UNIVERSIDAD DE CHILE

FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

DEPARTAMENTO DE INGENIERÍA MECÁNICA

PARALELIZACIÓN DE ALGORITMO NUMÉRICO PARA RESOLUCIÓN DE PROBLEMAS EN MECÁNICA DE SÓLIDOS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MECÁNICO

RUBÉN DARÍO TORRES VERDUGO

PROFESOR GUÍA:
ALEJANDRO ORTIZ BERNARDIN

MIEMBROS DE LA COMISIÓN:
ROGER BUSTAMANTE PLAZA
VIVIANA MERUANE NARANJO

SANTIAGO DE CHILE
2016

RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE: Ingeniero Civil Mecánico.
POR: Rubén Darío Torres Verdugo
FECHA: 28/01/2016
PROFESOR GUÍA: Alejandro Ortiz Bernardin

PARALELIZACIÓN DE ALGORTIMO NUMÉRICO PARA LA RESOLUCIÓN DE PROBLEMA EN LA MECÁNICA DE SÓLIDOS.

Dentro de los métodos de simulación numérica para resolución de ecuaciones diferenciales parciales (EDP's), los métodos sin malla han sido desarrollados desde hace aproximadamente veinte años. A diferencia del método de elementos finitos, los métodos sin malla no necesitan una malla definida para la construcción de sus funciones de base, las que se crean únicamente por la distribución de los nodos en el dominio. Esta característica hace a este método más atractivo en problemas donde exista distorsión de la malla, sin embargo, requieren un tiempo extra en su cálculo y un esfuerzo mayor en su programación.

El presente trabajo tiene por objetivo realizar un algoritmo numérico eficiente mediante programación en paralelo, para la resolución de problemas en la mecánica de sólidos mediante el método sin malla Galerkiano con funciones de base de máxima entropía. La motivación de este trabajo es suplir uno de los principales defectos de los métodos sin malla, ser computacionalmente costosos.

Son abordados los antecedentes del método sin malla, elasticidad lineal y la programación en paralelo. Se utiliza el método sin malla Galerkiano basado en la forma débil, con funciones de base de máxima entropía.

Se trabaja con el software computacional MATLAB y la librería Parallel Computing Toolbox en la implementación de la programación en paralelo para tres problemas específicos y se analiza el error numérico, convergencia, tiempo de cómputo, e indicadores de desempeño para la programación en paralelo, como lo son Speedup y Eficiencia paralela.

Se obtienen errores numéricos aceptables, entregados por las normas relativas L^2 y H^1, obteniendo convergencia en los tres problemas. Los tiempos de cómputo se reducen al implementar la programación paralela en todos los casos. La convergencia del problema es independiente del número de procesadores utilizados. Se obtienen los mejores resultados de Speedup y Eficiencia paralela para problemas por sobre los 5000 grados de libertad. Se recomienda trabajar problemas por sobre esta cifra en la implementación de la programación en paralelo para la resolución de problemas en la mecánica de sólidos mediante el método sin malla Galerkiano.

Se cumple objetivo principal, logrando realizar algoritmo numérico mediante programación en paralelo, para la resolución de problemas en la mecánica de sólidos mediante el método sin malla Galerkiano con funciones de base de máxima entropía.

Dedicado a mi familia

AGRADECIMIENTOS

Quiero agradecer muy especialmente a mi familia, que siempre ha estado allí y viviendo junto conmigo esta etapa que estoy superando.

A ti papá, que me has ayudado en todo momento cuando lo he necesitado, gracias por escucharme y guiarme, por tener que levantarte temprano para dejarme donde sea que fuese, y por cuidarnos.

A ti mamá, por cuidarme con mucha paciencia (sobre todo al principio de mi vida, jaja), por amarme tanto y hacerme muy feliz siempre, por enseñarme a ser quien soy hoy y dedicar toda mi vida para que esto sucediera, sin ti mamá esto no hubiera sido posible en ninguna medida.

A ti Diego, por ser más que un hermano, por estar siempre, por compartir todo, por reír, por disfrutar tan especialmente cada momento que hemos compartido.

A mi tía Sixtina, por ser tan buena persona, tan paciente, por entregar mucho a la familia, me ha enseñado mucho sin darse cuenta.

A mis perritos, que los extraño en el alma no tenerlos conmigo siempre, me han ayudado a aliviar cada dolor que he tenido en toda esta aventura.

A mis amigos (que me enorgullece decir que son muchos y no caben todos acá), gracias a todos ustedes por hacer de este camino más entretenido y agradable. Y finalmente a Claudia, has sido sin duda lo mejor de este ciclo, me has enseñado y ayudado a vivir mejor.

Tabla de Contenido

1 Introducción

Los métodos sin malla para resolver ecuaciones diferenciales parciales (en adelante EDP) han sido desarrollados hace aproximadamente dos décadas. A diferencia del método de elementos finitos, los métodos sin malla no necesitan una malla definida para la construcción de sus funciones de base, las que se crean únicamente por la distribución de los nodos en el dominio. Esta característica hace a este método más atractivo en problemas donde exista distorsión de la malla, sin embargo, requieren un tiempo extra en su cálculo y un esfuerzo mayor en su programación. Aquí el método sin malla Galerkiano basado en la forma débil [1][2] es adoptado, siendo el método sin malla más comúnmente usado.

Para resolver problemas que exigen gran capacidad de cálculo es necesario computadores con suficiente memoria y gran poder computacional. Programar en serie podría no solucionar estos problemas debido al límite de memoria. La programación en paralelo ha sido usada para resolver este tipo de problemas, trabajando en la estrategia "Divide y conquista" [3]. Usando esta estrategia, las tareas son divididas en otras más pequeñas y manejables, las que luego son asignadas a computadores multinúcleos.

Debido a que los métodos sin malla son computacionalmente costosos [4], el objetivo del presente trabajo es realizar un algoritmo numérico eficiente mediante programación en paralelo, para la resolución de problemas en la mecánica de sólidos mediante el método sin malla Galerkiano con funciones de base de máxima entropía. Para ello, se resuelven tres problemas en la mecánica de sólidos mediante el software MATLAB. Los problemas ocurren en una, dos y tres dimensiones, respectivamente.

Este trabajo está organizado como sigue. En el capítulo 1 se describe los antecedentes generales, objetivos y alcances del trabajo. En el capítulo 2 se estudia los antecedentes necesarios para entender el trabajo en profundidad. En el capítulo 3 se muestra el desarrollo de los tres problemas resueltos. En el capítulo 4 se presenta los resultados obtenidos, más un análisis e interpretación de estos. En el capítulo 5 se discuten los resultados, comparándolos con estudios similares, y se entregan recomendaciones al momento de programar paralelamente problemas mediante el método sin malla Galerkiano. Finalmente, en el capítulo 6 se muestran las principales conclusiones del trabajo.

1.1 Antecedentes generales

Para comprender los objetivos de esta memoria, es necesario entender en qué consiste los métodos sin malla, además de la teoría de elasticidad lineal que envuelve los problemas a resolver, y la programación en paralelo. A continuación se presentan los tópicos antes mencionados.

1.1.1 Métodos sin malla (MSM)

Una malla es una red formada mediante la conexión de nodos de una manera predefinida, ver Figura 1.1.

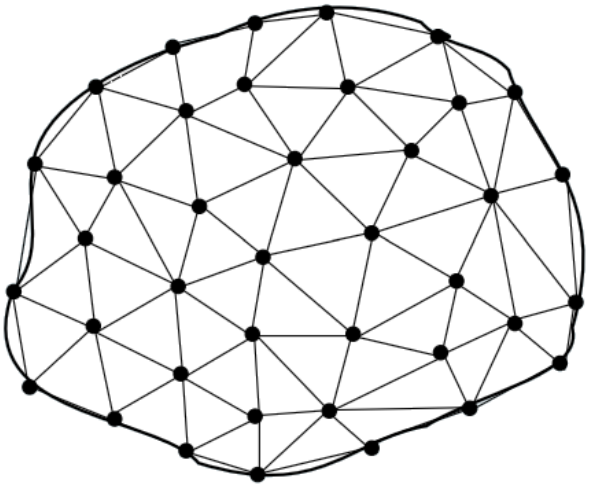

Figura 1.1 Red de nodos que forma una malla. [fuente: Elaboración propia]

Las mallas son utilizadas en variados métodos de simulación numérica, destacando el método de elementos finitos (MEF). Sin embargo, traen consigo desventajas, las cuales son [4]:

- El costoso gasto computacional en su creación, generalmente hecho por el usuario.
- Perdida de exactitud ante grandes deformaciones, y esto aumenta con la distorsión de los elementos.
- La dificultad para simular la rotura de material con un gran número de fragmentos; debido a que los elementos no pueden ser divididos; un elemento debe estar como un todo o desaparecer completamente. Esto generalmente lleva a una mala interpretación de la trayectoria de una rotura.

Debido a lo anterior, es que nacen los métodos sin malla (en adelante MSM). Estos usan nodos dispersos en el dominio y los borde del problema para representarlo.

La variable incógnita de la EDP, en cualquier punto arbitrario dentro del dominio del problema, es interpolada usando funciones de base en los nodos que están dentro de un pequeño soporte local del punto, es decir:

$$u^h = \sum_{i=1}^{n} \phi_i(x) \cdot u_i \tag{1.1}$$

donde

$u^h(x)$: Solución aproximada global en la posición x, con $x \in \mathbb{R}^m$, $m \in \{1,2,3\}$.

$\phi_i(x)$: Función de base en el nodo i.

u_i: Solución exacta para el nodo i.

n: número de nodos incluidos en el soporte del dominio local del punto x.

El soporte local dependerá de la función de base escogida.

Dependiendo del proceso de formulación, los MSM caen en tres categorías: basados en la forma débil, basados en la forma fuerte y mixtos.

Los basados en la forma débil, integran en todo el dominio la ecuación diferencial logrando un problema de algebra lineal mediante integración numérica. Ocurre que para utilizar integración numérica se debe utilizar celdas, lo que convenientemente lo ofrece una malla de elementos, por lo tanto, en los MSM basados en la forma débil se utiliza una malla de puntos solo para este propósito. Por otro lado, los basado en la forma fuerte son directamente discretizados usando técnicas de colocación. Por último, los mixtos utilizan ambos procesos de formulación para distintos grupos de nodos dentro del mismo problema.

En este trabajo se utiliza el método sin malla Galerkiano (MSMG) basado en la forma débil, junto con funciones de base de máxima entropía, para mayor información revisar la Sección 2.

1.1.2 Elasticidad Lineal

Elasticidad se refiere a la propiedad de algunos materiales de resistir esfuerzos externos y deformarse, regresando a su geometría inicial una vez que el esfuerzo es removido.

La ecuación diferencial que gobierna el fenómeno físico es la ecuación de equilibrio (Ecuación 1.2) y la ecuación constitutiva lineal (Ecuación 1.3) la cual se ve representada para un cuerpo cualquiera en la Figura 1.2.

$$\nabla \cdot \sigma + b = 0 \quad en\ \Omega \tag{1.2}$$

con

$$\sigma = C \cdot \varepsilon \tag{1.3}$$

$$n \cdot \sigma = \bar{t}\ en\ \Gamma_t \tag{1.4}$$

$$u = \bar{u}\ en\ \Gamma_u \tag{1.5}$$

$$\varepsilon = \frac{1}{2}(\nabla u + \nabla u^T) \tag{1.6}$$

donde

Ω: Dominio del problema

Γ: Contorno o superficie, con Γ_u condición de borde desplazamiento y Γ_t condición de borde de esfuerzo aplicado.

σ: Vector de esfuerzos.

b: Vector de fuerzas de campo

u: Vector de desplazamiento, con $\bar{u}$ desplazamiento impuesto condición de borde.

$n \cdot \sigma$: Vector de esfuerzo normal a la superficie.

$\bar{t}$: Esfuerzo aplicado a la superficie.

C: Matriz de constantes del material.

ε: Vector deformación.

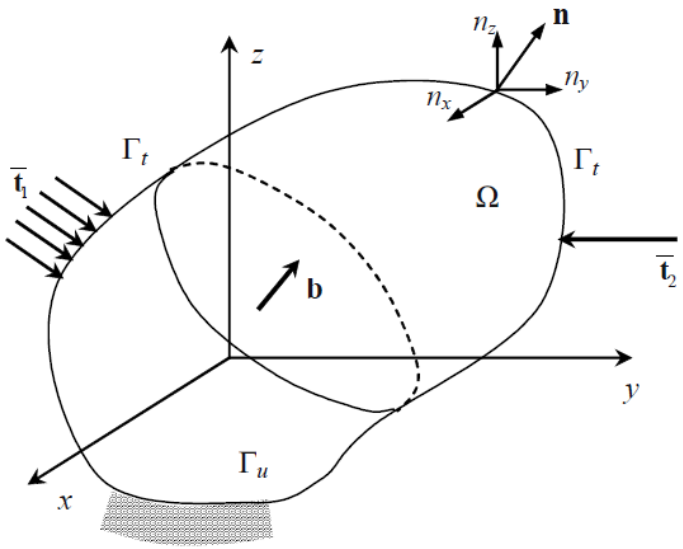

Figura 1.2 Material arbitrario sujeto a esfuerzos. Dominio Ω. [4]

1.1.3 Programación en Paralelo

La computación paralela es una técnica de programación en la que muchas instrucciones se ejecutan simultáneamente. Se basa en el principio de que los problemas grandes se pueden dividir en partes más pequeñas, que pueden resolverse de forma concurrente [3].

Los ordenadores paralelos se pueden clasificar según el nivel de paralelismo que admite su hardware: los ordenadores multinúcleo y multiproceso tienen varios elementos de procesamiento en una sola máquina, mientras que los clúster, emplean varios ordenadores para trabajar en la misma tarea.

La programación en paralelo es más difícil de escribir que la secuencial porque la concurrencia introduce nuevos tipos de errores de software. La comunicación y la sincronización entre las diferentes subtareas son típicamente las grandes barreras para conseguir un buen rendimiento de los programas paralelos.

El software se ha orientado tradicionalmente hacia la computación en serie. Para resolver un problema, se construye un algoritmo y se implementa en un flujo de instrucciones en serie. Estas instrucciones se ejecutan en la unidad central de procesamiento de un ordenador. En el momento en el que una instrucción se termina, se ejecuta la siguiente.

La computación paralela emplea elementos de procesamiento múltiple simultáneamente para resolver un problema. Esto se logra dividiendo el problema en partes independientes de tal manera que cada elemento de procesamiento pueda ejecutar su parte del algoritmo a la misma vez que los demás. Así se consigue más rapidez a la hora de realizar cualquier tipo de operaciones, lo cual desencadena en una gran serie de ventajas.

Los indicadores principales de desempeño para programas en paralelo son dos [19], el Speedup que mide cuanto se acelera el cómputo con respecto a la programación en serie, y la Eficiencia paralela, que mide que tan eficiente se es con respecto al número de procesadores utilizados.

1.2 Motivación

Comparados con el método de elementos finitos, los métodos sin malla son computacionalmente costosos y actualmente son limitados a problemas simples. Sin embargo, los métodos sin malla son ideales para modelar problemas con grandes deformaciones en la malla como lo son daño de material, penetración de proyectiles, fragmentación y movimiento de grietas.

La programación en paralelo ha sido utilizada extensivamente en el desarrollo de los métodos de simulación numérica. Los detalles pueden ser encontrados en bibliografía incluyendo [5;6;7;8;9;10].

La principal motivación de este trabajo es suplir uno de los principales defectos del método sin malla, el cual es ser computacionalmente costosos. Lo anterior se supera mediante la implementación de la programación en paralelo.

1.3 Objetivos y Alcances

1.3.1 Objetivo principal

El objetivo principal es realizar un algoritmo numérico eficiente mediante programación en paralelo, para la resolución de problemas en la mecánica de sólidos mediante el método sin malla Galerkiano.

1.3.2 Objetivos específicos

Los objetivos específicos de este trabajo son:

- Implementación programación paralela problema barra elástica 1D
- Implementación programación paralela problema viga en voladizo 2D
- Implementación programación paralela problema estrato elástico infinito 3D.
- Análisis del tiempo de cómputo, Speedup y Eficiencia Paralela para los tres problemas.

1.3.3 Alcances

Los alcances de este trabajo abarcan solo la implementación de la programación en paralelo mediante software MATLAB para tres problemas (uno, dos y tres dimensiones) específicos mediante el método sin malla Galerkiano, utilizando funciones de base de máxima entropía. Los problemas están en el marco de la mecánica de sólidos (elasticidad lineal).

2 Antecedentes

2.1 Funciones de base

Las funciones de base son aquellas que ponderan la solución nodal obtenida para distribuirla a todo el dominio. Sean *n* nodos en el dominio, las funciones de base en dos dimensiones deben cumplir que:

$$\sum_{i}^{n} \phi_i = 1 \tag{2.1}$$

$$\sum_{i}^{n} \phi_i x_i = x \ , \sum_{i}^{n} \phi_i y_i = y \tag{2.2}$$

con

ϕ_i: Función de base correspondiente a nodo i.

x_i: Coordenada en eje x de nodo i.

x: Coordenada en eje x arbitrario.

y_i: Coordenada en eje y de nodo i.

y: Coordenada en eje y arbitraria.

Las propiedades (2.1) y (2.2) pueden escribirse de la siguiente forma matricial:

$$\underbrace{\begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \end{bmatrix}}_{\substack{\mathbf{A} \\ (3\times n)}} \underbrace{\begin{Bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{Bmatrix}}_{\substack{\boldsymbol{\phi} \\ (n\times 1)}} = \underbrace{\begin{Bmatrix} 1 \\ x \\ y \end{Bmatrix}}_{\substack{\mathbf{b} \\ (3\times 1)}} \tag{2.3a}$$

$$A\phi = b \tag{2.3b}$$

con

A: Matriz que engloba vector unitario y los términos nodales x_i e y_i.

ϕ: Vector de funciones de base nodales.

b: Vector que posee coordenadas arbitrarias x e y.

De la Ecuación (2.3a) y (2.3b) es visible la propiedad de distribuir la solución nodal a cualquier punto [x,y] en el dominio. Además se tiene que no siempre la matriz **A** es cuadrada (depende del número de nodos *n*), lo que implica que el sistema no posee una única solución para las funciones de base ϕ.

Las funciones de base ϕ_i son localizadas en cada nodo i, vale decir, la función de base posee una ponderación para nodos vecinos, pero no para todos los nodos del dominio, lo anterior se cumple en los métodos sin malla.

2.2 Funciones base MAXENT

En teoría de la información, la noción de entropía como medida de la incerteza fue introducida por Shanon [11]. La cantidad

$$H(p1, p2, \dots, p_n) = -\sum_{i=1}^{n} p_i log(p_i) \tag{2.4}$$

es referida como la entropía informativa de un conjunto, donde p_i es la probabilidad asociada a la ocurrencia de un evento x_i, sujeto a la condición

$$\sum_{i}^{n} p_i = 1 \tag{2.5}$$

Tomando el trabajo de Shannon, Jaynes [12][13] propone el principio de máxima entropía (MAXENT), en donde establece que la distribución estadística menos sesgada que se le puede atribuir a un sistema estadístico es aquella en que dadas ciertas condiciones fijas maximiza la entropía, esto es, que la desinformación es máxima.

Así, las funciones de base ϕ_i se asocian a la probabilidad de ocurrencia p_i [14], y las coordenadas nodales x_i a los eventos x_i. Se introduce los prior m_i que son los encargados de localizar las funciones de base. Se tiene el siguiente problema de optimización:

$$MAX(H(\phi_1, \phi_2, \dots, \phi_n) = -\sum_{i=1}^{n} \phi_i \, log\left(\frac{\phi_i}{m_i}\right) \tag{2.6}$$

con

m_i: Prior, variable encargada de localizar la función de base.

sujeto a

$$\sum_{i=1}^{n} \phi_i = 1 \tag{2.7}$$

$$\sum_{i=1}^{n} \phi_i x_i = x \ , \sum_{i=1}^{n} \phi_i y_i = y \tag{2.8}$$

el cual se resuelve mediante multiplicadores de Lagrange.

En este trabajo el prior utilizado para los problemas barra elástica y estrato infinito es Gaussiano e igual a

$$m_a(q) = \mathrm{EXP}\left(-\frac{\gamma}{h_a} \cdot \|\mathrm{x_a} - \mathrm{x}\|\right) \tag{2.9}$$

Para el caso del problema viga bidimensional se utilizo el prior cuartico e igual a

$$m_a(q) = \begin{cases} 1 - 6q^2 + 8q^3 - 3q^4, & 0 \le q_a \le 1 \\ 0 \quad , & q_a > 1 \end{cases} \tag{2.10a}$$

$$q_a = \frac{\|x - x_a\|}{\gamma \cdot h_a} \tag{2.10b}$$

con

$\|x_a - x\|$: distancia entre coordenada en nodo a con vector arbitrario x.

h_a: espaciamiento nodal característico.

γ: constante adimensional que controla el radio del soporte.

Cabe señalar que cada computador genera distintos errores de aproximación (los que varían muy poco entre ordenadores), pero que pueden producir errores mayores si no se escoge el γ adecuado (distinto para cada ordenador). Es necesario entregar el valor de γ en cada resultado ya que controlan el radio de localización de las funciones de base.

2.3 Metodología para resolución de problemas mediante métodos sin malla.

Los pasos para resolver una EDP mediante el método sin malla son:

A. Inicialización de datos y geometría del problema.

B. Cuadratura de Gauss.

C. Implementación de condiciones de borde.

D. Generación aproximación (para todo el dominio) a partir de solución nodal.

Se explica a continuación en qué consiste cada etapa.

2.3.1 Inicialización de datos y geometría del problema

En esta etapa se inician los parámetros básicos del problema, como lo son todas las variables dentro de la EDP, constantes involucradas, etc.

Luego se inicia la malla de puntos, matriz de conectividad y nodos a utilizar en el problema. Se debe tener en cuenta que a medida que se refina la malla, se obtienen aproximaciones más convergentes a la solución analítica.

2.3.2 Cuadratura de Gauss

La cuadratura de Gauss se refiere a particionar el problema, esto es escoger los puntos de Gauss a utilizar para el desarrollo del problema. En esta etapa también se inicia la matriz de rigidez y el vector de fuerza.

La partición se hace de tal forma que cada elemento es independiente uno del otro, lo que tiene la ventaja de ser fácilmente paralelizable.

2.3.3 Implementación condición de borde

Las condiciones de borde son efectuadas al final. Dependiendo del tipo de condición de borde a implementar se debe modificar la matriz de rigidez o el vector de fuerza. Luego de esta etapa, se obtiene la solución a la EDP nodal.

2.3.4 Generación aproximación a partir de solución nodal.

Finalmente, se obtiene una solución por nodo u, la cual es ponderada con las funciones de base para entregar la solución final aproximada $u^h(x)$, es decir:

$$u^h(x) = \phi^T(x) \cdot u \tag{2.11}$$

con

$u^h(x)$: solución aproximada para posición en vector x arbitrario.

$\phi(x)$: vector de funciones de base en posición vector x.

u: Vector de solución nodal.

2.4 Test de la parcela

El error numérico y la convergencia del problema se obtienen de conocer las normas relativas L^2 y H^1 para cada malla del problema, las normas L^2 y H^1 se calculan como:

$$\frac{\|u - u_h\|_{L^2}}{\|u\|_{L^2}} = \sqrt{\frac{\sum_{e=1}^{numel} \int_{\Omega_e} (u - u_h) \cdot (u - u_h) d\Omega}{\sum_{e=1}^{numel} \int_{\Omega_e} u \cdot u \, d\Omega}} \tag{2.12}$$

$$\frac{\|u - u_h\|_{H^1}}{\|u\|_{H^1}} = \sqrt{\frac{\sum_{e=1}^{numel} \int_{\Omega_e} (\varepsilon - \varepsilon_h) \cdot (\sigma - \sigma_h) d\Omega}{\sum_{e=1}^{numel} \int_{\Omega_e} \varepsilon \cdot \sigma \, d\Omega}} \tag{2.13}$$

donde

u: Solución exacta del desplazamiento.

ε: Solución exacta de la deformación.

σ: Solución exacta del esfuerzo.

u^h:Solución numérica del desplazamiento.

ε^h: Solución numérica de la deformación.

σ^h:Solución numérica del esfuerzo.

Se debe conocer la solución analítica del problema para poder calcular las normas relativas. La tasa ideal de convergencia de las normas H^1 y L^2 son 1 y 2, respectivamente.

2.5 Programación en Paralelo MATLAB

A continuación se explica la forma en que trabaja MATLAB en el desarrollo de programación en paralelo y las librerías existentes.

2.5.1 Parallel MATLAB

Existen tres tipos de tecnologías que permiten desarrollar con MATLAB programas en paralelo: pMATLAB, Star-P y Parallel Computing Toolbox.

MATLAB permite resolver problemas usando procesadores multinúcleos o computadores tipo cluster, trabajando con "workers", quienes distribuyen entre sí el problema a paralelizar.

En la Figura 2.1 es posible visualizar la forma en que opera la paralelización en MATLAB, a través de una interfaz central llamada MATLAB client, se entrega el código paralelizado a MATLAB workers, quienes realizan el trabajo simultáneamente, la Figura 2.1 muestra el comando *parfor* de la librería Parallel Computing Toolbox.

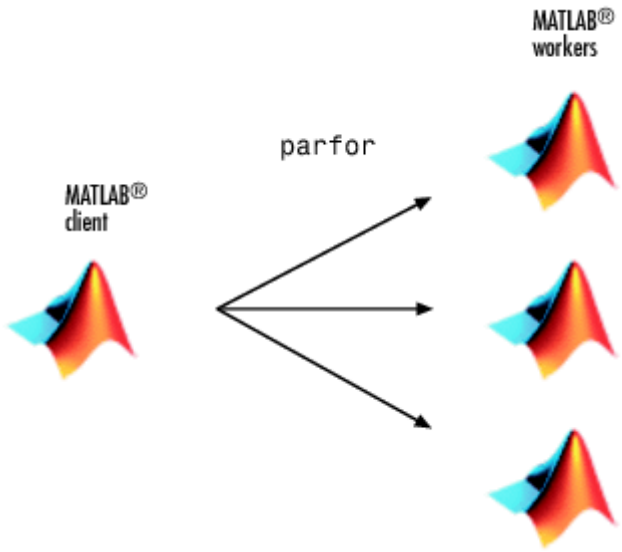

Figura 2.1 Paralelización en MATLAB mediante código *parfor* de la librería Parallel Computing Toolbox. [15]

El grado de dificultad a paralelizar aumenta al existir interacción entre workers. Las librerías pMATLAB [16] y Star-p ofrecen esta posibilidad, mientras que Parallel Computing Toolbox no [15]. Los métodos sin malla discretizan el problema, existiendo independencia entre cada partición, por lo que se adoptará por simplicidad la librería Parallel Computing Toolbox, ya que no existe interacción entre elementos.

2.5.2 Parallel Computing Toolbox

Como se dijo anteriormente, es posible trabajar con varias sesiones de MATLAB a la vez, esto es coordinado y ejecutado por el software MATLAB Distributed Computing Server. Se trabaja con la paralelización para bucles mediante el comando *parfor*.

El concepto básico de un bucle paralelo es el mismo que el bucle estándar de MATLAB, es decir, ejecuta una serie de declaraciones (en el cuerpo del ciclo) en un rango de valores. Parte del cuerpo del *parfor* se ejecuta en el cliente (o unidad central) de MATLAB (con el que se envía el *parfor*) y parte se ejecuta en paralelo en los trabajadores o workers. Los datos necesarios con los que opera *parfor*, se envían desde el cliente a los trabajadores, donde se realizan la mayoría de los cálculos y los resultados se envían de vuelta al cliente. Debido a que varios workers pueden ser computacionalmente concurrentes en el mismo ciclo, un ciclo *parfor* puede proporcionar un rendimiento significativamente mejor que un ciclo normal *for*. Cada ejecución del cuerpo de un bucle *parfor* es una iteración. Los "workers" de MATLAB evalúan las iteraciones sin ningún orden en particular y de forma independiente unas de otras. Debido a que cada iteración es independiente, no hay garantía de que las iteraciones se sincronicen en modo alguno, ni hay necesidad de esto. Si el número de trabajadores es igual al número de iteraciones del bucle, cada trabajador realiza una iteración del bucle. Si hay más iteraciones que "workers", algunos "workers" realizaran más de una iteración, en este caso un trabajador puede recibir múltiples iteraciones a la vez para reducir el tiempo de comunicación.

Un bucle *parfor* es útil en situaciones donde se necesitan muchas iteraciones de un bucle, donde estas iteraciones son de fácil cálculo. *Parfor* divide las iteraciones del bucle en grupos para que cada trabajador realice una parte del número total de iteraciones. Los bucles *parfor* también son útiles cuando se tienen iteraciones donde cada una de ellas tarda un tiempo prolongado en ejecutarse, porque los trabajadores pueden realizar estas iteraciones de forma simultánea. No se puede utilizar un bucle *parfor* cuando una iteración del bucle depende de los resultados de otras iteraciones. Cada iteración debe ser independiente de todas las demás. Dado que existe un coste en la comunicación en el bucle *parfor*, no hay ninguna ventaja si el número de operaciones que se tienen que realizar son muy pequeñas, ya que el tiempo de comunicación será mayor que el de realizar dichas operaciones, este fenómeno recibe el nombre de *overhead.*

2.6 Indicadores de desempeño para programas en paralelo

La eficiencia computacional es el propósito principal de escribir programas en paralelo. Es por ello que el tiempo de cómputo del programa en paralelo debe ser comparado con el correspondiente tiempo de la programación en serie. Los indicadores más comúnmente usados para evaluar el desempeño son "Speedup" y "Eficiencia paralela". El detalle de estos indicadores puede ser encontrado en bibliografía [3][5]. El tiempo de simulación y el costo computacional son dos parámetros necesarios para entender la Eficiencia paralela y el Speedup. Para un programa secuencial, el tiempo de simulación es el tiempo entre el comienzo y el final de la simulación, mientras que para un programa en paralelo, es el tiempo entre el comienzo de la simulación y el final del último procesador. El costo computacional de un programa en paralelo C_n es el tiempo tomado por todos los procesadores involucrados en el cómputo:

$$C_n = n \cdot T_n \tag{2.13}$$

Donde *n* es el número de procesadores y T_n es el tiempo del programa que tarda con *n* procesadores (tiempo de simulación en paralelo). Speedup S_n es el cociente entre el tiempo de cómputo que se tarda en un procesador, T_1, con el tiempo que tarda el programa con *n* procesadores, T_n, es decir:

$$S_n = \frac{T_1}{T_n} \tag{2.14}$$

En una situación ideal $S_n = n$, pero generalmente se tiene que $S_n \leq n$ debido al trabajo adicional hecho por el programa en paralelo, conocido como *overhead*. Este *overhead* consiste en la comunicación entre procesadores, gasto en sincronización, que genera tiempo de inactividad en procesadores debido a no dividir en partes iguales el problema. Más aun, es imposible paralelizar 100% un código debido a partes secuenciales propias, como lo son la entrada y salida de un programa. La eficiencia E_n es el cociente entre el tiempo de simulación de un solo procesador con el tiempo total de simulación de todos los procesadores, o alternativamente es el cociente entre el Speedup S_n con el número de procesadores *n*:

$$E_n = \frac{T_1}{nT_n} = \frac{S_n}{n} \tag{2.15}$$

En una situación ideal $E_n = 1$ pero generalmente $E_n \leq 1$.

3 Desarrollo problemas específicos

Durante este capítulo, se explica el desarrollo de los tres problemas específicos a resolver, junto con la solución analítica y su resolución mediante los métodos sin malla implementando la programación en paralelo.

3.1 Barra elástica en 1D.

El problema consiste en una barra elástica en una dimensión empotrada en un extremo a la cual se le aplica una fuerza axial t en el borde libre. La barra se observa en la Figura 3.1. Resolver el problema consiste en obtener el vector desplazamiento horizontal u_x.

Teniendo una sección de área transversal A y modulo de elasticidad E constantes a lo largo de la barra, se tiene que la ecuación diferencial de equilibrio interno, junto con las condiciones de borde del problema son:

$$EA\frac{d^2u_x(x)}{dx^2} = 0 \tag{3.1a}$$

$$u_x(0) = 0 \tag{3.1b}$$

$$\frac{du_x(L)}{dx} = \frac{t}{E} \tag{3.1c}$$

con

$u_x(x)$: Vector desplazamiento horizontal en la posición x.

El problema es resuelto utilizando los siguientes valores numéricos (ver Tabla 3.1).

Tabla 3.1 Propiedades mecánicas barra elástica

Propiedad Mecánica	Magnitud
Módulo de Young E	$10^7 [psi]$
Largo barra L	$8[in]$
Área sección transversal	$1[in^2]$
Carga t	$100000\ [psi]$

Fuente: Elaboración propia

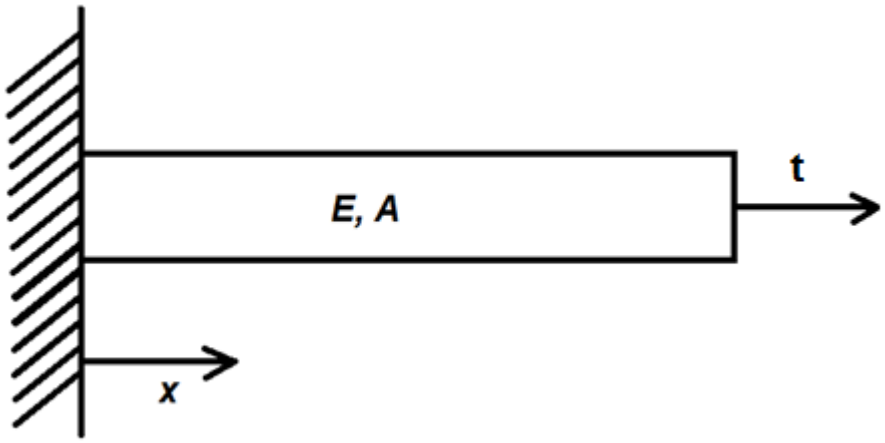

Figura 3.1 Problema barra elástica 1D. Sección de área A, módulo de Young E, y fuerza t en extremo libre.[Fuente: Elaboración propia].

3.1.1 Solución analítica barra elástica.

La resolución de la Ecuación 3.1 da como resultado el vector desplazamiento u_x e igual a

$$u_x(x) = \frac{t}{E}x \tag{3.2}$$

3.1.2 Solución aproximada barra elástica mediante MSMG.

Siguiendo la metodología de resolución de la Sección 2.3 de este informe, se resuelve el problema mediante el método sin malla Galerkiano (ver código en Anexo A) utilizando el software MATLAB. Se crea la función *parte1_parelela(nelementos,npgauss)* donde resuelve el problema de la barra elástica entregando el vector desplazamiento aproximado ${u_x}^h$ y el tiempo de iteración T_n con n el número de procesadores involucrados. Los argumentos de la función corresponden a el número de elementos que particiona el problema *nelementos*, y el número de puntos de Gauss *npgauss*.

La implementación de la programación en paralelo se realiza en la sección "Cuadratura de Gauss" del código. En esta sección, se resuelve la EDP discretamente y se tiene la característica que existe independencia en cada partición, por lo que no existe inconveniente al dividir este comando en los procesadores involucrados, esto ocurre para los tres problemas realizados, para más información ver los códigos de los tres problemas en la sección Anexos A,B y C.

3.2 Viga en voladizo 2D

El problema consiste en un viga bidimensional empotrada y sujeta a una carga de tracción parabólica en el extremo libre [17]. La geometría, el sistema de coordenadas, la carga y las condiciones de borde para el problema se ilustran en la Figura 3.2. La resolución del problema consiste en obtener los vectores desplazamientos horizontal y vertical, u_x y u_y, respectivamente.

Se resuelve utilizando los parámetros presentados en la Tabla 3.2. Además se trabaja utilizando tres mallas triangulares especificas, las cuales llamaremos malla 1, malla 2 y malla 3. Se puede apreciar el tamaño de cada malla en la Figura 3.3. El tamaño de la malla triangular se define como el diámetro máximo entre todos los círculos que inscriben a cada triángulo, los tamaños utilizados son 1.410, 0.821 y 0.438 para las mallas 1,2 y 3, respectivamente. Los detalles de las mallas utilizadas se encuentran en la Tabla 3.3

Tabla 3.2 Propiedades mecánicas viga en voladizo.

Propiedad Mecánica	Magnitud
Modulo de Elasticidad E	$10^7 [psi]$
Coeficiente de Poisson v	0,3
Largo viga L	8 $[in]$
Alto viga D	$4[in]$
Carga P, parabólica	-100000 $[lbf]$

Fuente: Elaboración propia

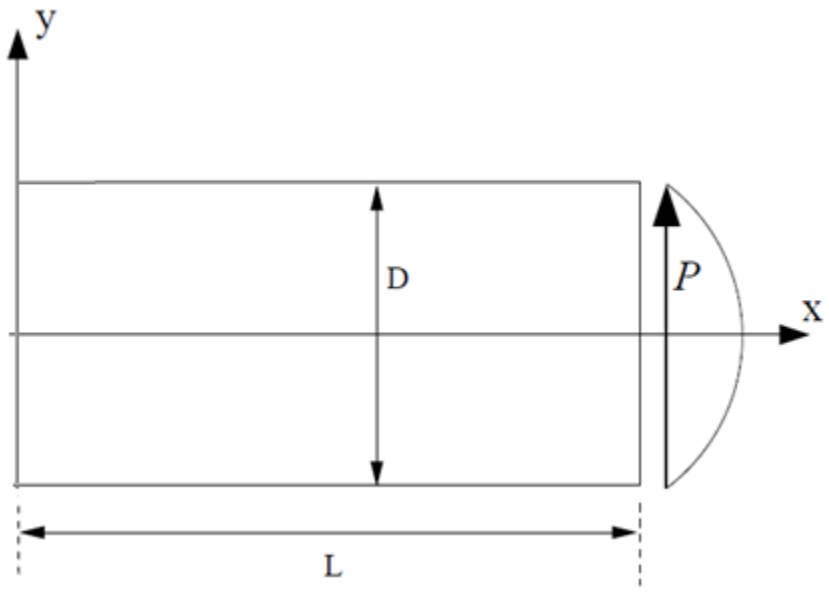

Figura 3.2 Geometría y condiciones de borde para el problema de la viga en voladizo. [Fuente imagen: Elaboración propia]

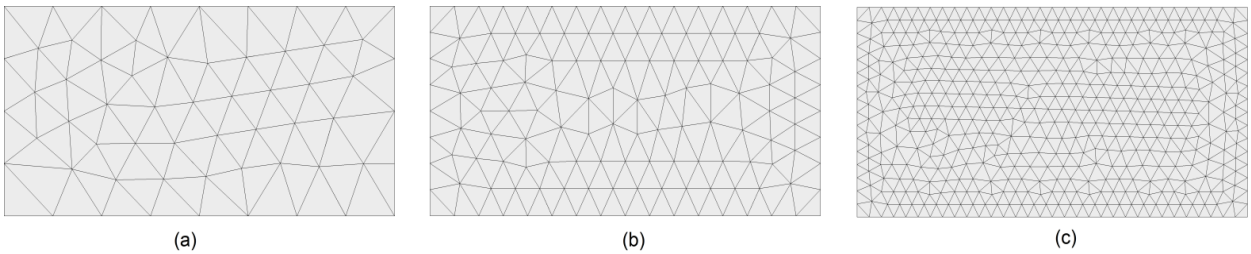

Figura 3.3 Mallas utilizadas para el problema viga bidimensional. El tamaño de malla disminuye de izquierda a derecha, con (a)malla 1, (b)malla 2 y (c)malla 3.

Tabla 3.3 Propiedades de mallas utilizadas.

Malla utilizada	Número de nodos	Número de elementos	Tamaño de malla
Malla 1	61	96	1.410
Malla 2	136	222	0.821
Malla 3	499	900	0.438

3.2.1 Solución analítica viga bidimensional empotrada

Con un estado de deformación plana, módulo de elasticidad E y alto de viga D constante a lo largo de la viga se tiene que la solución analítica según Timoshenko y Goodier [17] es:

$$u_x = -\frac{Py}{6\bar{E}I}\left((6L-3x)x + (2+\bar{v})y^2 - \frac{3D^2}{2}(1+\bar{v})\right) \tag{3.3a}$$

$$u_y = \frac{P}{6\bar{E}I}(3\bar{v}y^2(L-x) + (3L-x)x^2) \tag{3.3b}$$

donde

$$\bar{E} = \frac{E}{(1-v^2)} \tag{3.4a}$$

$$\bar{v} = \frac{v}{(1-v)} \tag{3.4b}$$

$$I = \frac{D^3}{12} \tag{3.4c}$$

con

P: Carga vertical máxima de la distribución parabólica en extremo libre.

v: Coeficiente de Poisson.

I: Segundo momento de área. Se tiene espesor unitario.

L: Largo de la viga.

3.2.2 Solución aproximada viga bidimensional mediante MSMG

Siguiendo la metodología de resolución de la sección 2.3 de este informe, se resuelve el problema mediante el método sin malla Galerkiano (ver código en Anexo B) utilizando el software MATLAB. Se crea el script *parte2_parelela()* donde resuelve el problema de la viga bidimensional entregando el tiempo de cómputo T_n y el vector desplazamiento u^h definido como

$$u^h = \begin{pmatrix} u_{x1} \\ u_{y1} \\ \vdots \\ u_{xn} \\ u_{yn} \end{pmatrix} \tag{3.5}$$

con

u_{xi}: Vector desplazamiento horizontal nodo i .

u_{yi}: Vector desplazamiento vertical nodo i .

Este vector posee las posiciones verticales y horizontales de todos los nodos. Dentro de la función *parte2_paralela()* se debe escoger que malla utilizar.

3.3 Estrato elástico infinito 3D

El problema consiste en un estrato infinito, sometido a una presión uniforme en la superficie (ver Figura 3.4). La altura del estrato es de 1 [m], y el valor de la presión es $q = 10^6[Pa]$. Un cubo de 2x2x2 de dominio es utilizado para resolver numéricamente el problema, con condiciones de frontera en las caras de forma que se restringe el movimiento al eje perpendicular de la cara, excepto para la cara superior que es donde recibe la presión. Las propiedades mecánicas del estrato están resumidas en la Tabla 3.4.

Se resuelve utilizando tres mallas tetraédricas, las cuales llamaremos malla 1, malla 2 y malla 3. Se puede apreciar el tamaño de cada malla en la Figura 3.5, el tamaño de la malla se define como el diámetro máximo entre todas las esferas que inscriben a cada tetraedro, los tamaños utilizados son 0.6326, 0.3754 y 0.292 para las mallas 1,2 y 3, respectivamente. Los detalles de las mallas utilizadas se encuentran en la Tabla 3.5.

Tabla 3.4 Propiedades mecánicas estrato infinito.

Propiedad Mecánica	Magnitud
Modulo de Young E_y	$4 \cdot 10^7 [Pa]$
Coeficiente de Poisson v	0,3
Carga q	$10^6 [Pa]$

Fuente: Elaboración propia

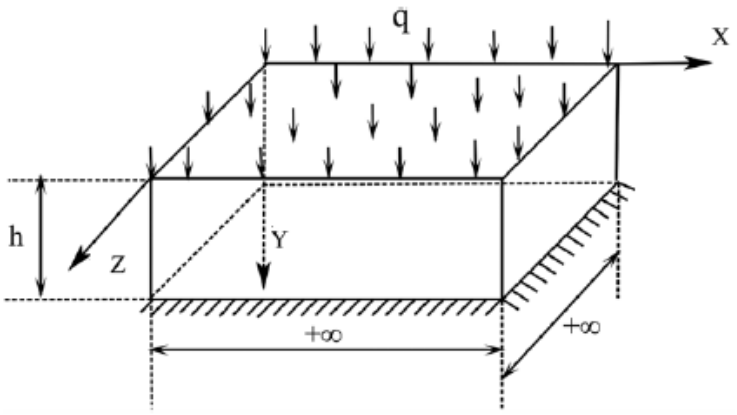

Figura 3.4 Geometría y Condiciones de contorno para problema de estrato infinito. [Fuente: Figura adaptada de [18]]

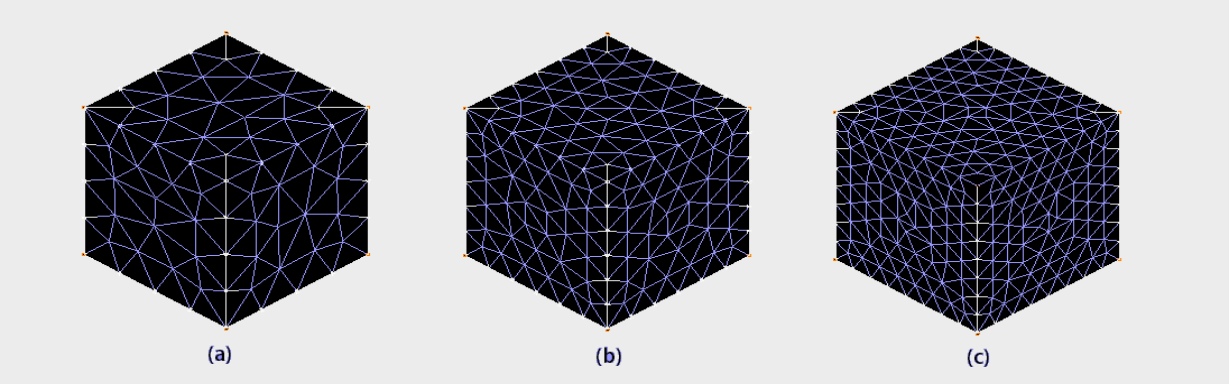

Figura 3.5 Mallas utilizadas para el problema estrato infinito tridimensional. (a) malla 1, (b) malla 2 y (c) malla 3.

Tabla 3.5 Propiedades mallas utilizadas.

Mallas utilizadas	Número de nodos	Número de elementos	Tamaño de malla
Malla 1	209	800	0.6326
Malla 2	446	1862	0.3754
Malla 3	1013	4814	0.2920

3.3.1 Solución analítica estrato infinito

La solución exacta del problema fue obtenida de la bibliografía [18] y es

$$u_x = u_z = 0 \tag{3.6}$$

$$u_y = \frac{(1+v)(1-2v)}{E_y(1-v)}(-qy) \tag{3.7}$$

con

u_i: Desplazamiento en eje i con $i \epsilon \{x, y, z\}$.

v: Coeficiente de Poisson.

E_y: Módulo de elasticidad.

h: Altura del estrato.

3.3.2 Solución aproximada estrato infinito mediante MSMG

Siguiendo la metodología de resolución de la Sección 2.3 de este informe, se resuelve el problema mediante el método sin malla Galerkiano (ver código en Anexo C) utilizando el software MATLAB. Se crea el script *parte3_paralela()* donde resuelve el problema del estrato infinito tridimensional entregando el tiempo de cómputo T_n y el vector desplazamiento nodal u^h definido como sigue:

$$u^h = \begin{pmatrix} u_{x1} \\ u_{y1} \\ u_{z1} \\ \vdots \\ u_{xn} \\ u_{yn} \\ u_{zn} \end{pmatrix} \tag{3.8}$$

con

u_{xi}: Vector desplazamiento en eje x, nodo i.

u_{yi}: Vector desplazamiento en eje y, nodo i.

u_{zi}: Vector desplazamiento en eje z, nodo i.

Este vector posee los desplazamientos en las tres dimensiones de todos los nodos. Dentro de la función *parte3_paralela()* se debe escoger que malla utilizar.

4 Resultados

En este capítulo se muestran los resultados obtenidos para los tres problemas específicos. Los problemas fueron resueltos en el cluster de la Universidad de Chile, Departamento de Ingeniería Mecánica, con 1-12 procesadores.

Se presentan las normas H^1 y L^2, el tiempo de cómputo, Speedup y la Eficiencia paralela para cada problema.

4.1 Resultados barra elástica

Los resultados se obtienen de correr el código (adjunto en Anexo A) para 10, 100 y 1000 elementos utilizando 10 puntos de Gauss por elemento en la cuadratura, con valor γ de 2.1 (Ver Sección 2.2 para mayor información sobre γ). En la Figura 4.1 y Figura 4.2 se observa las normas relativas H^1 y L^2, respectivamente. Se coloca la tasa óptima de convergencia en cada gráfico, con valor de 1 para la norma H^1 y 2 para L^2.Se resume lo obtenido en la Tabla 4.1.

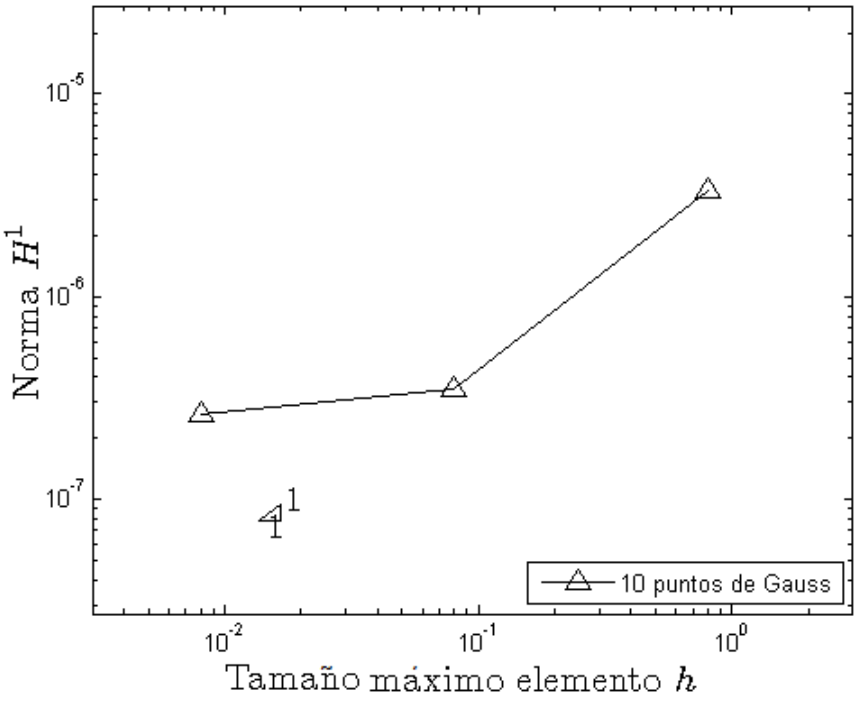

Figura 4.1 Norma relativa H^1 para problema barra elástica.

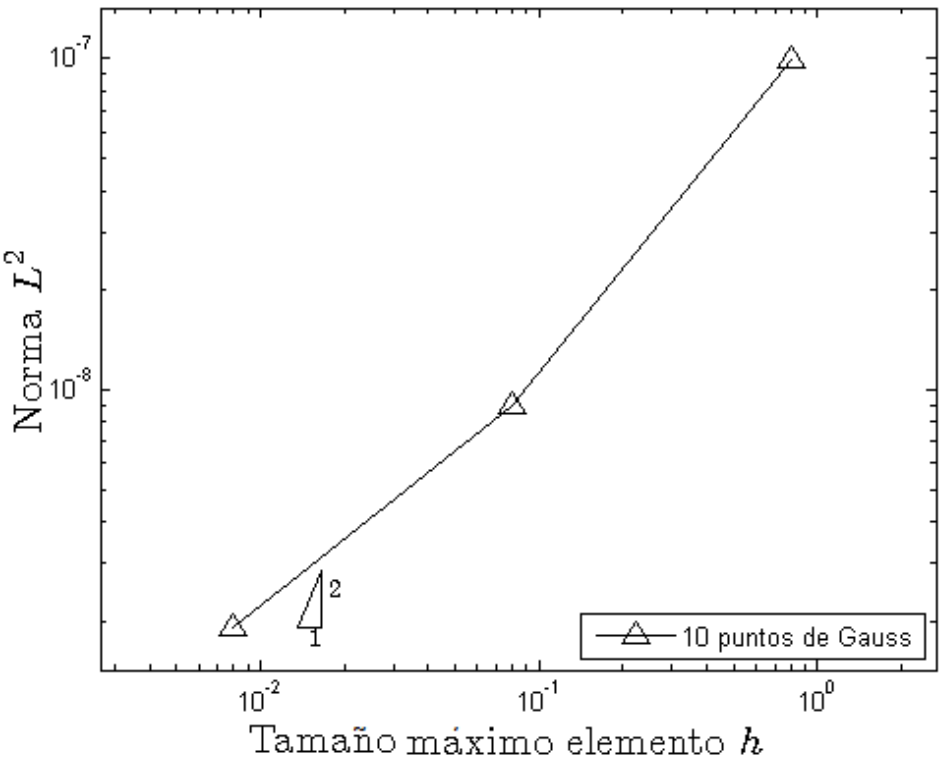

Figura 4.2 Norma relativa L^2 para problema barra elástica.

Tabla 4.1 Resultados normas relativas H^1 y L^2 para problema barra elástica.

Malla utilizada	Nodos	Norma relativa H^1	Norma relativa L^2
Malla 1	10	$3.36 \cdot 10^{-6}$	$9.895 \cdot 10^{-8}$
Malla 2	100	$3.50 \cdot 10^{-7}$	$8.975 \cdot 10^{-9}$
Malla 3	1000	$2.59 \cdot 10^{-7}$	$1.910 \cdot 10^{-9}$

Los grados de libertad (en adelante DOF, por sus siglas en inglés) del problema son el número de nodos multiplicado por el número de puntos de Gauss, y definen el número de iteraciones que realiza el programa. Las normas relativas obtenidas fueron independientes del número de procesadores utilizados.

Se obtiene el tiempo de cómputo, Speedup y Eficiencia paralela (ver Figura 4.3, 4.4 y 4.5, respectivamente). Además se compara con la situación ideal para los indicadores Speedup y Eficiencia paralela.

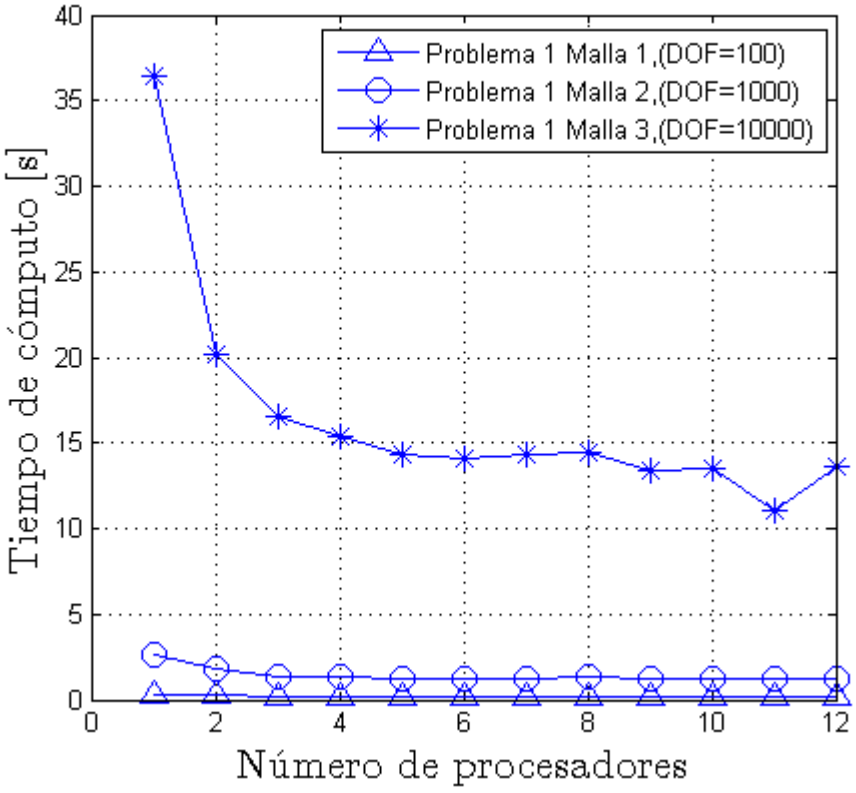

Figura 4.3 Tiempo de cómputo en función del número de procesadores involucrados.

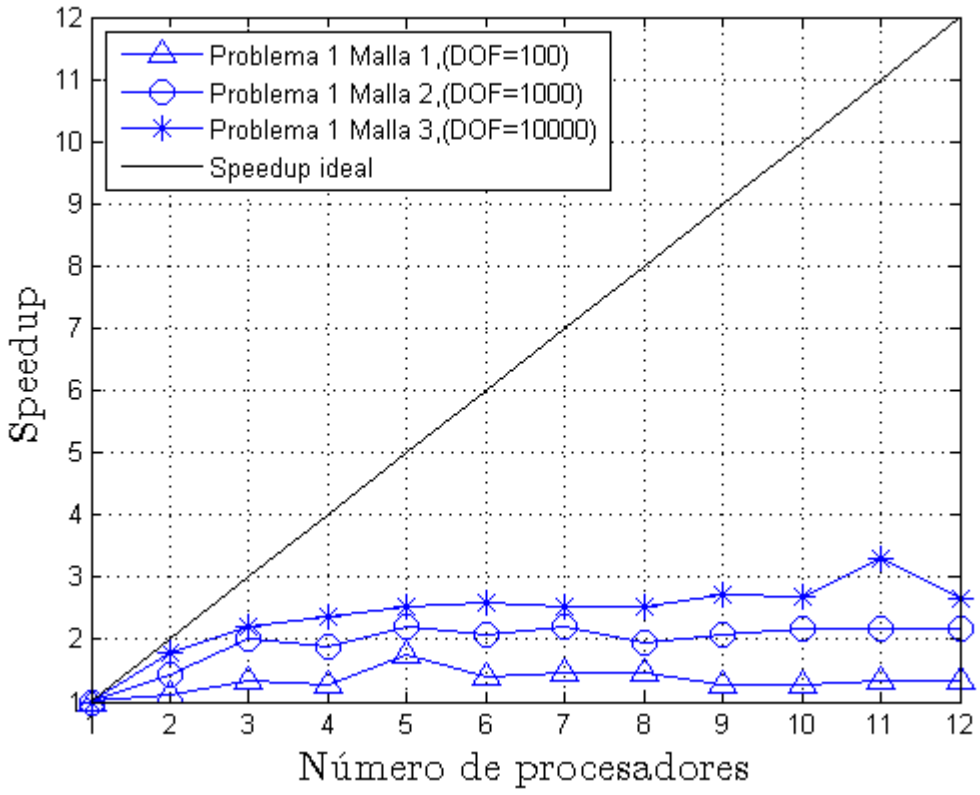

Figura 4.4 Speedup para distintas particiones del problema. Se compara con la situación ideal (en negro).

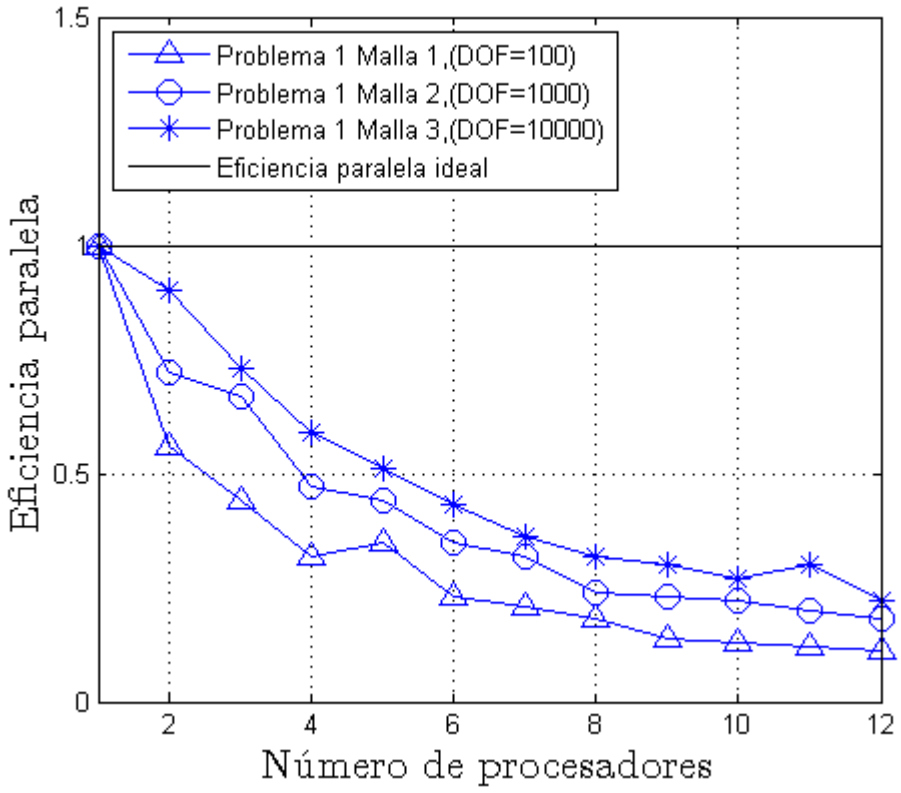

Figura 4.5 Eficiencia paralela para distintas particiones del problema. Se compara con la situación ideal (en negro).

4.2 Resultados viga bidimensional

Los resultados se obtienen de correr el código (adjunto en Anexo B) para tres mallas (Tabla 3.3) utilizando un factor γ de 1.51 para el prior cuartico (Ver Sección 2.2) y 6 puntos de Gauss por elemento en la integración numérica. Se obtienen las normas relativas H^1 y L^2, en la Figura 4.6 y 4.7, respectivamente. Se resume lo obtenido en la Tabla 4.2. Las normas relativas obtenidas fueron independientes del número de procesadores utilizados.

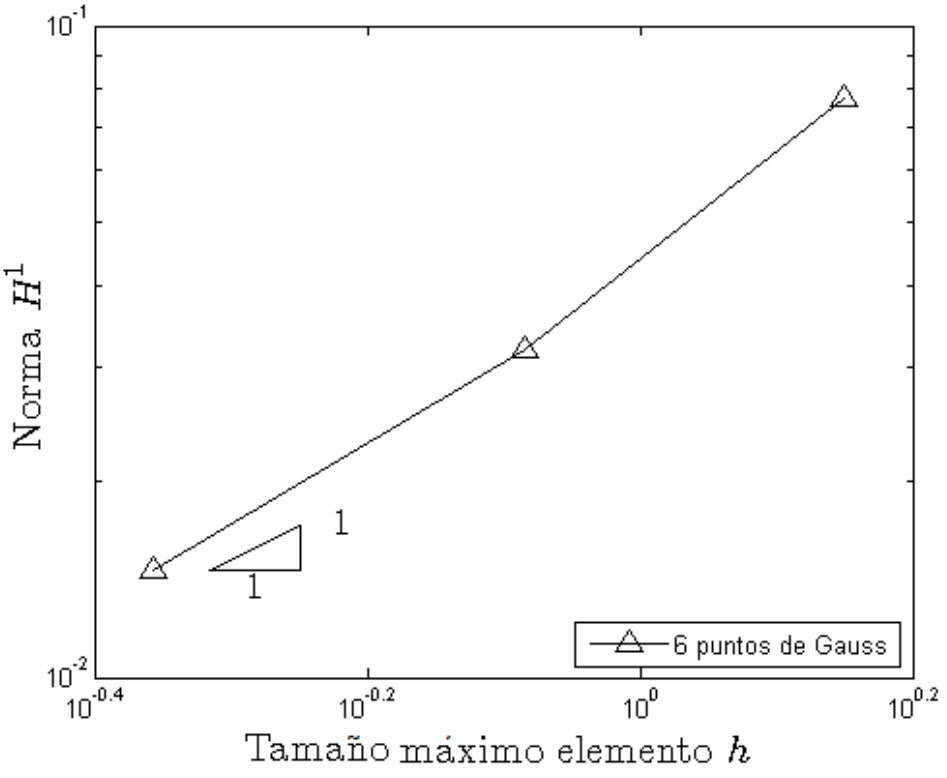

Figura 4.6 Norma relativa H^1 para problema viga bidimensional

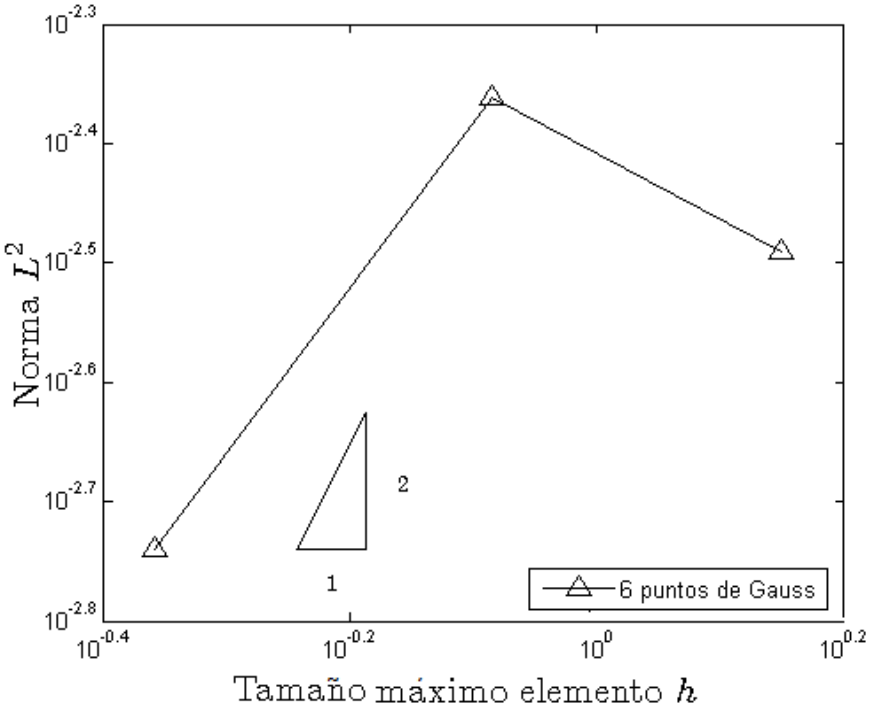

Figura 4.7 Norma relativa L^2para problema viga bidimensional.

Tabla 4.2 Resultados normas relativas H^1 y L^2 para problema viga bidimensional

Malla utilizada	Nodos	Norma relativa H^1	Norma relativa L^2
Malla 1	61	$7.752 \cdot 10^{-2}$	$3.230 \cdot 10^{-2}$
Malla 2	136	$3.185 \cdot 10^{-2}$	$4.350 \cdot 10^{-2}$
Malla 3	499	$1.466 \cdot 10^{-2}$	$1.820 \cdot 10^{-2}$

El tiempo de cómputo, Speedup y Eficiencia paralela se observan en la Figura 4.8, 4.9 y 4.10, respectivamente.

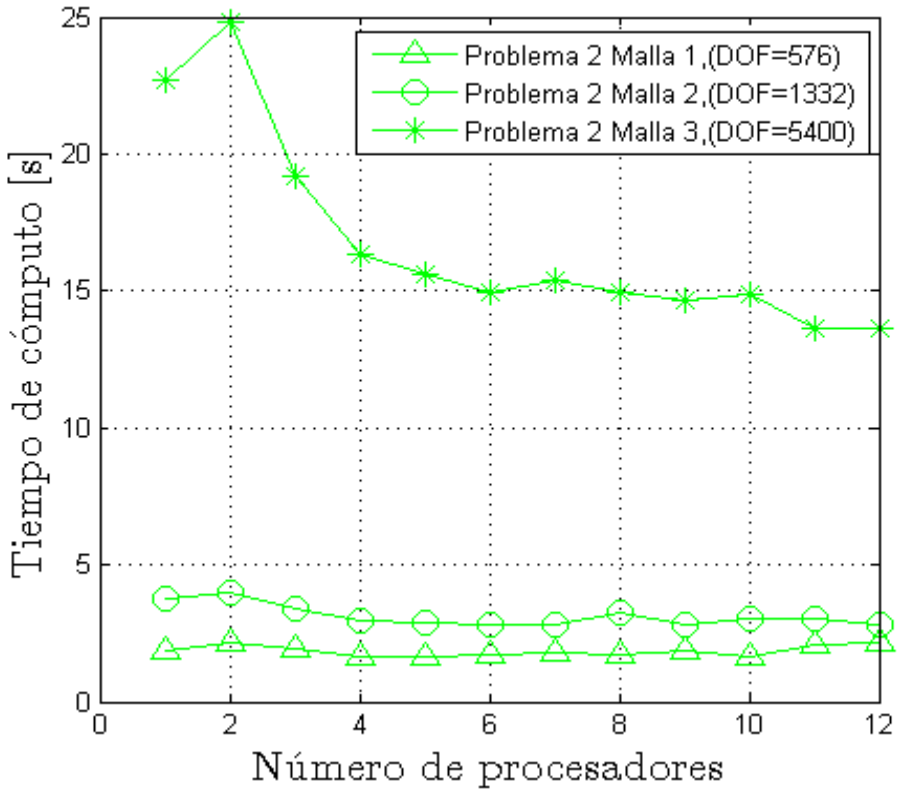

Figura 4.8 Tiempo de cómputo en función del número de procesadores involucrados.

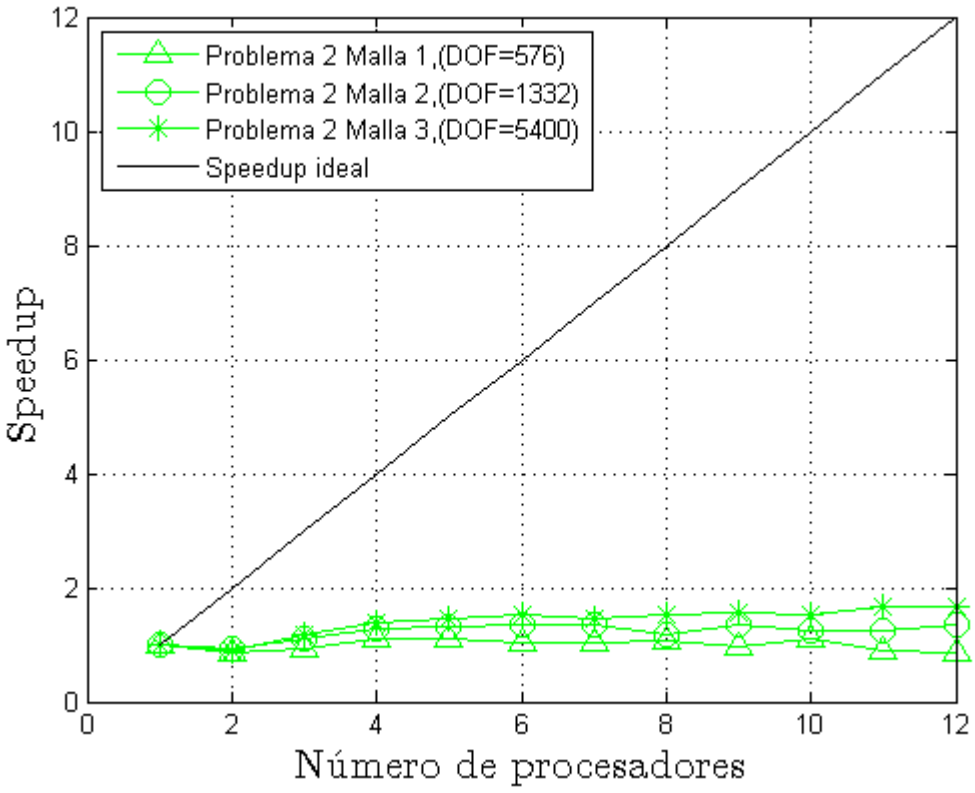

Figura 4.9 Speedup para distintas mallas utilizadas. Se compara con la situación ideal (en negro).

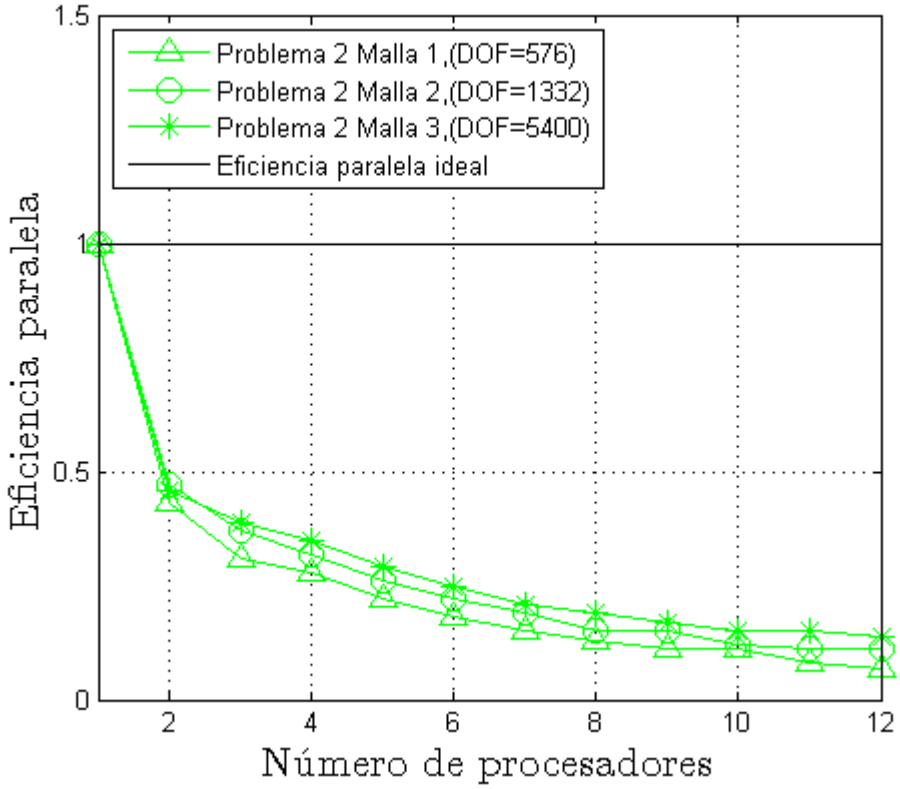

Figura 4.10 Eficiencia paralela para distintas mallas utilizadas. Se compara con la situación ideal (en negro).

4.3 Resultados estrato infinito

Los resultados se obtienen de correr el código (código adjunto en Anexos C) para tres mallas (Tabla 3.4) utilizando un factor γ de 2 para el prior Gaussiano (Ver Sección 2.2) y 3 puntos de Gauss por elemento para integración numérica . Se obtienen las normas relativas H^1 y L^2, en la Figura 4.11 y 4.12, respectivamente. Las normas relativas obtenidas fueron independientes del número de procesadores utilizados.

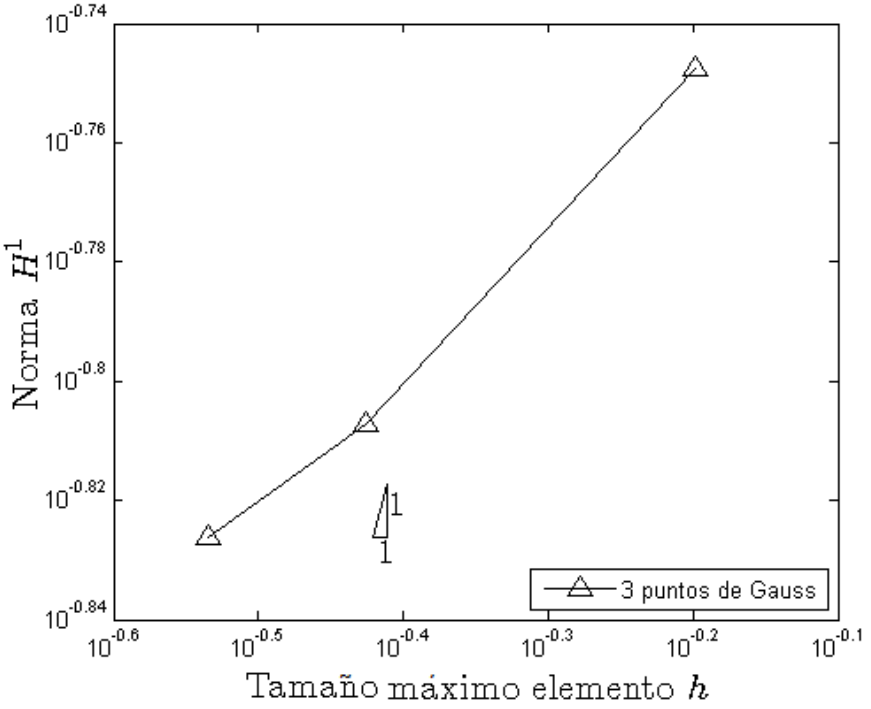

Figura 4.11 Norma relativa H^1 para problema estrato infinto.

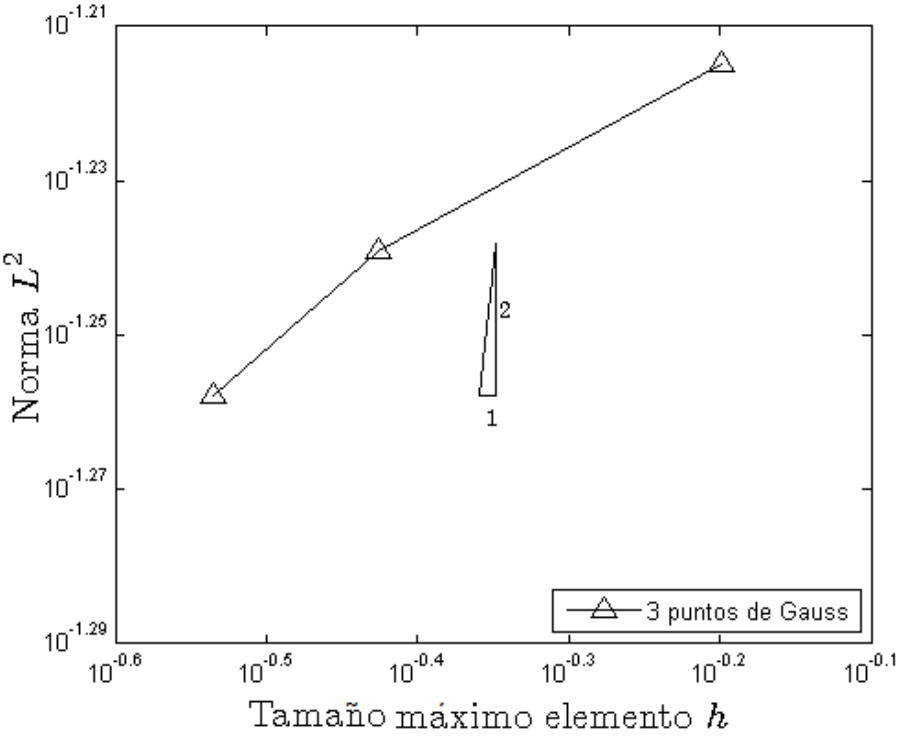

Figura 4.12 Norma relativa L^2 para problema estrato infinito.

Tabla 4.3 Resultados normas relativas H^1 y L^2 para problema estrato infinito.

Malla utilizada	Nodos	Norma relativa H^1	Norma relativa L^2
Malla 1	209	$1.788 \cdot 10^{-1}$	$6.103 \cdot 10^{-2}$
Malla 2	446	$1.559 \cdot 10^{-1}$	$5.774 \cdot 10^{-2}$
Malla 3	1013	$1.492 \cdot 10^{-1}$	$5.520 \cdot 10^{-2}$

El tiempo de cómputo, Speedup y Eficiencia paralela se observan en la Figura 4.13, 4.14 y 4.15, respectivamente.

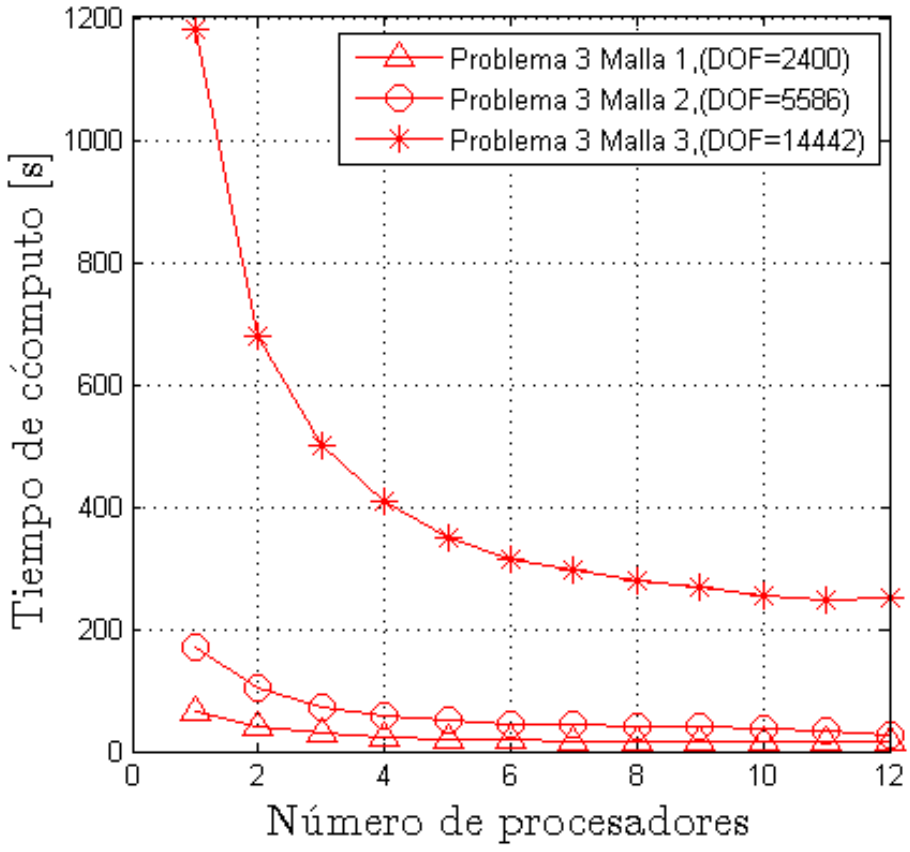

Figura 4.13 Tiempo de cómputo en función del número de procesadores involucrados

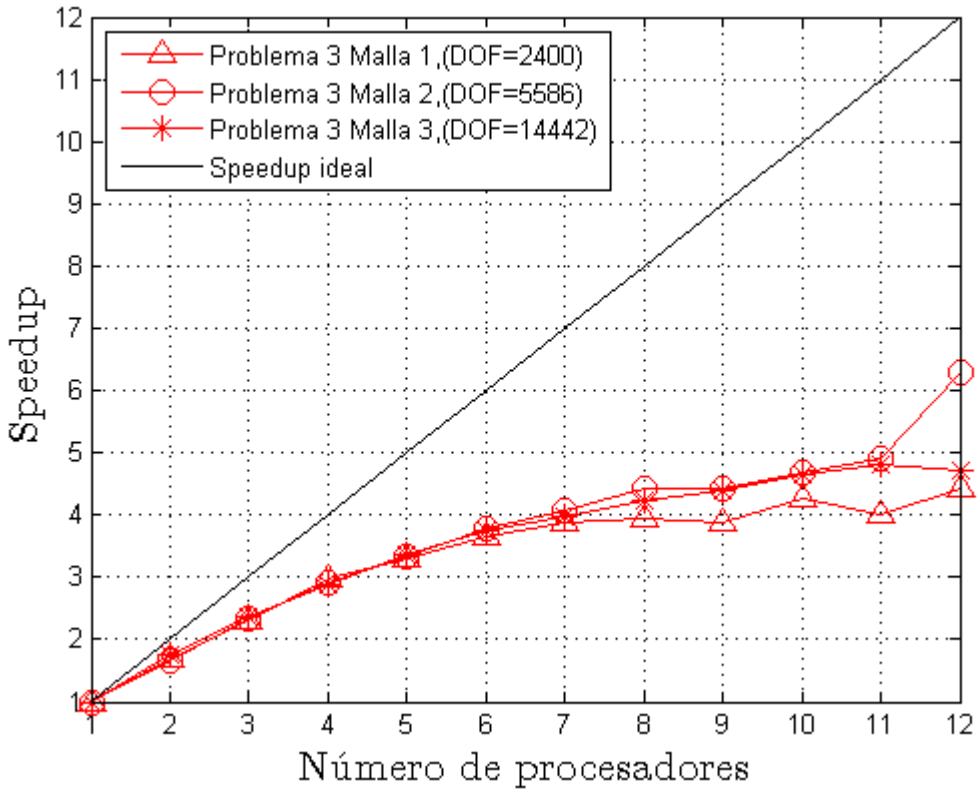

Figura 4.14 Speedup para distintas mallas utilizadas. Se compara con la situación ideal (en negro).

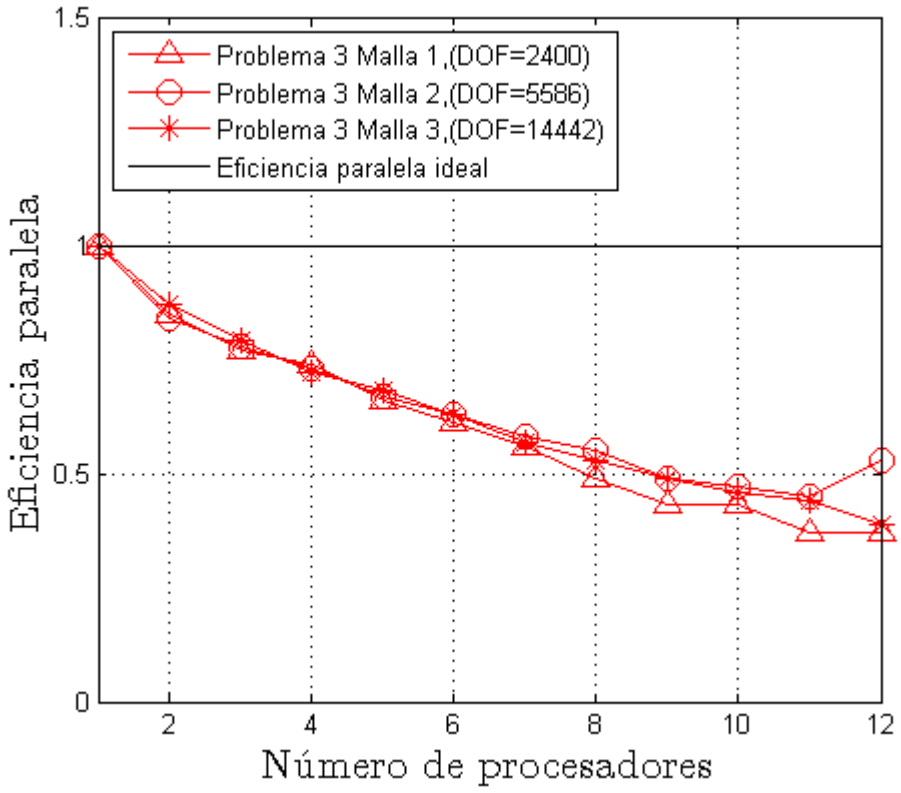

Figura 4.15 Eficiencia paralela para distintas mallas utilizadas. Se compara con la situación ideal (en negro).

4.4 Análisis e interpretación de resultados

Se analiza e interpreta los resultados obtenidos para los tres problemas. Se estudia los resultados del error numérico y convergencia, luego tiempo de cómputo, Speedup y Eficiencia paralela.

4.4.1 Test de la parcela

Es posible observar de las Figuras 4.1, 4.6 y 4.11, que se consigue la convergencia óptima de la norma H^1 para los problemas 1 y 2 y para el problema 3 se obtiene convergencia menor a la ideal. Lo mismo ocurre para el caso de la norma L^2 (ver Figuras 4.2, 4.7 y 4.12). En el problema 3 se obtiene una convergencia menor debido a problemas de integración numérica, esto ocurre al observar que solo posee tres puntos de Gauss en su integración.

Se validan los tres modelos al compararlos con su solución analítica, las normas obtenidas se resumen en las Tablas 4.1, 4.2 y 4.3.

4.4.2 Tiempo de cómputo

Se obtienen tiempos de cómputo en las Figuras 4.3, 4.8 y 4.13, para los problemas uno, dos y tres, respectivamente.

Los tiempos son cada vez menores a medida que se utilizan más procesadores y esto ocurre en todas las situaciones, por tanto, se logra siempre una reducción al utilizar programación paralela.

4.4.3 Speedup y Eficiencia Paralela

El Speedup se observa en las Figuras 4.4, 4.9 y 4.14. Se grafican todos los resultados en la Figura 4.16. Se resume en la Tabla 4.4 los valores de Speedup utilizando 12 procesadores, para los tres problemas.

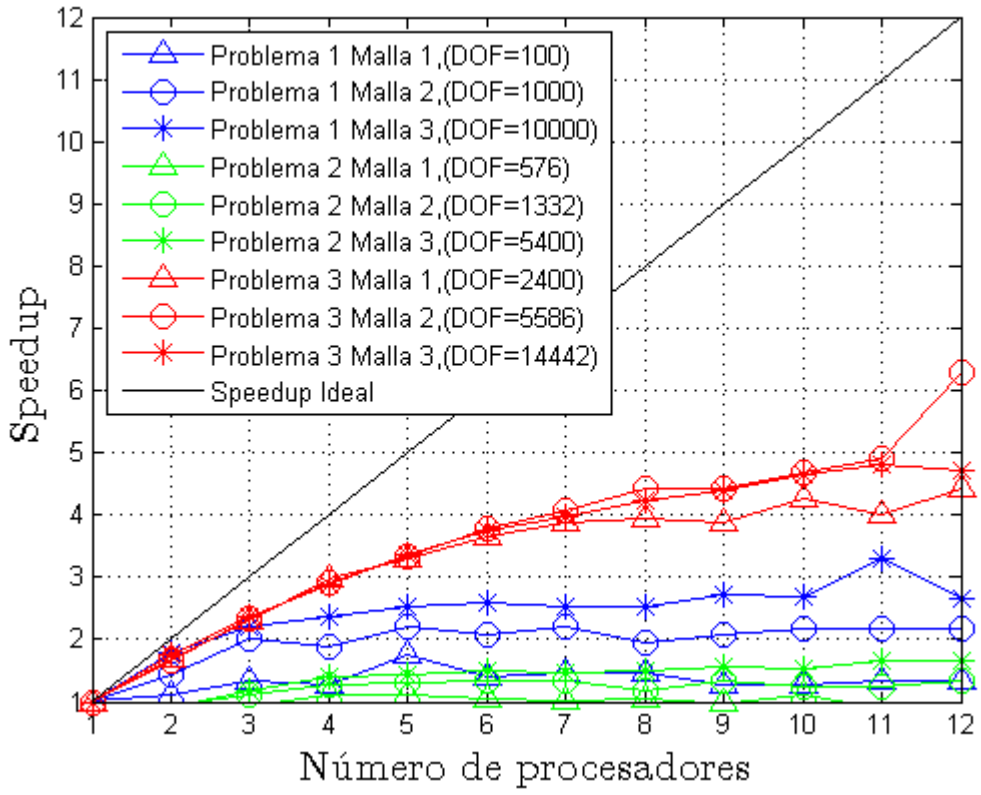

Figura 4.16 Speedup de todos los problemas realizados.

Tabla 4.4 $Speedup_{12}$ para problemas uno, dos y tres, utilizando 12 procesadores.

	Grados de libertad del problema	$Speedup_{12}$
Problema 1	100	1.33
	1000	2.17
	10000	2.67
Problema 2	576	0.85
	1332	1.34
	5400	1.66
Problema 3	2400	4.42
	5586	6.30
	14442	4.71

Es posible observar, que se alcanza Speedup superiores a 1 cuando se trabaja sobre 1000 grados de libertad (DOF). Los resultados mejoran al aumentar los DOF, independiente del problema, esto se observa en la Tabla 4.4 al visualizar los Speedup para 12 procesadores.

La Eficiencia paralela se visualiza en las Figuras 4.5, 4.10 y 4.15. Se grafican todos los resultados en la Figura 4.17. Se resume en la Tabla 4.5 los valores de Eficiencia paralela utilizando 12 procesadores, para los tres problemas.

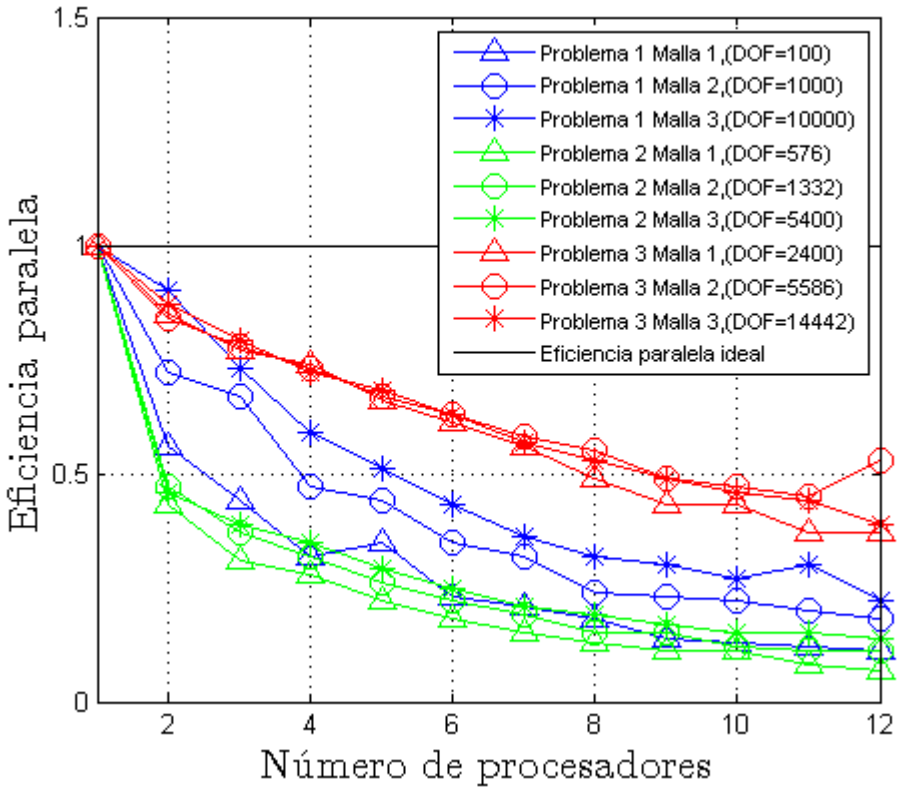

Figura 4.17 Eficiencia paralela de todos los problemas resueltos.

Tabla 4.5 $Eficiencia\ paralela_{12}$ para problemas uno, dos y tres, utilizando 12 procesadores

	Grados de libertad del problema	$Eficiencia\ paralela_{12}$
Problema 1	100	11%
	1000	18%
	10000	22%
Problema 2	576	7%
	1332	11%
	5400	14%
Problema 3	2400	37%
	5586	53%
	14442	39%

Los resultados mejoran al aumentar el número de grados de libertad (DOF) para los tres problemas. Se obtienen eficiencias sobre un 14% para problemas por sobre los 5000 DOF.

5 Discusión de resultados

Esta Sección se organiza como sigue. Se discute los resultados de las normas relativas H^1 y L^2 en 5.1, el tiempo de cómputo en 5.2 y Speedup y Eficiencia paralela en 5.3. Por último, en 5.4 se presentan las recomendaciones al programar en paralelo problemas resueltos mediante el método sin malla.

5.1 Test de la parcela.

Se obtienen errores numéricos pequeños, lo que valida los problemas resueltos. La convergencia se asegura para los tres problemas, alcanzando convergencia ideal para los problemas uno y dos. El problema tres no la consigue debido a errores en integración numérica, con solo tres puntos de Gauss por elemento.

5.2 Tiempo de cómputo

Se logra una disminución del tiempo de cómputo a medida que se utilizan más procesadores, lo que demuestra la eficacia de la programación en paralelo, sin embargo, la tasa de decaimiento no es lineal con respecto a los procesadores involucrados, y esto se acentúa a medida que los grados de libertad aumentan.

Los tiempos de cómputo aquí obtenidos dependen del cluster utilizado y las condiciones ambientales en la que este estaba (exceso de temperatura del computador alarga considerablemente los tiempos de cómputo). De tener grandes DOF, se alargan los tiempos de cómputo e inevitablemente se calientan los procesadores, lo que afecta la eficacia con que computa y esto produce errores en los resultados. Esto se evidencia al ver la Eficiencia paralela en el problema tres, al pasar de 5586 a 14442 DOF (Ver Tabla 4.5).

5.3 Speedup y Eficiencia paralela

El Speedup muestra cuanto se acelera el cómputo, donde idealmente es proporcional al número de procesadores involucrados. De los resultados obtenidos, es posible notar una mejora a medida que crece el número de DOF, o de otra forma, iteraciones. Esto ocurre principalmente por el fenómeno de *overhead*, el cual consiste en que existe una comunicación entre procesadores de sincronización, lo que genera tiempos de inactividad para algunos procesadores debido a la carga desequilibrada. Este tiempo de *overhead* toma un rol principal en el Speedup para problemas con pocos DOF, esto se evidencia en el problema de la viga bidimensional comparado con el problema del estrato infinito.

Lo anterior se valida, al contrastar los resultados obtenidos con lo publicado por el Sr. Z. Ullah en [19], en la Figura 5.1. Existe una proporción directa entre los grados de libertad del problema y el Speedup.

En la investigación, se desarrolla el problema de la viga bidimensional con 3782 y 14762 grados de libertad, mediante el método sin malla Galerkiano mediante funciones de base de máxima entropía. El cómputo se realizo en el Hamilton cluster, de la Universidad de Glasgow, y sus datos siguen la misma tendencia aquí obtenida.

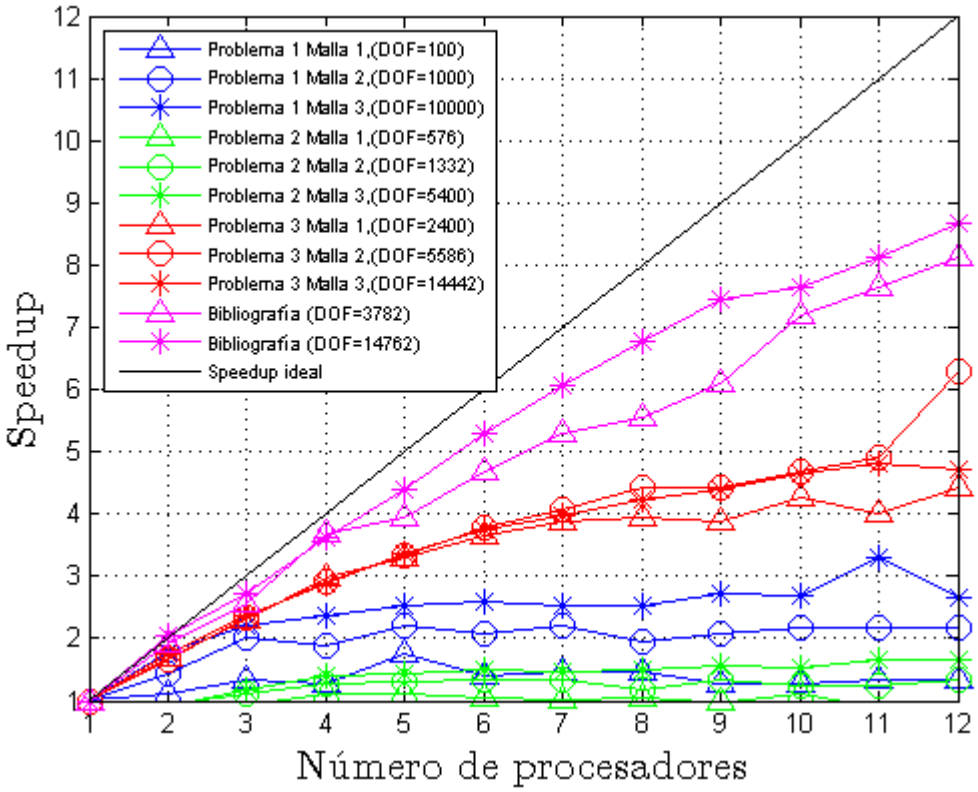

Figura 5.1 Resultados de Speedup comparados con los obtenidos en bibliografía [Fuente: Adaptado de [19]]

Por otro lado, la Eficiencia paralela indica que tan eficiente es el cómputo con respecto al número de procesadores involucrados, idealmente se tiene una eficiencia de un 100% pero esto nunca ocurrirá debido al fenómeno de *overhead*.

Al comparar las eficiencias obtenidas con los resultados en bibliografía, en la Figura 5.2, se puede observar nuevamente que existe un aumento de la eficiencia conforme los DOF aumentan, esto también se evidencia en la Tabla 4.5.

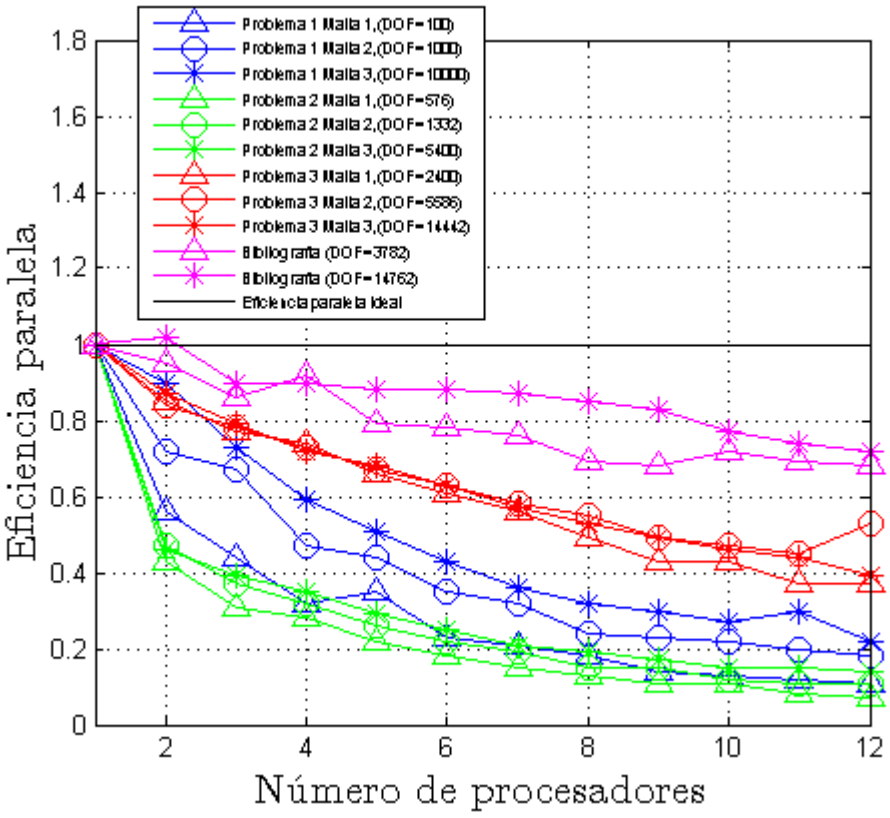

Figura 5.2 Resultados de Eficiencia paralela comparados con los obtenidos en bibliografía [Fuente: Adaptado de [19]].

5.4 Recomendaciones

A partir de los resultados obtenidos, se evidencia la relación entre DOF y la mejora de los indicadores de programación en paralelo (Speedup y Eficiencia paralela). A partir de 5000 DOF para problemas en 1D, 2D y 3D se observan mejoras significativas en estos indicadores (ver Tablas 4.4 y 4.5).

Por lo tanto, se recomienda programar paralelamente problemas con un mínimo de 5000 DOF, para obtener Eficiencias paralelas y Speedup significativos.

Para problemas en tres dimensiones con más de 10000 DOF, los resultados podrían ser afectados por la temperatura que generan los ordenadores, debido principalmente a que se toma mayor tiempo en su cómputo. Se recomienda tomar medidas contra esto para mayor eficiencia en la programación, como instalación de sistemas de ventilación u otra refrigeración del cluster.

6 Conclusiones

Los resultados obtenidos demuestran la reducción de los tiempos de cómputo al implementar la programación en paralelo, obteniendo mejores tiempos al disminuir el efecto de *overhead*, el cual disminuye para problemas con mayores grados de libertad (DOF).

El número de procesadores no afecta el orden de la aproximación, lo cual es significativo, ya que implica que paralelizar un problema solo conlleva a la disminución del tiempo de cómputo.

Se recomienda trabajar problemas con más de 5000 DOF al implementar programación en paralelo en el método sin malla Galerkiano.

Se concluye que se cumple objetivo principal, logrando realizar algoritmo numérico mediante programación en paralelo, para la resolución de problemas en la mecánica de sólidos mediante el método sin malla Galerkiano con funciones de base de máxima entropía. Esto se lleva a cabo al cumplir los objetivos específicos, los cuales son:

Implementación programación paralelo problema barra elástica

Implementación programación paralelo problema viga en voladizo

Implementación programación paralelo problema estrato infinito

Analizar tiempo cómputo, Speedup y eficiencia en programación paralelo.

7 Bibliografía

[1] Belytschko, T., Lu, Y. Y., and Gu, L. (1994), "Element-free Galerkin methods". International Journal for Numerical Methods in Engineering, 37:229-256

[2] A. Ortiz-Bernardin. Elementos Finitos Generalizado ME705 Universidad de Chile. Apuntes del curso. Semestre Otoño 2015.

[3] Pacheco, P. S. (1997), Parallel Programming with MPI. Morgan Kaufmann Publishers, Inc.

[4] G.R. Liu, Y.T Gu, "An introduction to Meshfree Methods and Their Programming". Springer ,2005.

[5] Grosso, A. D. and Righetti, G. (1988), "Finite element techniques and artificial intelligence on parallel machines". Computers & Structures, 30(4):999 –1007

[6] Becene, A. T. (2003), Parallel processing of finite strain, materially nonlinear and incompressible finite element analysis problems. PhD thesis, University of Rochester.

[7] Yagawa, G., Soneda, N., and Yoshimura, S. (1991), "A Large scale finite element analysis using domain decomposition method on a parallel computer". Computers & Structures, 38:615 – 625.

[8] Luo, J. C. and Friedman, M. B. (1990), "A parallel computational model for the finite element method on a memory-sharing multiprocessor computer". Computer Methods in Applied Mechanics and Engineering, 84(2):193 – 209.

[9] Chiang, K. N. and Fulton, R. E. (1990), "Concepts and implementation of parallel finite element analysis". Computers & Structures, 36(6):1039 – 1046.

[10] Carter, W. T., Sham, T. L., and Law, K. H. (1989), "A parallel finite element method and its prototype implementation on a hypercube". Computers & Structures, 31(6):921 – 934.}

[11] Shannon CE. A mathematical theory of communication. The Bell Systems Technical Journal 1948; 27:379–423.

[12] Jaynes ET. Information theory and statistical mechanics. Physical Review 1957

[13] Jaynes ET. Information theory and statistical mechanics. II. Physical Review 1957

[14] N. Sukumar. Construction of polygonal interpolants: a maximum entropy approach. INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING. 2004 61:2159–2181

[15] J. Kepner. Parallel MATLAB for Multicore and Multinode Computers. SIAM.,Massachusetts, first edition, 2009.

[16] MATHWORK. Parallel Computing Toolbox User's guide. The Mathwork inc. 2015

[17] S. P. Timoshenko and J. N. Goodier. Theory of Elasticity. McGraw-Hill, NY, third edition, 1970.

[18] Q. Duan, X. Gao, B. Wang, X. Li, and H. Zhang. A four-point integration scheme with quadratic exactness for three-dimensional element-free Galerkin method based on variationally consistent formulation. Computer Methods in Applied Mechanics and Engineering, 2014.

[19] Ullah, Z., Augarde, C.E., and Coombs, W.M. Parallel computations in nonlinear solid mechanics using adaptive finite element and meshless methods accepted for Engineering Computations. Engineering Computations , 2015.

Anexos

Anexos A. Codigo MATLAB problema #1

```
function  [K,uh,fg]= parte1_paralela( nelementos,npgauss )
%Desarrollo de resolución problema barra 1D trabajo "Paralelización de
%algoritmo numerico para la resolución de problema en la mecanica de
%solidos" por Ruben Dario Torres.

matlabpool open

tic
%% DATOS DEL PROBLEMA

E=1e7;  % Modulo de Young [psi]
L=8;    % Largo barra [in]
t=100000;  % Esfuerzo en la punta [psi]
A=1;      % Area seccion transversal [in2]


%  FABRICAR VECTOR NODOS Y MATRIZ DE CONECTIVIDAD 1D

dl=L/nelementos;    % largo del elemento

nodos=zeros(nelementos+1,1);        %Vector de nodos de la forma
[x1;x2;x3;...]

for i=1:nelementos
    nodos(i+1)=dl*(i);

end


% INICIALIZACION DE PARAMETROS PARA CUADRATURA DE GAUSS

%%puntos de gauss
xgauss_mapeado=gausspoints1d(npgauss);      %Coordenadas de los ptos de gauss
mapeado (en dominio [-1,1])
wgauss=gaussweights1d(npgauss);                 % funciones de pesos de los
puntos de gauss

%%Parametros para funcion maxent
dim=1;     %Dimension 1D
n=length(nodos);  %numero de nodos
ilambda=0;        %factor que se entrega a funcion maxent, 2D [0;0] , 1D [0]
h_node=nodespacing(dim,n,nodos);

%%Matriz de rigidez

K=zeros(n);         %Matriz de rigidez global
fg=zeros(n,1);      %vector de fuerza
fg(n)=A*t;
```

```
%% CUADRATURA DE GAUSS

parfor elemento=1:nelementos
    % Empezamos la sumatoria con respecto a los elementos, inicializamos
    % matriz de rigiz elemental y vector de fuerza elemental

    Ke=zeros(n);
    x1=nodos(elemento);
    x2=nodos(elemento+1);
    del=(x2-x1)./2;
    xpg=x1+del.*xgauss_mapeado+del;    %Cambio de coordenadas mapeadas a car-
tesianas para puntos de gauss



    for pg=1:length(xpg)
        %la sumatoria ahora sigue por punto de gauss
        x=xpg(pg);
        [phi,phider,contribute]=maxent(h_node,dim,n,nodos,x,ilambda);   %Ob-
tengo funciones phi, dephi y contribute(nodos) que contribuyen por pto gauss

        Be=phider'; %Vector Be de la forma [phider1 phider2 phider3...] con el
largo dependiente de cuantos contribuyentes arroje maxent
        Ne=phi';    %Vector de las funciones de base  [base1 base2 ...] con el
largo dependiente de cuantos contribuyentes arroje maxent

        Kpg=Be'*(dl/2)*E*Be*wgauss(pg);  %Matriz de rigidez por pto de gauss

        %% ENSAMBLE DE MATRIZ RIGIDEZ Y VECTOR DE FUERZA

        for c1=1:length(contribute)
            for c2=1:length(contribute)

Ke(contribute(c1),contribute(c2))=Ke(contribute(c1),contribute(c2))+Kpg(c1,c2)
;
            end
        end




    end

    K=K+Ke;


end

%% Condiciones de borde

for i=1:length(nodos)
    K(1,i)=0;
    K(i,1)=0;
```

```
end
K(1,1)=1;        %% CONDICION DE BORDE FIJO u(0)=0; (ya se tiene que
f(0)=u(0)=0 )




uh=K\fg;


toc

end
```

Anexo B. Codigo MATLAB problema #2

```
%% Script Parte 2 - Trabajo de titulo Paralelización problema algoritmo núme-
rico

%Desarrollo de resolución problema viga 2D trabajo "Paralelización de
%algoritmo numerico para la resolución de problema en la mecanica de
%solidos" por Ruben Dario Torres

clear all
close all
clc


tic

%% DATOS DEL PROBLEMA

E = 1e7;
%%Módulo de Young
v = 0.3;
%%Coeficiente de Poisson
v2 = v./(1-v);
C=((E*(1-v))/((1+v)*(1-2*v)))*[1 v2 0; v2 1 0; 0 0 (1-2*v)/(2*(1-v))];
%%Matriz para estado de deformación plana
E2 = E./(1-v.^2);
P = -1000000;
%%Fuerza (Lb)
L = 8;
%%Largo de la viga (in)
I = 4^3/12;
%%Momento de Inercia (in^4)
D = 4;
%%Altura de la viga (in)
mu=E./(2.*(1+v));

% MALLA
path = 'mesh2.txt';
Malla = importdata(path,' ',2);
Nodos = Malla.data; %Vector de forma [x1 y1;x2 y2;...;xn yn] que posee las
coordenadas de los nodos


Malla2 = importdata(path,' ',length(Nodos)+4);
Conect = [Malla2.data(:,1),Malla2.data(:,4:6)];       %Matriz de conectividad
de la forma [n°elemento, n°nodo1, n°nodo2,n°nodo3;...]
gamma= 1.58;

%% INICIALIZACION DE PARAMETROS CUADRATURA DE GAUSS


aux = [1 3 6 6 7 12 13 16 19 25 27 33 37 42];   %Puntos de gauss dependiendo
del orden de gausspointst3t4
order = 2;        %%Orden para Gausspoints - Número de puntos de gauss para la
integración númerica.
```

```
%Parametros globales
n=length(Nodos); %N° de nodos que existen en la malla
K=zeros(2*n); %matriz global de rigidez K


%Parametros para función maxent
ncoord = Nodos;
dim = 2;
ilambda = [0;0];
h_node=nodespacing(dim,n,ncoord);

%% CUADRATURA DE GAUSS

 w1= waitbar(0,['Malla 2 - Matriz K - orden ',int2str(order),' \gamma
',num2str(gamma)]);  %Barra de espera
parfor cont1 = 1:Conect(end,1)

         waitbar(cont1 / Conect(end,1));

    %Inicializo matriz de rigidez elemental
    Ke=zeros(2*n);

    Nel = [Conect(cont1,2),Conect(cont1,3),Conect(cont1,4)];   %Nodos del ele-
mento que recorro
    Elem = [Nodos(Nel(1),1:2);Nodos(Nel(2),1:2);Nodos(Nel(3),1:2)];   %Vector
de las coordenadas de los Nodos de la forma [x1 y1;x2 y2;x3 y3]
    [pe,we] = gausspointsT3T4(order, Elem);
    lpg = length(pe(:,1));

    for cont2 = 1:lpg

        x = pe(cont2,:);
%%Punto de gauss en el que estoy parado
        [phi,phider,ptos]=maxent(h_node,dim,n,ncoord,x,ilambda);    %%Obtener
las matrices Phi y phider y contribute por punto de gauss
        B = zeros(3,2*length(phider(:,1)));

        for cont3 = 1:length(phider(:,1))
            B(:,(2*cont3-1):2*cont3) = [phider(cont3,1), 0; 0, phi-
der(cont3,2);phider(cont3,2),phider(cont3,1)];        %%Ensamble de la matriz
B a partir de los phider encontrados para cada punto contribuidor
        end

        Kpg = B'*C*B.*we(cont2);                                          %%
Matriz K para el punto de gauss leido.

        for c1 = 1:length(ptos)
            for c2 = 1:length(ptos)
                Ke(2*ptos(c1)-1:2*ptos(c1),2*ptos(c2)-1:2*ptos(c2)) =
Ke(2*ptos(c1)-1:2*ptos(c1),2*ptos(c2)-1:2*ptos(c2)) + Kpg(2*c1-1:2*c1,2*c2-
1:2*c2);
            end
        end
```

```
    end

      K = K + Ke;

end

% close(w1)

%% CALCULAR VECTOR DE FUERZA

%Vector fuerza
F = zeros(2*n,1);

% Buscamos los nodos que se encuentran en el borde, estos reciben la fuerza
% vertical distribuida parabolicamente.
borde(:,1)= find(Nodos(:,1) == 8)';
borde(:,2:3) = Nodos(Nodos(:,1) == 8,1:2);

force = -P./4.*borde(:,3).^2+P;
xgauss=gausspoints1d(20);
wgauss=gaussweights1d(20);

%% DATOS MAXENT

dim = 1;
n2 = length(borde(:,3));
ncoord2 = borde(:,3);
ilambda = 0;
h_node2=nodespacing(dim,n2,ncoord2);

%% CUADRATURA DE GAUSS EN 1D BORDE
for cont1 = 1:n2-1

    Fe = zeros(2*n2,1);
    y1 = borde(cont1,3);
    y2 = borde(cont1+1,3);
    del = (y2-y1)./2;
    ypg = y1+del+xgauss.*del;

    for cont2 = 1:length(ypg)

        [phi2,phider2,ptos2] =
maxent(h_node2,dim,n2,ncoord2,ypg(cont2),ilambda);
        tf = [0;  del.*P.*((D.^2/4)-ypg(cont2).^2)/(2.*I)];
        N = zeros(2,2*length(phi2));

        for cont3 = 1:length(ptos2)
            N(:,(2*cont3-1):2*cont3) = [phi2(cont3), 0 ; 0, phi2(cont3)];
        end
        Fpg = N'*tf.*wgauss(cont2);

        for cont4 = 1:length(ptos2)
            Fe(2*ptos2(cont4),1) = Fe(2*ptos2(cont4),1) + Fpg(2*cont4,1);
        end
```

```
    end

    for cont5 = 1:n2
        F(2*borde(cont5,1),1) = F(2*borde(cont5,1),1) + Fe(2*cont5,1);
    end
end

%% CONDICIONES DE BORDE

borde0(:,1) = find(Nodos(:,1) == 0)';
borde0(:,2:3) = Nodos(Nodos(:,1) == 0,1:2);


for cont1 = 1: length(borde0(:,1))

    ux = (-P.*borde0(cont1,3)./(6.*E2.*I)).*((2+v2).*borde0(cont1,3).^2-
(3/2).*D.^2.*(1+v2));
    uy = (P./(6.*E2.*I)).*(3.*v2.*borde0(cont1,3).^2.*(L-borde0(cont1,2)));

    F = F - K(:,borde0(cont1,1)*2-1).*ux - K(:,borde0(cont1,1)*2).*uy;

    F(borde0(cont1,1)*2-1:borde0(cont1,1)*2,1) = [ux;uy];


    K(:,borde0(cont1,1)*2-1:borde0(cont1,1)*2) = zeros(length(K),2);
    K(borde0(cont1,1)*2-1:borde0(cont1,1)*2,:) = zeros(2,length(K));
    K(borde0(cont1,1)*2-1:borde0(cont1,1)*2,borde0(cont1,1)*2-
1:borde0(cont1,1)*2) = [1 0;0 1];

end
%% Solucion

uh = K\F;

toc


%% COMPARACION SOLUCION ANALITICA

%% Solucion Analitica
Uf = zeros(length(Nodos),1);
Ux = (-P.*Nodos(:,2)./(6.*E2.*I)).*((6.*L-
3.*Nodos(:,1)).*Nodos(:,1)+(2+v2).*Nodos(:,2).^2-(3/2).*D.^2.*(1+v2));
Uy = (P./(6.*E2.*I)).*(3.*v2.*Nodos(:,2).^2.*(L-Nodos(:,1)) + (3.*L-
Nodos(:,1)).*Nodos(:,1).^2);

for cont1 = 1:length(Nodos)
    Uf(2*cont1-1:2*cont1,1) = [Ux(cont1);Uy(cont1)];
end

%% normas

dim = 2;
ilambda = [0;0];
```

```
N1a = 0;
N1b = 0;
N2a = 0;
N2b = 0;

% w2= waitbar(0,['Malla 2 - Normas - orden ',int2str(order),' \gamma
',num2str(gamma)]);

for cont1 = 1:Conect(end,1)
    %% Integración númerica por elemento.
%     waitbar(cont1 / Conect(end,1));

    Nel = [Conect(cont1,2),Conect(cont1,3),Conect(cont1,4)];
    Elem = [Nodos(Nel(1),1:2);Nodos(Nel(2),1:2);Nodos(Nel(3),1:2)];
    [pe,we] = gausspointsT3T4(order, Elem);
    lpg = length(pe(:,1));

    for cont2 = 1:length(pe(:,1))
        %%
        %         phi3 = PhiMAX(PhiMAX(:,2)==(cont1-1)*aux(order)+cont2,4);
        %         ptos3 = PhiMAX(PhiMAX(:,2)==(cont1-1)*aux(order)+cont2,3);
        %         phider3 = PhiderMAX(PhiderMAX(:,2)==(cont1-
1)*aux(order)+cont2,4:5);
        x = pe(cont2,:);
        [phi3,phider3,ptos3] = maxent(h_node,dim,n,ncoord,x,ilambda);

        uhpg = zeros(2*length(ptos3),1);
        N = zeros(2,2*length(phi3));
        Be = zeros(3,2*length(phider3));

        %%
        for cont3 = 1:length(ptos3)
            N(:,(2*cont3-1):2*cont3) = [phi3(cont3), 0 ; 0, phi3(cont3)];
            Be(:,(2*cont3-1):2*cont3) = [phider3(cont3,1), 0; 0, phi-
der3(cont3,2);0.5.*phider3(cont3,2),0.5.*phider3(cont3,1)];
            uhpg(2*cont3-1:2*cont3) = uh(2*ptos3(cont3)-1:2*ptos3(cont3),1);

        end

        Uhpg = N*uhpg;

        ux = (-P.*pe(cont2,2)./(6.*E2.*I)).*((6.*L-
3.*pe(cont2,1)).*pe(cont2,1)+(2+v2).*pe(cont2,2).^2-(3/2).*D.^2.*(1+v2));
        uy = (P./(6.*E2.*I)).*(3.*v2.*pe(cont2,2).^2.*(L-pe(cont2,1)) + (3.*L-
pe(cont2,1)).*pe(cont2,1).^2);
        Upg = [ux;uy];

        eh = Be*uhpg;
        Sh = C*eh;

        ex = -(P.*(L-pe(cont2,1)).*pe(cont2,2))./(E2.*I);
        ey = (v2.*P.*(L-pe(cont2,1)).*pe(cont2,2))./(E2.*I);
        exy = (P.*(D.^2./4-pe(cont2,2).^2))./(4.*I.*mu);

        sx = (-P.*(L-pe(cont2,1)).*pe(cont2,2))./I;
        sy = 0;
```

```
        sxy = (P.*(D.^2./4-pe(cont2,2).^2))./(2.*I);

        Sf =[sx; sy; sxy];
        ef =[ex; ey; exy];


        N1a = N1a + dot(Upg - Uhpg,Upg - Uhpg).*we(cont2);
        N1b = N1b + dot(Upg,Upg).*we(cont2);

        N2a = N2a + dot(ef-eh,Sf-Sh).*we(cont2);
        N2b = N2b + dot(ef,Sf).*we(cont2);


    end

end

% close(w2)

Norma1 = N1a.^(0.5)./N1b.^(0.5)
Norma2 = N2a.^(0.5)./N2b.^(0.5)


toc
```

Anexo C. Codigo MATLAB problema #3

```
%% Script Parte 3 - Trabajo de titulo Paralelización problema algoritmo núme-
rico

%Desarrollo de resolución problema estrato infinito 3D trabajo "Paralelización
de
%algoritmo numerico para la resolución de problema en la mecanica de
%solidos" por Ruben Dario Torres

clear all
close all
clc


tic

%% DATOS DEL PROBLEMA

E=4e7;  %Modulo de Young [Pa]
v=0.3;  %Coeficiente de poisson
g=9.8;  %Aceleración de gravedad [m/s2]
p=7500; %Densidad del material [kg/m3]
G=0.5*(E/(1+v));
kk=v*E/((1+v)*(1-2*v));
aa=v*E/((1+v)*(1-2*v));
C=[kk+2*G kk kk 0 0 0;kk kk+2*G kk 0 0 0;kk kk kk+2*G 0 0 0;0 0 0 G 0 0;0 0 0
0 G 0;0 0 0 0 0 G]; %Matriz constitutiva 3D (material homogeneo, elastico,
lineal e isotropo)


%Malla

Nodos=importdata('Nodos_malla2.txt');       %Nodos de la forma
[x1,y1,z1;x2,y2,z2;...]
Conect=importdata('Elementos_malla2.txt');      %Matriz de conectividad de la
forma [#elemento,nodo1,nodo2,nodo3,nodo4;...]
Conect2=zeros(length(Conect(:,1)),4);
n=length(Conect);

for j=1:n
Co-
nect2(j,:)=[Nodos(Conect(j,2),2),Nodos(Conect(j,3),2),Nodos(Conect(j,4),2),Nod
os(Conect(j,5),2)];
end


    %% INICIALIZACION DE PARAMETROS CUADRATURA DE GAUSS

n=length(Nodos);    %Número de nodos
K=zeros(3*n); %matriz global de rigidez K
order=2;
%Parametros funcion MAXENT

ncoord=Nodos;
```

```
dim=3;
ilambda=[0;0;0];
h_node=nodespacing(dim,n,ncoord);

%% CUADRATURA DE GAUSS

parfor cont1=1:Conect(end,1)   %Voy por elemento

    Ke=zeros(3*n);
    Nel=[Conect(cont1,2),Conect(cont1,3),Conect(cont1,4),Conect(cont1,5)];
%Nodos del elemento al que recorro de la forma [nodo1,nodo2,nodo3,nodo4]

Elem=[Nodos(Nel(1),1:3);Nodos(Nel(2),1:3);Nodos(Nel(3),1:3);Nodos(Nel(4),1:3)]
; %Coordenadas de los nodos del elemento al que recorro de la forma
[x1,y1,z1;...;x4,y4,z4]
    [pe,wg]=gausspointsT3T4(order, Elem);
    lpg = length(pe(:,1));

    for cont2=1:lpg

        x=pe(cont2,:);       %%Punto de gauss en el que estoy parado
        [phi,phider,ptos]=maxent(h_node,dim,n,ncoord,x,ilambda);    %%Obtener
las matrices Phi y phider y contribute por punto de gauss
        B=zeros(6,3*length(phider(:,1)));

        for cont3=1:length(phider(:,1))
            B(:,3*cont3-
2:3*cont3)=[phider(cont3,1),0,0;0,phider(cont3,2),0;0,0,phider(cont3,3);phider
(con-
t3,2),phider(cont3,1),0;0,phider(cont3,3),phider(cont3,2);phider(cont3,3),0,ph
ider(cont3,1)];
        end

        Kpg=B'*C*B.*wg(cont2);

        for c1 = 1:length(ptos)
            for c2 = 1:length(ptos)
                Ke(3*ptos(c1)-2:3*ptos(c1),3*ptos(c2)-2:3*ptos(c2)) =
Ke(3*ptos(c1)-2:3*ptos(c1),3*ptos(c2)-2:3*ptos(c2)) + Kpg(3*c1-2:3*c1,3*c2-
2:3*c2);
            end
        end

    end

     K = K + Ke;

end

%% CALCULAR VECTOR DE FUERZAS

%Vector de fuerza
F=zeros(3*n,1);

%Buscamos los nodos que se encuentran en la superficie superior, estos
```

```
%reciben la fuerza de compresión.

borde(:,1)=find(Nodos(:,2)==1)';    %borde forma [#no-
do,nodo1,nodo2,nodo3,nodo4;...]
borde(:,2:4)=Nodos(Nodos(:,2) == 1,1:3);

q=1;

for i=1:length(Conect(:,1))
    if (((Co-
nect2(i,1)==1)||(Conect2(i,2)==1))&&((Conect2(i,1)==1)||(Conect2(i,3)==1))&&((
Co-
nect2(i,1)==1)||(Conect2(i,4)==1))&&((Conect2(i,2)==1)||(Conect2(i,3)==1))&&((
Conect2(i,2)==1)||(Conect2(i,4)==1))&&((Conect2(i,3)==1)||(Conect2(i,4)==1)))
    kk=find(1==Conect2(i,:));
    Nel2(q,1)=i;
    Nel2(q,2:4)=[Conect(i,kk(1)+1),Conect(i,kk(2)+1),Conect(i,kk(3)+1)];
    q=q+1;
    end
end




force=-1000000;


% Datos Maxent

dim=2;
n2=length(borde);
ncoord2=[borde(:,2),borde(:,4)];  % coordenadas ncoord2= [x1 z1;x2 z2;...]


ilambda=[0;0];
h_node2=nodespacing(dim,n2,ncoord2);

%% CUADRATURA DE GAUSS EN 2D SUPERFICIE

for cont1=1:length(Nel2)   %voy por elemento

    Fe=zeros(3*n2,1);

Elem2=[Nodos(Nel2(cont1,2),1),Nodos(Nel2(cont1,2),3);Nodos(Nel2(cont1,3),1),No
dos(Nel2(cont1,3),3);Nodos(Nel2(cont1,4),1),Nodos(Nel2(cont1,4),3)];
    [xgauss,wgauss]=gausspointsT3T4(order,Elem2);
    lpg = length(xgauss(:,1));

    for cont2=1:lpg

        x=xgauss(cont2,:);
        [phi2,phider2,ptos2]=maxent(h_node2,dim,n2,ncoord2,x,ilambda);
        tf=[0;force;0];
        N=zeros(3,3*length(phi2));
```

```
        for cont3=1:length(ptos2)
            N(:,3*cont3-
2:3*cont3)=[phi2(cont3),0,0;0,phi2(cont3),0;0,0,phi2(cont3)];
        end

        Fpg=N'*tf.*wgauss(cont2);

        for cont4=1:length(ptos2)
            Fe(3*ptos2(cont4)-1,1)=Fe(3*ptos2(cont4)-1,1)+Fpg(3*cont4-1,1);
        end
    end

    for cont5=1:n2
        F(3*borde(cont5,1)-1,1) = F(3*borde(cont5,1)-1,1) + Fe(3*cont5-1,1);
    end

end

%% CONDCIONES DE BORDE

borde0(:,1)=find(Nodos(:,2)==-1)';           %borde0 de la forma [#elemento
nodo1 nodo2 nodo3 nodo4;...]de nodos base plano x-z
borde0(:,2:4)= Nodos(Nodos(:,2) == 1,1:3);

borde1(:,1)=find(Nodos(:,1)==-1)';           %borde1 de la forma [#elemen-
to,nodo1,nodo2,nodo3,nodo4;...] de nodos plano x-y, x=-1
borde1(:,2:4)=Nodos(Nodos(:,1)==-1,1:3);

borde2(:,1)=find(Nodos(:,1)==1)';           %borde2 de la forma [#elemen-
to,nodo1,nodo2,nodo3,nodo4;...] de nodos plano x-y, x=1
borde2(:,2:4)=Nodos(Nodos(:,1)==-1,1:3);

borde3(:,1)=find(Nodos(:,3)==-1)';           %borde3 de la forma [#elemen-
to,nodo1,nodo2,nodo3,nodo4;...] de nodos plano y-z, z=-1
borde3(:,2:4)=Nodos(Nodos(:,1)==-1,1:3);

borde4(:,1)=find(Nodos(:,3)==1)';           %borde4 de la forma [#elemen-
to,nodo1,nodo2,nodo3,nodo4;...] de nodos plano y-z, z=1
borde4(:,2:4)=Nodos(Nodos(:,1)==-1,1:3);

for cont99=1:n

    F(cont99*3-2,1)=0;
    F(cont99*3,1)=0;

    K(:,cont99*3-2)=zeros(length(K),1);
    K(cont99*3-2,:)=zeros(1,length(K));
    K(cont99*3-2,cont99*3-2)=1;

    K(:,cont99*3)=zeros(length(K),1);
    K(cont99*3,:)=zeros(1,length(K));
    K(cont99*3,cont99*3)=1;

end
```

```
for cont0 = 1: length(borde0(:,1))

    % No modificamos el vector fuerza debido a que el desplazamiento
    % impuesto es nulo ( F= F-Kij*u_impuesto = F ), u_impuesto=0

    F(borde0(cont0,1)*3-1,1) = 0;           %u impuesto = 0 en direccion x-y-z
para nodos base
    F(borde0(cont0,1)*3-2,1)=0;
    F(borde0(cont0,1)*3,1)=0;

    K(:,borde0(cont0,1)*3-1)=zeros(length(K),1);
    K(borde0(cont0,1)*3-1,:)=zeros(1,length(K));
    K(borde0(cont0,1)*3-1,borde0(cont0,1)*3-1)=1;

    K(:,borde0(cont0,1)*3-2)=zeros(length(K),1);
    K(borde0(cont0,1)*3-2,:)=zeros(1,length(K));
    K(borde0(cont0,1)*3-2,borde0(cont0,1)*3-2)=1;

    K(:,borde0(cont0,1)*3)=zeros(length(K),1);
    K(borde0(cont0,1)*3,:)=zeros(1,length(K));
    K(borde0(cont0,1)*3,borde0(cont0,1)*3)=1;

end

for cont1=1: length(borde1(:,1))

    F(borde1(cont1,1)*3,1)=0; %u impuesto = 0 en direccion z y x para nodos
plano x-y, x=-1 (z normal a plano)
    F(borde1(cont1,1)*3-2,1)=0;

    K(:,borde1(cont1,1)*3)=zeros(length(K),1);
    K(borde1(cont1,1)*3,:)=zeros(1,length(K));
    K(borde1(cont1,1)*3,borde1(cont1,1)*3)=1;

    K(:,borde1(cont1,1)*3-2)=zeros(length(K),1);
    K(borde1(cont1,1)*3-2,:)=zeros(1,length(K));
    K(borde1(cont1,1)*3-2,borde1(cont1,1)*3-2)=1;


end

for cont2=1: length(borde2(:,1))

    F(borde2(cont2,1)*3,1)=0;       %u impuesto = 0 en direccion z para nodos
plano x-y, x=1 (z normal a plano)
    F(borde2(cont2,1)*3-2,1)=0;

    K(:,borde2(cont2,1)*3)=zeros(length(K),1);
    K(borde2(cont2,1)*3,:)=zeros(1,length(K));
    K(borde2(cont2,1)*3,borde2(cont2,1)*3)=1;

    K(:,borde2(cont2,1)*3-2)=zeros(length(K),1);
    K(borde2(cont2,1)*3-2,:)=zeros(1,length(K));
    K(borde2(cont2,1)*3-2,borde2(cont2,1)*3-2)=1;
```

```
end

for cont3=1: length(borde3(:,1))

    F(borde3(cont3,1)*3-2,1)=0; % u impuesto = 0 en direccion x para nodos
plano z-y, z=-1 (x normal a plano)
    F(borde3(cont3,1)*3,1)=0;

    K(:,borde3(cont3,1)*3-2)=zeros(length(K),1);
    K(borde3(cont3,1)*3-2,:)=zeros(1,length(K));
    K(borde3(cont3,1)*3-2,borde3(cont3,1)*3-2)=1;

    K(:,borde3(cont3,1)*3)=zeros(length(K),1);
    K(borde3(cont3,1)*3,:)=zeros(1,length(K));
    K(borde3(cont3,1)*3,borde3(cont3,1)*3)=1;

end

for cont4=1: length(borde4(:,1))

    F(borde4(cont4,1)*3-2,1)=0; %u impuesto=0 en direccion x para nodos plano
z-y, z=1 (x normal a plano)
    F(borde4(cont4,1)*3,1)=0;

    K(:,borde4(cont4,1)*3-2)=zeros(length(K),1);
    K(borde4(cont4,1)*3-2,:)=zeros(1,length(K));
    K(borde4(cont4,1)*3-2,borde4(cont4,1)*3-2)=1;

    K(:,borde4(cont4,1)*3)=zeros(length(K),1);
    K(borde4(cont4,1)*3,:)=zeros(1,length(K));
    K(borde4(cont4,1)*3,borde4(cont4,1)*3)=1;

end

%% SOLUCION

uh = K\F;

%% COMPARACIOON SOLUCION ANALITICA

Uf=zeros(length(Nodos),1);

% Normas

N1a=0;
N1b=0;
N2a=0;
N2b=0;
dim=3;
ilambda=[0;0;0];
for cont1=1:Conect(end,1)

    Nel=[Conect(cont1,2),Conect(cont1,3),Conect(cont1,4),Conect(cont1,5)];
%Nodos del elemento al que recorro de la forma [nodo1,nodo2,nodo3,nodo4]
```

```
Elem=[Nodos(Nel(1),1:3);Nodos(Nel(2),1:3);Nodos(Nel(3),1:3);Nodos(Nel(4),1:3)]
; %Coordenadas de los nodos del elemento al que recorro de la forma
[x1,y1,z1;...;x4,y4,z4]
    [pe,wg]=gausspointsT3T4(order, Elem);
    lpg = length(pe(:,1));

    for cont2=1:lpg
    x=pe(cont2,:);
    [phi,phider,ptos]=maxent(h_node,dim,n,ncoord,x,ilambda);
    uhpg = zeros(3*length(ptos),1);

    B=zeros(6,3*length(phider(:,1)));
    N=zeros(3,3*length(phi));

        for cont3=1:length(phider(:,1))
            B(:,3*cont3-
2:3*cont3)=[phider(cont3,1),0,0;0,phider(cont3,2),0;0,0,phider(cont3,3);phider
(con-
t3,2),phider(cont3,1),0;0,phider(cont3,3),phider(cont3,2);phider(cont3,3),0,ph
ider(cont3,1)];
            N(:,3*cont3-
2:3*cont3)=[phi(cont3),0,0;0,phi(cont3),0;0,0,phi(cont3)];
            uhpg(3*cont3-2:3*cont3)=uh(3*cont3-2:3*cont3);
        end

        Uhpg = N*uhpg;

     ux=0;
     uy=((1+v)*(1-2*v)/(E*(1-v)))*(-force*(pe(cont2,2)-2));
     uz=0;

     Upg=[ux;uy;uz];

     eh=B*uhpg;
     Sh=C*eh;

     exx=0;
     eyy=force*((1+v)*(1-2*v)/(E*(1-v)));
     ezz=0;
     exy=0;
     eyz=0;
     ezx=0;


     sxx=force*(v/(1-v));
     syy=force;
     szz=force*(v/(1-v));
     sxy=0;
     syz=0;
     szx=0;


     Sf=[sxx;syy;szz;sxy;syz;szx];
     ef=[exx;eyy;ezz;exy;eyz;ezx];
```

```
      N1a = N1a + dot(Upg - Uhpg,Upg - Uhpg).*wg(cont2);
      N1b = N1b + dot(Upg,Upg).*wg(cont2);

      N2a = N2a + dot(ef-eh,Sf-Sh).*wg(cont2);
      N2b = N2b + dot(ef,Sf).*wg(cont2);

    end

end


Norma1 = N1a.^(0.5)./N1b.^(0.5)
Norma2 = N2a.^(0.5)./N2b.^(0.5)
```