h)
b(i)=2*(1+(X(i)*h))
end do

b=10.*b

end subroutine



subroutine solve_tridiag(C,L,S,B,D,N)
implicit none
!C - diagonal inferior
!L - diagonal principal
!S - diagonal superior
!B - lado derecho
!D - respuesta
!N - número de ecuaciones

        integer,intent(in) :: n
        real*8,dimension(n),intent(in) :: C,L,S,B
        real*8,dimension(n),intent(out) :: D
        real*8,dimension(n) :: cp,dp
        real*8 :: m
        integer i
!Inicializamos cp y dp
        cp(1) = S(1)/L(1)
        dp(1) = B(1)/L(1)
!Solución de los vectores cp y dp
         do i = 2,n
```

```
           m = L(i)-cp(i-1)*C(i)
           cp(i) = S(i)/m
           dp(i) = (B(i)-dp(i-1)*C(i))/m
         enddo
!Inicializamos D
         D(n) = dp(n)
!Solución para D de los vectores cp y dp
        do i = n-1, 1, -1
          D(i) = dp(i)-cp(i)*D(i+1)
        end do

end subroutine solve_tridiag
```

```
!Autores: César Eduardo Romero Bautista y
!Javier Pérez Ramírez
!Programa que utiliza el método de elemento
!finito para resolver una ecuación
!diferencial en dos dimensiones
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!El programa principal es el que nos ayuda
!a llamar y unir todos los programas
PROGRAM elemento_f
USE estructura
implicit none
integer::n,tI

print*, 'En cuantos subintervalos quieres dividir la region?'
Read(*,*) n
tI=(n*n)*2


CALL triangulos (n,TrianguloI,MR,CCC,LimitI,LimitII,LimitIII)
CALL vecB(CCC,n,tI,LimitI,LimitII,LimitIII,VBI)
CALL gaussEF(tI,VBI,MR,x)


deallocate(TrianguloI,CCC,MR,LIMITI,LIMITII,LIMITIII)

END PROGRAM

!El Module nos ayuda a declarar todas las variables que vamos
!a utilizar en todo el programa
MODULE estructura
implicit none
Integer :: i,k,j,L,t,M,ii
Real :: delt_x,delt_y, DET,CC,a,b,c,d,e,f
real, dimension (3,1) :: u
REAL, dimension (3,3,3) :: EE
REAL, dimension (3,3) :: AA
Real, dimension (3,1) :: R
Real, dimension (3) :: RR
```

```
real,allocatable:: MR(:,:)
real,allocatable:: CCC(:,:)
real,allocatable:: VB(:),VBI(:)
real, allocatable:: limitI(:)
real, allocatable:: limitII(:)
real, allocatable:: limitIII(:)
real, allocatable:: x(:)

!El type nos ayuda a definir la variable triangulo
!con diferente tipo de datos:
!   -Numero de triangulo
!   -Numero de nodo
!   -Cordenadas (x,y)
TYPE triangulo
INTEGER :: num_trig, num_nod ! número del nodo
REAL :: x,y
END TYPE triangulo
type(triangulo),target, ALLOCATABLE :: trianguloI(:,:)

CONTAINS
!En esta subrutina se realiza el calculo de las cordenadas
!de cada triangulo, dado que la division de los nodos
!depende del tamaño que quiera el usuario (n)
!y tambien depende del numero de triangulos, Por ejemplo
!si el usuario quiere 2 divisiones, el numero de nodos
!seria 9 y el numero de triangulos seria 8
!En pocas palabras este programa realiza el mallado del dominio.
SUBROUTINE Triangulos(n,trianguloI,MR,CCC,LimitI,LimitII,LimitIII)
integer,intent (in)::n
integer::t
real, allocatable,intent(out):: CCC(:,:)
real, allocatable,intent(out):: MR(:,:)
real, allocatable,intent(out):: LimitI(:)
real, allocatable,intent(out):: LimitII(:)
real, allocatable,intent(out):: LimitIII(:)
type(triangulo),target, ALLOCATABLE :: trianguloI(:,:)

delt_x=1./n
```

```
delt_y=1./n
L=(n+1)*(n+1)!Numeros de nodos
t=(n*n)*2!Numero de triangulos

!Le damos el tamaño de las variables, dado que
!las declaramos dinamicas, porque pueden variar su tamaño
ALLOCATE (trianguloI(t,3),CCC&
(t,3),MR(t,t),vb(t),VBI(t),LimitI(t)&
,LimitII(t),LimitIII(t),x(t))
vb=0
do k=0,n-1
!Declaramos los dos primeros triangulos (El que está
!por encima y el que está por debajo) después de esto
!solamente es iterativo encontrar
!los siguientes triangulos
 trianguloI(1+(k*(2*n)),1)%num_trig=1+(k*(2*n))
 trianguloI(1+(k*(2*n)),1)%num_nod=1
 trianguloI(1+(k*(2*n)),1)%x=1
 trianguloI(1+(k*(2*n)),1)%y=1+(k*delt_y)

 trianguloI(1+(k*(2*n)),2)%num_trig=1+(k*(2*n))
 trianguloI(1+(k*(2*n)),2)%num_nod=2
 trianguloI(1+(k*(2*n)),2)%x=1+delt_x
 trianguloI(1+(k*(2*n)),2)%y=1+(k*delt_y)

 trianguloI(1+(k*(2*n)),3)%num_trig=1+(k*(2*n))
 trianguloI(1+(k*(2*n)),3)%num_nod=3
 trianguloI(1+(k*(2*n)),3)%x=1
 trianguloI(1+(k*(2*n)),3)%y=1+delt_y+(k*delt_y)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

 trianguloI(2+(k*(2*n)),1)%num_trig=2+(k*(2*n))
 trianguloI(2+(k*(2*n)),1)%num_nod=1
 trianguloI(2+(k*(2*n)),1)%x=1+delt_x
 trianguloI(2+(k*(2*n)),1)%y=1+(k*delt_y)

 trianguloI(2+(k*(2*n)),2)%num_trig=2+(k*(2*n))
```

```
 trianguloI(2+(k*(2*n)),2)%num_nod=2
 trianguloI(2+(k*(2*n)),2)%x=1+delt_x
 trianguloI(2+(k*(2*n)),2)%y=1+delt_y+(k*delt_y)

 trianguloI(2+(k*(2*n)),3)%num_trig=2+(k*(2*n))
 trianguloI(2+(k*(2*n)),3)%num_nod=3
 trianguloI(2+(k*(2*n)),3)%x=1
 trianguloI(2+(k*(2*n)),3)%y=1+delt_y+(k*delt_y)


 do i=3,2*n,2
  do j=1,3
   trianguloI(i+(k*(2*n)),j)%num_trig=i+(k*(2*n))
   trianguloI(i+(k*(2*n)),j)%num_nod=j
   trianguloI(i+(k*(2*n)),j)%x=trianguloI(i-2,j)%x+delt_x
   trianguloI(i+(k*(2*n)),j)%y=trianguloI(i-2,j)%y+(k*delt_y)
  end do
 end do

 do i=4,2*n,2
  do j=1,3
   trianguloI(i+(k*(2*n)),j)%num_trig=i+(k*(2*n))
   trianguloI(i+(k*(2*n)),j)%num_nod=j
   trianguloI(i+(k*(2*n)),j)%x=trianguloI(i-2,j)%x+delt_y
   trianguloI(i+(k*(2*n)),j)%y=trianguloI(i-2,j)%y+(k*delt_y)
  end do
 end do
end do
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Guardamos las variables de las cordenadas porque nos
!ayudararan a calcular el valor de las C's, debemos
!despejar las C's de las funciones
!U=C1+C2X+C3Y , Donde X y Y son las cordenadas de cada nodo.
!mediante el metodo de Cramer obtenemos el valor de cada C's,
!esto se realiza para cada cordenada
do ii=1,t

 a=TrianguloI(ii,1)%x
```

```
 b=TrianguloI(ii,1)%y
 c=TrianguloI(ii,2)%x
 d=TrianguloI(ii,2)%y
 e=TrianguloI(ii,3)%x
 f=TrianguloI(ii,3)%y
 limitI(ii)=a!-1
 limitII(ii)=c!-1
 limitIII(II)=e!-1
 call CRAMER (a,b,c,d,e,f,RR)

!Guardamos los valores de las C's en la matriz
!CCC que tiene tamaño (numero de nodos,3)
 CCC(ii,1)=RR(1)
 CCC(ii,2)=RR(2)
 CCC(ii,3)=RR(3)

end do

!Matriz de rigidez
!La costrucción de la matriz de rigidez se realizó de
!la siguiente manera y empalmamos los triangulos, porque
!en nuestro problema principal no se intersectan.
!Nos Daría una matriz diagonal, lo cual sería trivial.

 do i=1,t
  MR(i,i)=((CCC(i,2)*CCC(i,2))+(CCC(i,3)*CCC(i,3)))&
*(1./2.)*(1./n)*(1./n)
 end do
 do i=2,t
  MR(i-1,i)=((CCC(i,2)*CCC(i+1,2))+(CCC(i,3)*CCC&
(i+1,3)))*(1./2.)*(1./n)*(1./n)!CCC(i,2)*CCC(i+1,2)&
MR(i,i-1)=((CCC(i,2)*CCC(i-1,2))+(CCC(i,3)*&
CCC(i-1,3)))*(1./2.)*(1./n)*(1./n)!CCC(i,2)*CCC(i-1,2)
 end do

end SUBROUTINE triangulos

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
!Esta subrutina hace el cálculo de las C's por el
!método de Cramer, para ahorrarnos tiempo y trabajo
!computacional lo hicimos para una matriz de 3X3 manualmente.
subroutine CRAMER (a,b,c,d,e,f,RR)
real,intent(in)::a,b,c,d,e,f
real::DETX,DETY,DETZ
integer:: NN
real,dimension(3)::U
Real, dimension (3),intent (out)::RR
Real,dimension (3,3) :: cof

 M=0
 AA(1,1)=1
 AA(1,2)=a
 AA(1,3)=b
 AA(2,1)=1
 AA(2,2)=c
 AA(2,3)=d
 AA(3,1)=1
 AA(3,2)=e
 AA(3,3)=f
NN=3
do i=1,NN
 U(i)=i
end do


DET=((AA(1,1)*AA(2,2)*AA(3,3))+(AA(2,1)*AA(3,2)*AA(1,3))&
+(AA(3,1)*AA(1,2)*AA(2,3)))-((AA(1,3)*AA(2,2)&
*AA(3,1))+(AA(2,3)*AA(3,2)*AA(1,1))+(AA(3,3)*AA(1,2)*AA(2,1)))



if (DET .NE. 0) then


DETX=((U(1)*AA(2,2)*AA(3,3))+(AA(1,2)*AA(2,3)*U(3))&
```

```
+(AA(1,3)*U(2)*AA(3,2)))-((AA(1,2)*U(2)*AA(3,3))&
+(U(1)*AA(2,3)*AA(3,2))+(AA(1,3)*AA(2,2)*U(3)))
DETY=((AA(1,1)*U(2)*AA(3,3))+(AA(2,1)*U(3)*AA(1,3))&
+(AA(3,1)*U(1)*AA(2,3)))-((AA(1,3)*U(2)*AA(3,1))&
+(AA(2,3)*U(3)*AA(1,1))+(AA(3,3)*U(1)*AA(2,1)))
DETZ=((AA(1,1)*AA(2,2)*U(3))+(AA(2,1)*AA(3,2)*U(1))&
+(AA(3,1)*AA(1,2)*U(2)))-((U(1)*AA(2,2)*AA(3,1))&
+(U(2)*AA(3,2)*AA(1,1))+(U(3)*AA(1,2)*AA(2,1)))


RR(1)=DETX/DET
RR(2)=DETY/DET
RR(3)=DETZ/DET


else
  print*, 'La matriz es singular'
end if

END subroutine CRAMER


end module estructura


!INTEGRACION DOBLE MEDIANTE
!LA REGLA DE SIMPSON
!Este programa nos ayuda a calcular la integral del
!lado derecho de nuestro sistema y así encontrar el vector b

subroutine vecB(CCC,n,tI,LimitI,LimitII,LimitIII,VBI)
implicit none
integer::i,j!,t
integer,intent(in)::n,tI
real,intent(in),dimension(tI,3):: CCC
real,intent(in),dimension(tI):: LimitI,LimitII,LimitIII
real,dimension(tI)::VB
real,intent(out),dimension(tI)::VBI
```

```
real::CS,CSS,CSSS,LMINF,LMSUP,TT

!Parece que realizamos las mismas isntrucciones dos
!veces, pero se realizo de esta forma dado que se
!hace secuencialmente, la primera parte lo realiza
!para los triangulos parados y la segunda lo realiza
!para los que están de cabeza.

VBI=0
do i=1,tI,2
 CS=CCC(i,1)! c1
 CSS=CCC(i,2) !C2
 CSSS=CCC(i,3) !C3
 LMINF=LimitI(i) !LIMITE INFERIOR DE LA INTEGRACION DE LA X
 LMSUP=LimitII(i)!LIMITE SUPERIOR DE LA INTEGRACION DE LA X
 call integraciondoble(CS,CSS,CSSS,LMINF,LMSUP,TT)
 VB(i)=TT
end do
do j=2,tI,2
 CS=CCC(j,1)
 CSS=CCC(j,2)
 CSSS=CCC(j,3)
 LMINF=LimitIII(j)
 LMSUP=LimitII(j)
 call integraciondobleI(CS,CSS,CSSS,LMINF,LMSUP,TT)
 VB(j)=TT
end do
VBI=VB



end SUBROUTINE

SUBROUTINE integraciondoble(CS,CSS,CSSS,LMINF,LMSUP,TT)
implicit none
real,intent(in)::CS,CSS,CSSS,LMINF,LMSUP
real,intent(out)::TT
INTEGER::M,N,I,J
```

```
REAL::HX,HY,S,W,C,D,F,X,Y

M=20

N=20

HX=(LMSUP-LMINF) /M
TT=0
DO I=0, M
 X=LMINF+I*HX
!Encuentra el límite inferior de los valores de y
 CALL FUNCT1 (LMINF,C, X)
!Encuentra el límite superior de los valores de y
 CALL FUNCT2 (LMSUP,D, X)
 HY=(D-C)/N !Tamaño del intervalo en la dirección de y
 s=0
 DO J=0,N
 Y=C+J*HY !Valor en y de los puntos de la retícula
CALL FUNCT3 (CS,CSS,CSSS,F, X, Y)
!Encuentra el valor de
 W=4.
 IF (INT(J/2)*2.EQ.J) W=2.
 IF((J.EQ.0).OR.(J.EQ.N)) W=1.
 S=S+W*F !Integración en la dirección de y
END DO
S=S*HY/3.
W=4.
IF (INT(I/2)*2 .EQ.I) W=2.
IF((I.EQ.0).OR.(I.EQ.M)) w=1.

TT=TT+W*S    !Integración en la dirección de x
END DO
TT=TT*HX/3.  !Resultado de la integración


END subroutine

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
!Subrutina para los triangulos acostados
SUBROUTINE integraciondobleI(CS,CSS,CSSS,LMINF,LMSUP,TT)
real,intent(in)::CS,CSS,CSSS,LMINF,LMSUP
real,intent(out)::TT
INTEGER::M,N,I,J
REAL::HX,HY,S,W,C,D,F,X,Y


M=20

N=20


HX=(LMSUP-LMINF) /M
TT=0
DO I=0, M
 X=LMINF+I*HX
!Encuentra el limite superior de los valores de y
 CALL FUNCT1 (LMINF,C, X)
!Encuentra el limite inferior de los valores de y
 CALL FUNCT2 (LMSUP,D, X)
 HY=(C-D)/N !Tamaño del intervalo en la dirección de y
 s=0
 DO J=0,N
 Y=D+J*HY !Valor en y de los puntos de la reticula
 CALL FUNCT3 (CS,CSS,CSSS,F, X, Y)!Encuentra el valor de
 W=4
 IF (INT(J/2)*2.EQ.J) W=2
 IF((J.EQ.0).OR.(J.EQ.N)) W=1
 S=S+W*F !!Integracion en la dirección de y
END DO
S=S*HY/3
W=4
IF (INT(I/2)*2 .EQ.I) W=2
IF((I.EQ.0).OR.(I.EQ.M)) w=1

TT=TT+W*S    !Integracion en la dirección de
END DO
```

```
TT=TT*HX/3   !Resultado de la integración


END subroutine

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
SUBROUTINE FUNCT1 (LMINF,C, X) !Curva limite inferior
  REAL :: LMINF
 C= LMINF
END
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
SUBROUTINE FUNCT2 (LMSUP,D, X) !Curva limite superior
 REAL :: LMSUP
 D= LMSUP-X
END
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Evalúa la función a integrar
SUBROUTINE FUNCT3 (CS,CSS,CSSS,F, X, Y)
!F=0
F=CS+CSS*X+CSSS*Y !q=1
!F=CS*X+CSS*X*X+CSSS*X*Y !q=X
! F=CS*X*X+CSS*X*X*X+CSSS*X*X*Y !q=X²
END

!Esta subrutina calcula Nuestra solución por el
!método de Gauss Ax=b
subroutine gaussEF(tI,VBI,MR,x)
implicit none
integer,intent(in) :: tI
real,dimension(tI),intent(in) :: VBI
real,dimension(tI,tI),intent(in) :: MR
real,dimension(tI+1,tI)::A
real,dimension(tI),intent(out) :: x
real,dimension(tI+1)::P
real :: m,z,aux2,aux
integer i,n,j,k
n=tI
do i=1,n
```

```
 do j=1,n
  A(i,j)=MR(i,j)
  A(n+1,i)=VBI(i)
 end do
end do
DO j=1,N-1
 aux=A(j,j)
 DO k=1,N+1,1
  P(k)=A(k,j)/aux
 END DO
 DO i=j+1,N,1
  aux2=A(j,i)
  DO k=1,N+1,1
   A(k,i)=A(k,i)-P(k)*aux2
  END DO
 END DO
END DO
DO I=1,N,1
 x(I)=0
END DO
 x(N)=A(N+1,N)/A(N,N)
DO I=N-1,1,-1
 Z=0;
 DO K=1,N,1
  Z=Z+x(K)*A(k,I)
 END DO
 x(I)=(A(N+1,I)-Z)/A(I,I)
END DO
DO I=1,N,1
 write(*,*) i, x(i)
END DO
END subroutine
```

Apéndice D. Características del Hardware de Laptop

Laptop compaq6910p	
Procesador	Intel $Core^{TM}$2 Duo Processor T8100
Memoria RAM	2GB std
Disco Duro	120GB Serial ATA
Chipset	Mobile Intel GL40 Express
Gráficas	Mobile Intel Graphics Media Accelerator GMA X3100
Bateria	6 celdas

Figura 10: Características Laptop Hp 6910p