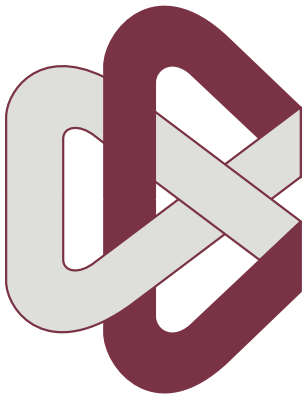


CIMAT

Centro de Investigación en Matemáticas A.C.

Maestría en Ciencias con Especialidad en
Computación y Matemáticas Industriales

“Cálculo de Estructuras Utilizando Elemento Finito con Cómputo en Paralelo”

por

José Miguel Vargas Félix

Asesor: Dr. Salvador Botello Rionda

Guanajuato, abril de 2010

Resumen

Nuestro trabajo trata sobre la solución numérica de problemas de deformación lineal de sólidos por medio del método de elementos finitos, estos problemas se resuelven utilizando estratégias de cómputo en paralelo. Hablamos sobre algunas formas de paralelizar los algoritmos, tanto utilizando modelos de memoria compartida como de memoria distribuída.

En particular nos centraremos en la descomposición de dominios usando el método alternante de Schwarz para resolver problemas de elemento finito con mallas muy refinadas. El método alternante de Schwarz se refiere a particionar el dominio del problema de tal forma que haya traslape en las fronteras comunes de las particiones. Cada partición se resuelve como un problema independiente, después se intercambian valores en las fronteras con las particiones adyacentes. Este proceso se repite de forma iterativa hasta la convergencia global del problema. Hablaremos de la paralelización utilizando dos tipos de algoritmos para resolver los sistemas de ecuaciones resultantes: gradiente conjugado y factorización Cholesky.

En cuanto a la factorización Cholesky, explicaremos varias estratégias para hacerla más eficiente, como son: almacenamiento utilizando esquemas de matrices ralas, factorización Cholesky simbólica para determinar la estructura de la matriz antes de calcular sus entradas y el reordenamiento de la matiz de rigidez para obtener una factorización más económica.

Se describe en esta tesis la implementación de un programa de cómputo que utiiliza la descomposición de Schwarz para resolver problemas de deformación de sólidos en dos y tres dimensiones. Este programa fue implementado para funcionar en un cluster de computo con el objetivo de resolver problemas de gran escala en forma eficiente. Finalmente mostraremos algunas gráficas de tiempos obtenidas al resolver sistemas de ecuaciones con decenas de millones de variables.

Palabras clave: *Ecuaciones diferenciales parciales, descomposición de dominios, álgebra lineal, cómputo en paralelo, método de elemento finito, método alternante de Schwarz.*

Agradecimientos

En primera instancia y de forma general quiero agradecer al CIMAT, es un gran lugar para crecer, tanto en forma académica, como en forma personal. Es una comunidad fantástica de la cual recibí incontable apoyo. En partícular ahora quiero decir gracias al Dr. Salvador Botello por compartir su conocimiento y experiencia, por todo el apoyo que me brindó. Además de asesorar y guiar con por buen camino esta tésis, consiguió los recursos (becas, cursos y hardware) necesarios para llevarla a cabo. A los revisores Dr. Arturo Hernández Aguirre y Dr. Miguel Ángel Móreles Vázquez, por sus valiosos comentarios y opiniones. Agradezco también a Jose Jesus Rocha Quezada quien fue el encargado poner a punto el cluster de cómputo en el cual se implementó esta tesis.

Al Dr. Juan José Tapia Armenta por su interés en este trabajo de tésis y por tomarse el tiempo de leer y enviarme comentarios, los cuales fueron de gran ayuda.

Apoyos recibidos

Durante el tiempo en que realicé mis estudios de maestía recibí apoyos en forma de becas, sin los cuales no hubiese sido posible la conclusión de estos y el desarrollo de esta tésis.

Centro de Investigación en Matemáticas (CIMAT)

- Octubre a diciembre de 2007. Art20 - III - Beca de estudios de Maestría (Acta 2007-013)
- Enero de 2008. Art20 - III - Beca de estudios de Maestría (Acta 2008-001)
- Agosto a diciembre de 2009. Art19- VI- Beca de elaboración de tesis de Maestría (Acta 2009-011)

Consejo Nacional de Ciencia y Tecnología (Conacyt)

- Enero de 2008 a julio de 2009. “Convocatoria de Becas Conacyt Nacionales enero - julio 2008”.
- Noviembre a diciembre de 2009. “Proyecto de Investigación 83974-Y”

A los implicados

Índice General

1. Introducción

1.1. Motivación

Algunos problemas modelados con elemento finito requieren de una discretización muy fina para reducir el grado de error, lo cual implica utilzar una gran cantidad de recursos de cómputo para poder ser resueltos. En particular nos enfocamos en problemas de elemento finito resultantes de la modelación de la deformación elástica de sólidos en dos y tres dimensiones. Sucede entonces que el tamaño de las matrices resultantes de esta modelación fina son tan grandes que no es factible almacenarlas en la memoria de una sola computadora, además, el tiempo que se tarda en encontrar la solución del sistema de ecuaciones puede ser demasiado extenso como para resultar práctico. Lo que nos lleva a cambiar de paradigma de programación y enfocarnos en el cómputo en paralelo, con el cual hacemos que múltiples computadoras trabajen de forma cooperativa en la resolución de este tipo de problemas.

1.2. Distribución de los capítulos

Iniciaremos más adelante en este capítulo hablando de forma breve del cómputo en paralelo, tanto de hardware y software, la motivación matemática de éste y haremos una intruducción a la técnica de descomposición de dominios hablando brevemente del estado del arte de éste.

El orden de los capítulos siguientes tiene que ver con la forma en que se fue atacando el problema. Con el capítulo 2 describiremos la teoría de deformación elástica de sólidos con elementos finitos.

En el capítulo 3 describiremos la paralelización de dicho programa utilizando el esquema de memoria compartida utilizando como *solver* el método de gradiente conjugado en paralelo, presentamos algunos resultados.

A continuación, en el capítulo 4, hablaremos de la teoría e implementación un esquema híbrido, que combina los esquemas de memoria distribuida y memoria compartida. En pocas palabras, lo que hacemos es particionar el dominio de un problema de elementos finitos para así formar problemas independientes que están relacionados entre sí a través de condiciones de frontera comunes. Cada problema independiente se resuelve utilizando el solver descrito en el capítulo 3. Las soluciones locales a cada partición se combinan con las de las particiones adyacentes de forma iterativa hasta lograr una convergencia. Mostramos algunos resultados y denotamos algunos de los problemas de este esquema híbrido.

Para solventar los problemas encontrados con el esquema híbrido, decidimos implementar como *solver* del sistema de ecuaciones la factorización Cholesky simbólica para matrices ralas. En el capítulo 5 mostramos la teoría y detalles de implementación de este solver.

Al utilizar la factorización Cholesky para matrices ralas como solver para el esquema de memoria distribuida implementado se obtuvieron resultados interesantes, los cuales son mostrados en el capítulo 6.

Se incluye al final de este trabajo un apéndice con los detalles del programa de software que implementa los algoritmos descritos en los capítulos 3 a 5.

1.3. Modelos de cómputo en paralelo

El modelo más común para clasificar las arquitecturas de cómputo en paralelo es la taxonomía de Flynn [Flyn72]. Esta taxonomía agrupa las diferentes arquitecturas de computadoras basándose en el flujo de datos e instrucciones en una computadora, denotando si se puede o no procesar datos y/o ejecutar instrucciones en paralelo, lo que da como resultado las cuatro categorías de la tabla 1.1.

	Una instrucción	**Múltiples instrucciones**
Un dato	SISD	MISD
Múltiples datos	SIMD	MIMD

Tabla 1.1. Taxonomía de Flynn.

SISD (*single instruction, single data*). En este modelo no existe paralelismo ni de procesamiento de datos ni de ejecución de instrucciones. Se refiere entonces al procesamiento serial encontrado en las computadoras con un solo procesador.

SIMD (*single instruction, multiple data*). Este tipo de computadoras tienen arreglos de procesadores que ejecutan la misma secuencia de instrucciones en datos diferentes. Las tarjetas de video modernas o GPUs se basan en este modelo, cada una puede tener decenas o cientos de procesadores que pueden, por ejemplo, aplicar la misma operación a muchas entradas de una matriz al mismo tiempo. El diseño de este tipo de hardware lo hace especialmente eficiente para operaciones sencillas, pero no lo es tanto cuando los algoritmos presentan condiciones de la forma: si el dato es A ejecuta esta instrucción, si es B esta otra. Este tipo de condicionantes no son fáciles de paralelizar, lo que resulta en que en estos casos se ejecute primero la instrucción para A y luego la instrucción para B de forma serial.

MISD (*multiple instruction, single data*). Es el caso menos común de esta taxonomía, en él múltiples procesadores trabajan en paralelo pero en un mismo dato. Este tipo de arquitectura se utiliza en sistemas que tienen que operar a prueba de fallos, en los cuales es necesario tener redundancia. Por ejemplo la computadora de control de vuelo del transbordador espacial utiliza esta arquitectura.

MIMD (*multiple instruction, multiple data*). En este caso se tienen muchos procesadores, cada uno ejecutando una secuencia de instrucciones aplicada a datos diferentes. Por ejemplo, las computadoras de escritorio *multi-core* siguen esta arquitectura. Los sistemas distribuidos también se consideran dentro de esta clase, tanto si utilizan esquemas de memoria compartida o distribuida. Es la arquitectura más

utilizada en computación científica de alto rendimiento, la mayoría de las computadoras en la lista TOP500, utilizan este esquema.

1.4. Avance del cómputo en paralelo

En años recientes se ha logrado un avance exponencial en cuanto a la capacidad de procesamiento, de los procesadores, como se muestra en la figura 1.1.

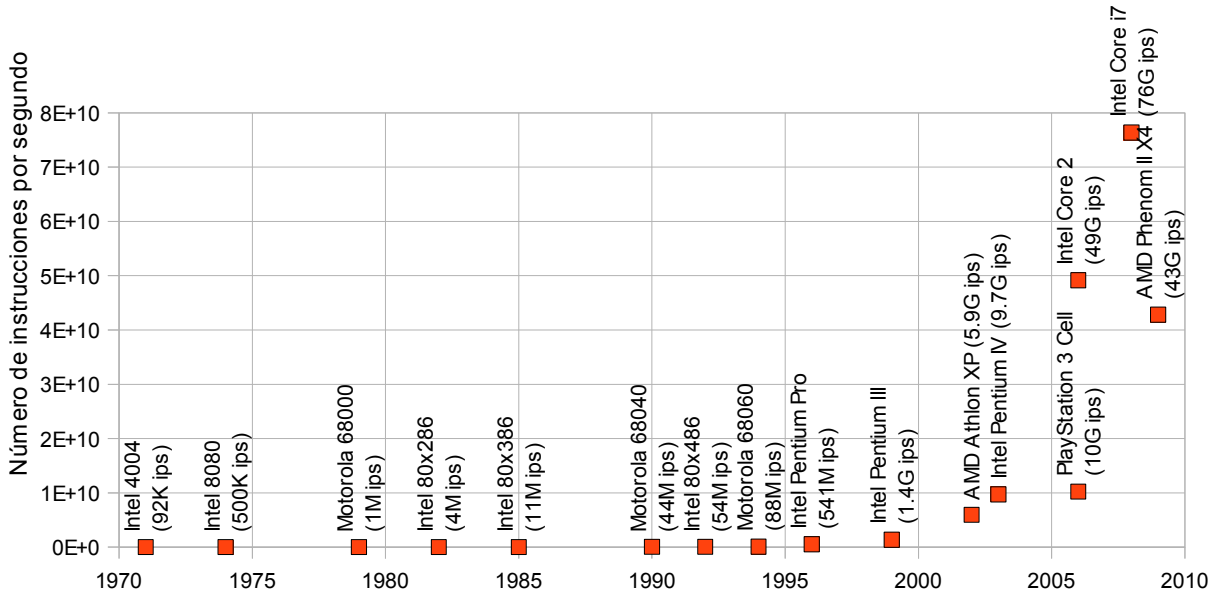


Figura 1.1. Evolución de la capacidad de los procesadores en instrucciones por segundo (ips).

Este incremento no solo se debe al desarrollo de la tecnología con la que se fabrican los procesadores, sino al hecho de que ahora se fabrican procesadores con varios núcleos. Por ejemplo el procesador Intel Core i7-980X tiene 6 núcleos. Los sistemas operativos y compiladores modernos aprovechan esta característica para ejecutar sus rutinas en paralelo. El estándar más utilizado para procesadores multi-core es OpenMP [Chap08].

Otro avance importante en los últimos años es que el uso de *clusters* de computadoras se ha hecho accesible. En particular los *clusters* Beowulf [Ster95], son implementados con computadoras comerciales que son interconectadas por medio de una red local (figura 1.2), las computadoras funcionan ejecutando sistemas operativos libres (GNU/Linux, FreeBSD, etc.) y se programan utilizando librerías que permiten la comunicación entre los programas de cálculo numérico. La librería más utilizada para desarrollar aplicaciones en paralelo en *clusters* Beowulf es la interface de pase de mensajes MPI (del inglés, *Message Passing Interface)* [MPIF08], ésta permite implementar con facilidad programas que se ejecuten bajo el esquema de memoria distribuida.

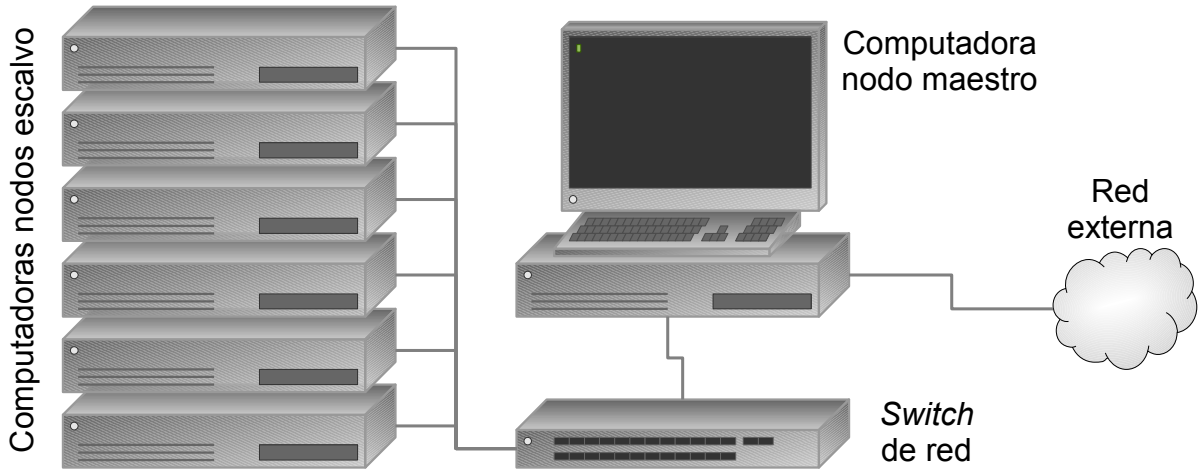


Figura 1.2. Configuración típica de un cluster *Beowulf.*

El trabajo de esta tesis implicó el desarrollo de un programa de software para resolver problemas de elemento finito utilizando cómputo en paralelo. Para la implementación y prueba de este programa se utilizó el *cluster* de cómputo del CIMAT, el cual sigue el modelo MIMD. Cuenta con 30 computadoras multi-core, las cuales proveen un total de 192 procesadores, los cuales son accesados a través de un nodo maestro. Éstas tienen instalado el sistema operativo GNU/Linux, la *suite* de compiladores GNU/GCC y la librería de MPI OpenMPI..

1.5. Las matemáticas del paralelismo

Hay muchas operaciones matemáticas básicas que pueden paralelizarse, esto significa que pueden separarse en varias sub-operaciones que puede realizarse de forma independiente. Por ejemplo, el la suma de dos vectores $\boldsymbol{x}$, $\boldsymbol{y}$ para producir otro vector $\boldsymbol{c}$,

$$c_i = x_i + y_i,\ i = 1, \ldots, N.$$

En este caso las N sumas pueden realizarse simultáneamente, asignando una a cada procesador. Lo que hay que resaltar es que no hay dependencia entre los diferentes pares de datos, tenemos entonces el paralelismo más eficiente.

Hay operaciones que presentan mayor dificultad para paralelizarse, por ejemplo el producto punto $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$

$$a = \sum_{i=1}^{N} x_i\, y_i,$$

donde a es un escalar, una primera aproximación sería verlo como una secuencia de sumas de productos que requieren ir acumulando un resultado, al verlo así no es una operación paralelizable. Sin embargo, podemos reorganizar el proceso como se muestra en la figura 1.3.

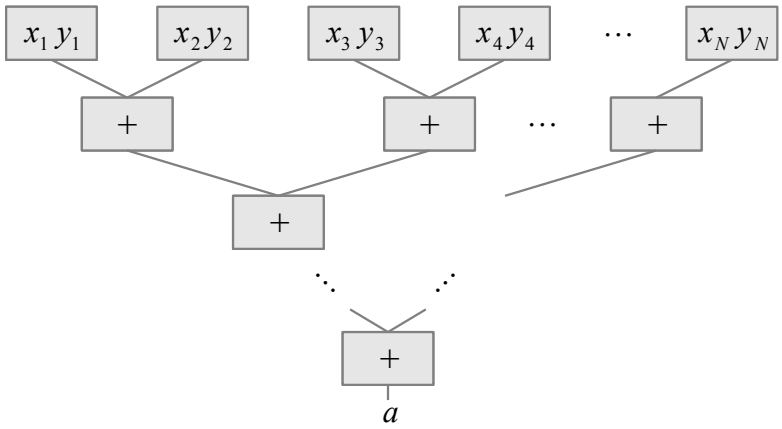


Figura 1.3. Paralelización del producto punto.

En este caso se tiene una paralelización eficiente de la multiplicación de las entradas de los vectores, después se va reduciendo la eficiencia, el primer grupo se sumas es realizado por $N/2$ procesadores, las segundas sumas por $N/4$ procesadores, hasta llegar a la suma final en un procesador, el que obtiene el resultado final a. En este tipo de paralelización existe dependencia entre los datos y ello provoca que sea menos eficiente. Muchos algoritmos seriales requieren, para ser paralelizados, de re-ordenar las operaciones con una estrategia de "divide y vencerás", como en el caso del producto punto. Usualmente se tendrán memos procesadores que el tamaño del vector, por lo que se asignan varias operaciones de un grupo a cada procesador, las cuales se ejecutarán en secuencia, lo que reduce aún más la eficiencia del paralelismo.

1.6. Sobre el estado del arte

La teoría de descomposición de dominios para resolver problemas de ecuaciones diferenciales parciales es un tema en constante desarrollo. En años recientes ha sido beneficiado por la creación de sistemas de cómputo cada vez más veloces, con mayor capacidad de memoria y velocidad de comunicación entre nodos. La tendencia iniciada en la década de los años noventa ha sido trabajar al mismo tiempo con las soluciones individuales de cada particion y de forma general con un subdominio grueso de todo el problema.

En el caso del trabajo con problemas de elemento finito en mecánica de sólidos en régimen lineal se presentan matrices simétricas positivas definidas, varios de los métodos más modernos para resolver este tipo de problemas usan descomposición de dominios sin traslape, por ello vamos a hablar de éstos en las siguientes secciones, aunque muy brevemente. Para esta tesis, sin embargo, elegimos trabajar usando el método de descomposición de dominios con traslape, el cual será descrito en el capítulo 4.

1.6.1. Discretización con el método de elementos finitos

Una técnica muy utilizada para resolver numéricamente problemas de ecuaciones difrenciales parciales es el método de elementos finitos. El método consiste en discretizar un dominio dividiéndolo en elementos geométricos que aproximadamente cubran al dominio. Se genera así una malla de elementos, los cuales

comparten aristas y nodos, figura 1.4. Las relaciones entre nodos corresponden a entradas en una matriz. Así, la relación entre los nodos i y j equivale a un valor en la entrada a_{ij} en una matriz $\boldsymbol{A}$. Dado que existe una relación del nodo i al nodo j, también existe una relación (no necesariamente con el mismo valor) del nodo j al nodo i, lo que producirá una matriz con estructura simétrica, aunque no necesariamente simétrica en cuanto a sus valores (en las secciones siguientes se tratan sólo problemas con matrices simétricas tanto en estructura como en valores). Además, en la diagonal aparecen entradas que representan a los nodos.

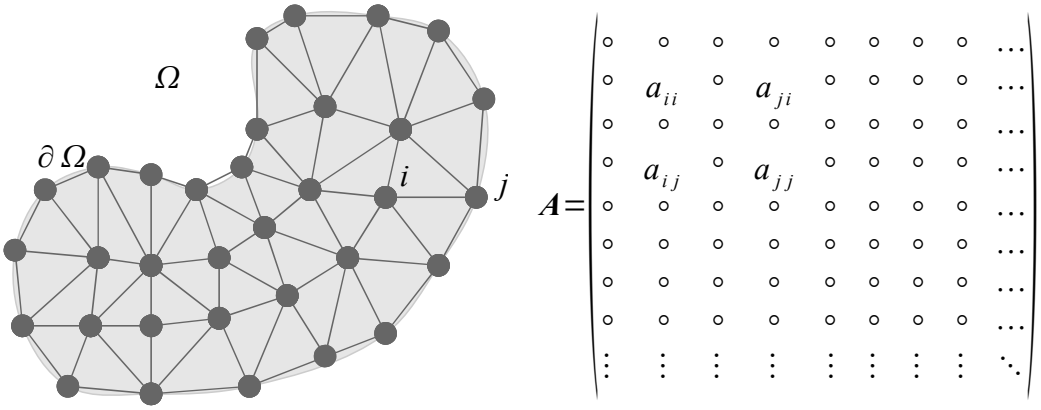


Figura 1.4. Discretización de un dominio Ω.

Tendremos entonces la representación del problema de ecuaciones diferenciales parciales como un sistema de ecuaciones

$$\boldsymbol{A}\,\boldsymbol{x}=\boldsymbol{b},$$

$$\boldsymbol{x}=\boldsymbol{c} \text{ en } \partial\,\Omega$$

con ciertas condiciones (Dirichlet o Neumann) en la frontera $\partial\,\Omega$. En el capítulo 2 formularemos problema de deformación lineal de sólidos utilizando este método.

1.6.2. Descomposición de dominios

La descomposición de dominios nace de la necesidad de trabajar ecuaciones diferenciales en dominios discretizados que producen sistemas de ecuaciones grandes, tratando de resolverlos de forma eficiente. En forma general podemos decir que existen dos tipos de descomposición de dominios, utilizando particiones sin traslape o con traslape, figura 1.5.

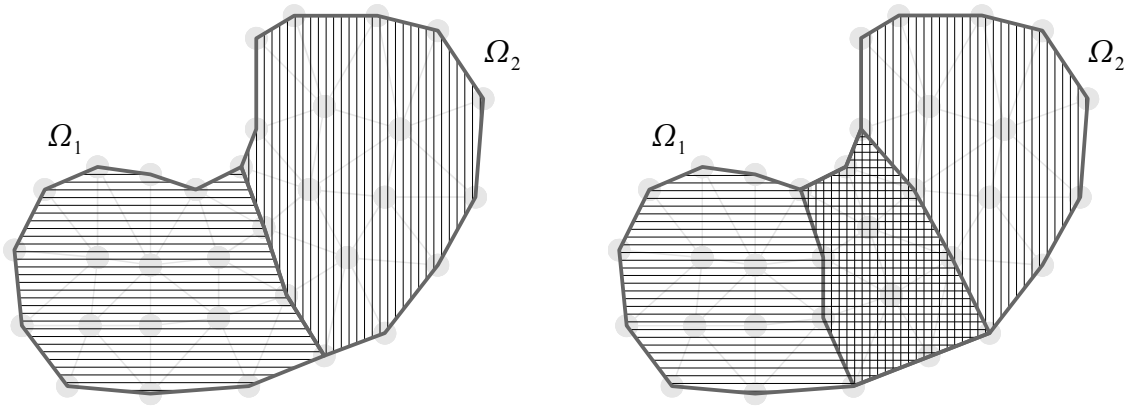


Figura 1.5. Descomposición en dominios sin traslape (izquierda) y con traslape (derecha).

Dividir un dominio en P particiones equivale entonces a separar en P bloques la matriz $\boldsymbol{A}$ que representa las relaciones entre nodos. Cada dominio entonces estará representado por una matriz $\boldsymbol{A}^{(p)}$, con $p=1\ldots P$. El siguiente paso es resolver el sistema de ecuaciones con la matriz $\boldsymbol{A}^{(p)}$ de cada partición de foma independiente, utilizando algún método convencional para resolver sistemas de ecuaciones. La soluciones obtenidas se intercambiarán con las particiones adyacentes y así, de forma iterativa, se irá aproximando la solución global del sistema.

1.6.3. Dominios sin traslape, complemento de Schur

En dominios sin traslape,

$$\overline{\Omega}=\overline{\Omega_1\cup\Omega_2},\ \Omega_1\cap\Omega_2=\emptyset,\ \Gamma=\partial\Omega_1\cap\partial\Omega_2,$$

dos particiones adyacentes tendrán nodos en común en la frontera Γ, lo que equivaldrá a entradas iguales en sus respectivas matrices. Podemos entonces formar la matriz $\boldsymbol{A}$ como

$$\boldsymbol{A}=\begin{pmatrix}\boldsymbol{A}_{II}^{(1)} & \boldsymbol{0} & \boldsymbol{A}_{I\Gamma}^{(1)}\\ \boldsymbol{0} & \boldsymbol{A}_{II}^{(2)} & \boldsymbol{A}_{I\Gamma}^{(2)}\\ \boldsymbol{A}_{\Gamma I}^{(1)} & \boldsymbol{A}_{\Gamma I}^{(2)} & \boldsymbol{A}_{\Gamma\Gamma}\end{pmatrix},\ \boldsymbol{x}=\begin{pmatrix}\boldsymbol{x}_I^{(1)}\\ \boldsymbol{x}_I^{(2)}\\ \boldsymbol{x}_\Gamma\end{pmatrix},\ \boldsymbol{b}=\begin{pmatrix}\boldsymbol{b}_I^{(1)}\\ \boldsymbol{b}_I^{(2)}\\ \boldsymbol{b}_\Gamma\end{pmatrix}. \tag{1.1}$$

Para cada patición p tendremos entonces

$$\boldsymbol{A}^{(p)}=\begin{pmatrix}\boldsymbol{A}_{II}^{(p)} & \boldsymbol{A}_{I\Gamma}^{(p)}\\ \boldsymbol{A}_{\Gamma I}^{(p)} & \boldsymbol{A}_{\Gamma\Gamma}^{(p)}\end{pmatrix},\ \boldsymbol{x}^{(p)}=\begin{pmatrix}\boldsymbol{x}_I^{(p)}\\ \boldsymbol{x}_\Gamma^{(p)}\end{pmatrix},\ \boldsymbol{b}^{(p)}=\begin{pmatrix}\boldsymbol{b}_I^{(p)}\\ \boldsymbol{b}_\Gamma^{(p)}\end{pmatrix},\ \text{con } p=1\ldots P. \tag{1.2}$$

Una estrategia para resolver este tipo de problemas es utilizar el sistema de complementos de Schur [Tose05 pp1-7]. Parte de eliminar las incógnitas $\boldsymbol{x}_I^{(p)}$ en el interior de la partición, esto corresponde a la factorización de la matriz de (1.1) como

$$\boldsymbol{A}=\boldsymbol{L}\,\boldsymbol{R}=\begin{pmatrix}\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0}\\ \boldsymbol{A}_{\Gamma I}^{(1)}\boldsymbol{A}_{II}^{(1)-1} & \boldsymbol{A}_{\Gamma I}^{(2)}\boldsymbol{A}_{II}^{(2)-1} & \boldsymbol{I}\end{pmatrix}\begin{pmatrix}\boldsymbol{A}_{II}^{(1)} & \boldsymbol{0} & \boldsymbol{A}_{I\Gamma}^{(1)}\\ \boldsymbol{0} & \boldsymbol{A}_{II}^{(2)} & \boldsymbol{A}_{I\Gamma}^{(2)}\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{S}\end{pmatrix},$$

y un sistema lineal resultante

$$\begin{pmatrix}\boldsymbol{A}_{II}^{(1)} & \boldsymbol{0} & \boldsymbol{A}_{I\Gamma}^{(1)}\\ \boldsymbol{0} & \boldsymbol{A}_{II}^{(2)} & \boldsymbol{A}_{I\Gamma}^{(2)}\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{S}\end{pmatrix}\boldsymbol{x}=\begin{pmatrix}\boldsymbol{b}_I^{(1)}\\ \boldsymbol{b}_I^{(2)}\\ \boldsymbol{g}_\Gamma\end{pmatrix},$$

aquí

$$\boldsymbol{S}=\boldsymbol{A}_{\Gamma\Gamma}-\boldsymbol{A}_{\Gamma I}^{(1)}\left(\boldsymbol{A}_{II}^{(1)}\right)^{-1}\boldsymbol{A}_{I\Gamma}^{(1)}-\boldsymbol{A}_{\Gamma I}^{(2)}\left(\boldsymbol{A}_{II}^{(2)}\right)^{-1}\boldsymbol{A}_{I\Gamma}^{(2)},$$

es el complemento de Schur relativo a las incógnitas en Γ. Acomodando como en (1.2), podemos escribir el complemento de Schur local

$$\boldsymbol{S}^{(p)}=\boldsymbol{A}_{\Gamma\Gamma}^{(p)}-\boldsymbol{A}_{\Gamma I}^{(p)}\left(\boldsymbol{A}_{II}^{(p)}\right)^{-1}\boldsymbol{A}_{I\Gamma}^{(p)},\ \text{con } p=1\ldots P,$$

encontraremos el complemento de Schur para $\boldsymbol{x}_\Gamma$ con

$$\boldsymbol{S}\,\boldsymbol{x}_\Gamma=\boldsymbol{g}_\Gamma, \tag{1.3}$$

con

$$\boldsymbol{S}=\boldsymbol{S}^{(1)}+\boldsymbol{S}^{(2)},$$

$$\boldsymbol{g}_\Gamma=\left(\boldsymbol{b}_\Gamma^{(1)}-\boldsymbol{A}_{\Gamma I}^{(1)}\left(\boldsymbol{A}_{II}^{(1)}\right)^{-1}\boldsymbol{b}_I^{(1)}\right)+\left(\boldsymbol{b}_\Gamma^{(2)}-\boldsymbol{A}_{\Gamma I}^{(2)}\left(\boldsymbol{A}_{II}^{(2)}\right)^{-1}\boldsymbol{b}_I^{(2)}\right)=\boldsymbol{g}_\Gamma^{(1)}+\boldsymbol{g}_\Gamma^{(1)}.$$

El método de complementos de Shur consiste entonces en encontrar primero la solución en la frontera común $\boldsymbol{x}_\Gamma$ resolviendo (1.3) y después aplicar ésta como condición de Dirichlet para encontrar las incógnitas dentro de los dominios con

$$\boldsymbol{x}_I^{(p)}=\left(\boldsymbol{A}_{II}^{(p)}\right)^{-1}\left(\boldsymbol{b}_I^{(p)}-\boldsymbol{A}_{I\Gamma}^{(p)}\boldsymbol{x}_\Gamma\right).$$

Si partimos de una condición de Neumann en la frontera común Γ,

$$\boldsymbol{\lambda}_\Gamma=\boldsymbol{\lambda}_\Gamma^{(1)}=-\boldsymbol{\lambda}_\Gamma^{(2)}, \tag{1.4}$$

resolvemos los sistemas locales de Neumann para encontrar $\boldsymbol{x}^{(1)}$ y $\boldsymbol{x}^{(2)}$,

$$\begin{pmatrix}\boldsymbol{A}_{II}^{(p)} & \boldsymbol{A}_{I\Gamma}^{(p)}\\ \boldsymbol{A}_{\Gamma I}^{(p)} & \boldsymbol{A}_{\Gamma\Gamma}^{(p)}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}_I^{(p)}\\ \boldsymbol{x}_\Gamma^{(p)}\end{pmatrix}=\begin{pmatrix}\boldsymbol{b}_I^{(p)}\\ \boldsymbol{b}_\Gamma^{(p)}+\boldsymbol{\lambda}_\Gamma^{(p)}\end{pmatrix}, \text{ con } p=1\ldots P.$$

Usando la factorización en bloques de las matrices locales, encontramos

$$\boldsymbol{x}_\Gamma^{(p)}=\left(\boldsymbol{S}^{(p)}\right)^{-1}\left(\boldsymbol{g}_\Gamma^{(p)}-\boldsymbol{\lambda}_\Gamma^{(p)}\right),$$

ahora podemos encontrar $\boldsymbol{\lambda}_\Gamma$ usando

$$\boldsymbol{F}\,\boldsymbol{\lambda}_\Gamma=\boldsymbol{d}_\Gamma, \tag{1.5}$$

con

$$\boldsymbol{F}=\left(\boldsymbol{S}^{(1)}\right)^{-1}+\left(\boldsymbol{S}^{(2)}\right)^{-1},$$

$$\boldsymbol{d}_\Gamma=\boldsymbol{d}_\Gamma^{(1)}+\boldsymbol{d}_\Gamma^{(2)}=-\left(\boldsymbol{S}^{(1)}\right)^{-1}\boldsymbol{g}_\Gamma^{(1)}+\left(\boldsymbol{S}^{(2)}\right)^{-1}\boldsymbol{g}_\Gamma^{(2)}.$$

1.6.4. Métodos Neumann-Neumann

Se les conoce como métodos Neumann-Neuman [Tose05 pp10-12] porque se resuelve un problema Neumann a cada lado de la interfaz entre los subdominios.

El primer paso para resolver el algoritmo Neumann-Neumann es resolver el problema con condiciones Dirichlet en cada domino Ω_p, con valores x_Γ^0 en Γ, después en cada subdominio resolver un problema Neumann, con valores escogidos como la diferencia en las derivadas normales de la solución de dos problemas Dirichlet. Los valores en Γ de las soluciones de estos problemas Neumann son utilizados para corregir la aproximación inicial x_Γ^0 y obtener x_Γ^1. Para dos partciones, podemos describir el método utilizando operadores diferenciales, así, para $n\geq 0$,

$$\text{Problema Dirichlet } D_p \quad \left\{ \begin{array}{ll} -\Delta x_p^{n+1/2} = b & \text{en } \Omega_p \\ x_p^{n+1/2} = 0 & \text{sobre } \partial\Omega_p \setminus \Gamma \\ x_p^{n+1/2} = x_\Gamma^n & \text{en } \Gamma \end{array} \right\}, \; p = 1, 2,$$

$$\text{Problema Neumann } N_p \quad \left\{ \begin{array}{lll} -\Delta \psi_p^{n+1} = 0 & & \text{en } \Omega_p \\ \psi_p^{n+1} = 0 & & \text{sobre } \partial\Omega_p \setminus \Gamma \\ \dfrac{\partial \psi_p}{\partial n_i}^{n+1} = \dfrac{\partial x_1}{\partial n_1}^{n+1/2} + \dfrac{\partial x_2}{\partial n_2}^{n+1/2} & & \text{en } \Gamma \end{array} \right\}, \; p = 1, 2.$$

Para la siguiente iteración usaremos

$$x_\Gamma^{n+1} = x_\Gamma^n - \theta\left(\psi_1^{n+1} + \psi_2^{n+1}\right) \text{ en } \Gamma,$$

con un $\theta \in (0, \theta_{\max})$ elegido de forma adecuada.

Para mostrar en este procedimiento forma matricial intoduciremos los vectores internos de grados e libertad $\boldsymbol{v}_p = \boldsymbol{x}_I^{(p)}$ y $\boldsymbol{w}_p = \boldsymbol{\psi}_I^{(p)}$, así

$$\text{Problema Dirichlet } D_p \quad \boldsymbol{A}_{II}^{(p)} \boldsymbol{v}_p^{n+1/2} + \boldsymbol{A}_{I\Gamma}^{(p)} \boldsymbol{x}_\Gamma^n = \boldsymbol{b}_I^{(p)}, \; p = 1, 2, \tag{1.6}$$

$$\text{Problema Neumann } N_p \quad \begin{pmatrix} \boldsymbol{A}_{II}^{(p)} & \boldsymbol{A}_{I\Gamma}^{(p)} \\ \boldsymbol{A}_{\Gamma I}^{(p)} & \boldsymbol{A}_{\Gamma\Gamma}^{(p)} \end{pmatrix} \begin{pmatrix} \boldsymbol{w}_p^{n+1/2} \\ \boldsymbol{\eta}_p^{n+1/2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{r}_\Gamma \end{pmatrix}, \; p = 1, 2, \tag{1.7}$$

$$\boldsymbol{x}_\Gamma^{n+1} = \boldsymbol{x}_\Gamma^n - \theta\left(\boldsymbol{\eta}_1^{n+1} + \boldsymbol{\eta}_2^{n+1}\right),$$

donde el residual $\boldsymbol{r}_\Gamma$ se define como

$$\boldsymbol{r}_\Gamma = \left(\boldsymbol{A}_{\Gamma I}^{(1)} \boldsymbol{v}_1^{n+1/2} + \boldsymbol{A}_{\Gamma\Gamma}^{(1)} \boldsymbol{x}_\Gamma^n - \boldsymbol{b}_\Gamma^{(1)}\right) + \left(\boldsymbol{A}_{\Gamma I}^{(2)} \boldsymbol{v}_2^{n+1/2} + \boldsymbol{A}_{\Gamma\Gamma}^{(2)} \boldsymbol{x}_\Gamma^n - \boldsymbol{b}_\Gamma^{(2)}\right).$$

A continuación eliminamos $\boldsymbol{v}_p^{n+1/2}$, $\boldsymbol{w}_p^{n+1/2}$ de (1.6) y (1.7). A partir del problema Dirichlet D_p,

$$\boldsymbol{r}_\Gamma = -\left(\boldsymbol{g}_\Gamma - \boldsymbol{S}\, \boldsymbol{x}_\Gamma^n\right),$$

que indica que la diferencia $\boldsymbol{r}_\Gamma$ es igual al negativo del residual del complemento de Schur para el sistema (1.5). Usamos la factorización por bloques de las matrices locales $\boldsymbol{A}^{(p)}$, los problemas de Neuman N_p dan

$$\boldsymbol{\eta}_p^{n+1} = \left(\boldsymbol{S}^{(p)} \boldsymbol{r}_\Gamma\right)^{-1} = -\left(\boldsymbol{S}^{(p)} \boldsymbol{r}_\Gamma\right)^{-1} \left(\boldsymbol{g}_\Gamma - \boldsymbol{S}\, \boldsymbol{x}_\Gamma^n\right).$$

Finalmente encontramos

$$\boldsymbol{x}_\Gamma^{n+1} - \boldsymbol{x}_\Gamma^n = \theta\left(\left(\boldsymbol{S}_1^{(1)}\right)^{-1} + \left(\boldsymbol{S}^{(2)}\right)^{-1}\right)\left(\boldsymbol{g}_\Gamma - \boldsymbol{S}\, \boldsymbol{x}_\Gamma^n\right).$$

El precondicionador para el sistema (1.5) sería

$$\boldsymbol{S}\, \boldsymbol{F} = \left[\left(\boldsymbol{S}^{(1)}\right)^{-1} + \left(\boldsymbol{S}^{(2)}\right)^{-1}\right] \boldsymbol{S} = \left[\left(\boldsymbol{S}^{(1)}\right)^{-1} + \left(\boldsymbol{S}^{(2)}\right)^{-1}\right]\left(\boldsymbol{S}^{(1)} + \boldsymbol{S}^{(2)}\right).$$

Tomando como base el algoritmo Neumann-Neumann, se ha desarrollado el método BDD (en inglés, B*alancing Domain Decomposition*), la formulación de éste se puede encontrar en [Mand93]. Los métodos BDD son algoritmos iterativos que trabajan con subestructuras, es decir, métodos donde los grados de libertad interiores de cada una de las particiones sin traslape son eliminados. En cada iteración se resuelven los problemas locales de cada partición y se combinan las soluciones con un problema en una malla más gruesa creata a partir del subdominio del espacio nulo. En las fornteras comunes se establecen condiciones Neumann.

Una de las estratégias más recientes para resolver problemas de descomposición de dominios para problemas de elemento finito de deformación elástica de sólidos que producen matrices simétricas positivas definidas es el método de balance de descomposición de dominios por restricciones, BDDC (en inglés, B*alancing Domain Decomposition by Constraints*) [Mand02] y [Dohr03]. El problema se resuelve utilizando formulando un espacio grueso (*coarse space*) el cual consiste de fuciones de minimización de energía con los grados de libertad dados en la malla gruesa, una descripción condensada puede encontrarse en [Widl08].

Recientemente se han desarrollado algoritmos basados en BDD formulados con subdominios con traslape, la descripción de estos puede consultarse en [Kimn06a] y [Kimn06b].

1.6.5. Métodos Dirichlet-Dirichlet

El caso dual al algoritmo Neumann-Neuman es el algoritmo con condiciones Dirichlet a ambos lados de la interfaz común de los subdominios [Tose05 pp12-14]. Si comenzamos con una aproximación inicial λ_Γ^0 en Γ, como en (1.4), podemos resolver inicialmente problemas con condiciones Neumann, en cada Ω_p. Después resolveremos un problema con las condiciones Dirichlet en Γ elegidas como la diferencia de la traza de las soluciones de los problemas Neumann en Γ. Los valores en Γ de las derivadas normales de las soluciones de estos problemas Dirichlet son empleados para corregir la λ_Γ^0 incial y encontrar el valor λ_Γ^1 para la nueva iteración. Para dos partciones, podemos describir el método utilizando operadores diferenciales, así, para $n \geq 0$,

$$\text{Problema Neumann } N_p \quad \left\{ \begin{array}{ll} -\Delta x_p^{n+1/2} = b & \text{en } \Omega_p \\ x_p^{n+1/2} = 0 & \text{sobre } \partial\Omega_p \setminus \Gamma \\ \dfrac{\partial x_p}{\partial n_p}^{n+1/2} = \lambda_p^n & \text{en } \Gamma \end{array} \right\}, \; p = 1, 2,$$

$$\text{Problema Dirichlet } D_p \quad \left\{ \begin{array}{ll} -\Delta \psi_p^{n+1} = 0 & \text{en } \Omega_p \\ \psi_p^{n+1} = 0 & \text{sobre } \partial\Omega_p \setminus \Gamma \\ \psi_p^{n+1} = x_1^{n+1/2} - x_2^{n+1/2} & \text{en } \Gamma \end{array} \right\}, \; p = 1, 2.$$

Para la siguiente iteración usaremos

$$\lambda_\Gamma^{n+1} = \lambda_\Gamma^n - \theta \left(\frac{\partial \psi_1}{\partial n_1}^{n+1} + \frac{\partial \psi_2}{\partial n_2}^{n+1} \right) \text{en } \Gamma,$$

con un $\theta \in (0, \theta_{\max})$ elegido de forma adecuada.

Para mostrar en este procedimiento forma matricial intoduciremos los vectores internos de grados e libertad $\boldsymbol{v}_p = \boldsymbol{x}_I^{(p)}$ y $\boldsymbol{w}_p = \boldsymbol{\psi}_I^{(p)}$, así

$$\text{Problema Neumann } N_p \quad \begin{pmatrix} \boldsymbol{A}_{II}^{(p)} & \boldsymbol{A}_{I\Gamma}^{(p)} \\ \boldsymbol{A}_{\Gamma I}^{(p)} & \boldsymbol{A}_{\Gamma\Gamma}^{(p)} \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_p^{n+1/2} \\ \boldsymbol{\gamma}_p^{n+1/2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_I^{(p)} \\ \boldsymbol{b}_\Gamma^{(p)} + \boldsymbol{\lambda}_p^n \end{pmatrix}, \; p = 1, 2, \tag{1.8}$$

$$\text{Problema Dirichlet } D_p \quad \boldsymbol{A}_{II}^{(p)} \boldsymbol{w}_p^{n+1} + \boldsymbol{A}_{I\Gamma}^{(p)} \boldsymbol{r}_\Gamma = \boldsymbol{0}, \; p = 1, 2, \tag{1.9}$$

$$\boldsymbol{\lambda}_\Gamma^{n+1} = \boldsymbol{\lambda}_\Gamma^n - \theta \left(\boldsymbol{\eta}_1^{n+1} + \boldsymbol{\eta}_2^{n+1} \right),$$

donde el residual $\boldsymbol{r}_\Gamma$ se define como

$$\boldsymbol{r}_{\Gamma}=\boldsymbol{\gamma}_1^{n+1/2}-\boldsymbol{\gamma}_2^{n+1/2},$$

y los flujos $\boldsymbol{\eta}_p^{n+1}$ por

$$\boldsymbol{\eta}_p^{n+1}=\boldsymbol{A}_{\Gamma I}^{(p)}\boldsymbol{w}_p^{n+1/2}+\boldsymbol{A}_{\Gamma\Gamma}^{(p)}\boldsymbol{r}_{\Gamma}.$$

Para eliminar $\boldsymbol{v}_p^{n+1/2}$, $\boldsymbol{w}_p^{n+1/2}$ y $\boldsymbol{\gamma}_p^{n+1/2}$ de (1.8) y (1.9) usamos la factorización por bloques de las matrices locales $\boldsymbol{A}^{(p)}$, los problemas de Neuman N_p dan

$$\boldsymbol{r}_{\Gamma}=-\left(\boldsymbol{d}_{\Gamma}-\boldsymbol{F}\,\boldsymbol{\lambda}_{\Gamma}^{n}\right),$$

que indica que la diferencia $\boldsymbol{r}_{\Gamma}$ es igual al negativo del residual del complemento de Schur para el sistema (1.5). De los problemas Dirichlet D_p podemos establecer

$$\boldsymbol{\eta}_p^{n+1}=\boldsymbol{S}^{(p)}\boldsymbol{r}_{\Gamma}=-\boldsymbol{S}^{(p)}\boldsymbol{r}_{\Gamma}\left(\boldsymbol{d}_{\Gamma}-\boldsymbol{F}\,\boldsymbol{\lambda}_{\Gamma}^{n}\right).$$

Finalmente encontramos

$$\boldsymbol{\lambda}_{\Gamma}^{n+1}-\boldsymbol{\lambda}_{\Gamma}^{n}=\theta\left(\boldsymbol{S}^{(1)}+\boldsymbol{S}^{(2)}\right)\left(\boldsymbol{d}_{\Gamma}-\boldsymbol{F}\,\boldsymbol{\lambda}_{\Gamma}^{n}\right).$$

El precondicionador para el sistema (1.5), éste sería la matriz

$$\boldsymbol{S}\,\boldsymbol{F}=\boldsymbol{S}\left[\left(\boldsymbol{S}^{(1)}\right)^{-1}+\left(\boldsymbol{S}^{(2)}\right)^{-1}\right]=\left(\boldsymbol{S}^{(1)}+\boldsymbol{S}^{(2)}\right)\left[\left(\boldsymbol{S}^{(1)}\right)^{-1}+\left(\boldsymbol{S}^{(2)}\right)^{-1}\right].$$

A este método se le conoce como FETI (*Finite Element Tearing and Interconnect*), específicamente FETI con precondicionador [Tose05 p14].

El método FETI se puede reformular como un problema de minimización con restricciones, minimizando la suma de las formas de energía de los dos subproblemas sujetos a la restricción $\boldsymbol{x}_{\Gamma}^{(1)}-\boldsymbol{x}_{\Gamma}^{(2)}=\boldsymbol{0}$, tendremos entonces que $\boldsymbol{\lambda}_{\Gamma}$ formará un vector de multiplicadores de Lagrange [Li05], a este nuevo método se le conoce como FETI-DP (FETI *Dual-Primal*) [Farh01]. Una descripción profunda de estos métodos puede encontrarse en [Tose05], [Mand05] y en [Widl08].

2. Deformación elástica de sólidos con el método de los elementos finitos

Vamos a describir la teoría de elasticidad de sólidos en dos y tres dimensiones. En este desarrollo modelaremos sólidos formados por materiales homogéneos e isotrópicos.

2.1. Elasticidad bidimensional

2.1.1. Esfuerzos y deformaciones

Para una estructura plana con esfuerzos y deformaciones aplicados en el mismo plano, podemos utilizar la hipótesis de que, para un mismo material, las deformaciones perpendiculares al plano serán de la misma magnitud. Así, podemos omitir esta dimensión y trabajar sólo en el plano, asumiendo que el campo de desplazamientos está perfectamente definido si se conocen los desplazamientos en las coordenadas x e y en todos sus puntos [Bote03 p8].

El vector de desplazamientos en un punto se define como

$$\boldsymbol{u}(x,y)=\begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix}, \tag{2.1}$$

en donde $u(x,y)$ y $v(x,y)$ representan los desplazamientos de un punto en las coordenadas x e y respectivamente.

A partir del campo de desplazamientos (2.1) se deducen las deformaciones haciendo uso de la teoría general de elasticidad [Esch97], el vector de deformaciones es entonces

$$\boldsymbol{\varepsilon}=\begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix}=\begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y}+\dfrac{\partial v}{\partial x} \end{bmatrix}=\begin{bmatrix} \dfrac{\partial}{\partial x} & 0 \\ 0 & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}, \tag{2.2}$$

donde ε_x y ε_y son las deformaciones normales y γ_{xy} la deformación por cizalladura. Con respecto a la deformación longitudinal ε_z hay que señalar que en el caso de deformación plana se utiliza la hipótesis de que es nula. Por otra parte, en un estado de esfuerzo dicha deformación no es nula, pero sí se supone que

lo es σ_z (la componente del esfuerzo perpendicular al plano). Por consiguiente, en ninguno de los dos casos hay que considerar la deformación ε_z ya que no interviene en las ecuaciones del trabajo de deformación al ser el producto $\sigma_z \varepsilon_z$ nulo. También consideramos que $\gamma_{xz}=\gamma_{yz}=0$.

De la ecuación (2.2) se deduce que los esfuerzos tangenciales τ_{xz} y τ_{yz} son nulos. Usando la misma hipótesis con respecto a la deformación ε_z, el esfuerzo σ_z no interviene, por tanto el vector de esfuerzos será

$$\boldsymbol{\sigma}=\begin{bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{bmatrix}, \tag{2.3}$$

con σ_x y σ_y esfuerzos normales y τ_{xy} el esfuerzo tangencial.

La relación entre esfuerzos y deformaciones se deduce de la ecuación constitutiva de la elasticidad tridimensional [Esch97], con las hipótesis para ε_z, σ_z y $\gamma_{xz}=\gamma_{yz}=0$ mencionadas anteriormente. Se deduce entonces que la relación matricial entre esfuerzos y deformaciones está dada por

$$\boldsymbol{\sigma}=\boldsymbol{D}\,\boldsymbol{\varepsilon}. \tag{2.4}$$

En el caso de considerar esfuerzos iniciales y deformaciones iniciales [Zien05 p25], debidos a cambios de temperatura, encogimiento, crecimiento de cristales, esfuerzos residuales iniciales, etc., utilizamos la forma más general de (2.4) que es

$$\boldsymbol{\sigma}=\boldsymbol{D}\left(\boldsymbol{\varepsilon}-\boldsymbol{\varepsilon}_0\right)+\boldsymbol{\sigma}_0. \tag{2.5}$$

La matriz $\boldsymbol{D}$ se conoce como matriz constitutiva o matriz de constantes elásticas. Del Teorema de Maxwell-Betti se deduce que $\boldsymbol{D}$ es siempre simétrica [Esch97 p46]. En el caso de elasticidad isotrópica

$$\boldsymbol{D}=\frac{E}{1-\nu^2}\begin{bmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & (1-\nu)/2\end{bmatrix},$$

donde E es el módulo de Young y ν el coeficiente de Poisson.

2.1.2. Principio de trabajos virtuales

La expresión integral de equilibrio en problemas de elasticidad bidimensional puede obtenerse haciendo uso del principio de los trabajos virtuales [Zien05 pp65-71]. Teniendo en cuenta los esfuerzos y deformaciones que contribuyen al trabajo virtual de la estructura, la expresión del principio de trabajos virtuales puede escribirse como

$$\int\int_A \left(\delta\varepsilon_x\sigma_x+\delta\varepsilon_y\sigma_y+\delta\gamma_y\tau_y\right)t\,dA=\int\int_A \left(\delta u b_x+\delta v b_y\right)t\,dA+\int_l \left(\delta u\,t_x+\delta v\,t_y\right)t\,ds+\sum_i \left(\delta u_i U_x+\delta v_i V_y\right). \tag{2.6}$$

A la izquierda de la ecuación está representado el trabajo que los esfuerzos σ_x, σ_y y τ_{xy} realizan sobre las deformaciones virtuales $\delta\varepsilon_x$, $\delta\varepsilon_y$ y $\delta\gamma_{xy}$. El primer miembro a la derecha de la igualdad representa las fuerzas repartidas por unidad de volumen b_x y b_y. El segundo miembro indica las fuerzas repartidas sobre el contorno t_x y t_y. Y finalmente el tercer miembro las fuerzas puntuales U_i y V_i sobre los desplazamientos virtuales δu y δv. A y l son el área y el contorno de la sección transversal del sólido y t su espesor. En problemas de esfuerzo plano, t coincide con el espesor real, mientras que en problemas de deformación plana es usual asignar a t un valor unidad.

Definiendo el vector de fuerzas puntuales

$$\boldsymbol{q}_i = \begin{bmatrix} U_i \\ V_i \end{bmatrix},$$

el vector de fuerzas sobre el contorno

$$\boldsymbol{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

y el vector de fuerzas másicas como

$$\boldsymbol{b} = \begin{bmatrix} b_x \\ b_y \end{bmatrix},$$

podemos entonces expresar (2.6) en forma matricial

$$\int\int_A \delta \boldsymbol{\varepsilon}^{\mathrm{T}} \boldsymbol{\sigma}\, t\, dA = \int\int_A \delta \boldsymbol{u}^{\mathrm{T}} \boldsymbol{b}\, t\, dA + \oint_l \delta \boldsymbol{u}^{\mathrm{T}} \boldsymbol{t}\, t\, ds + \sum_i \delta \boldsymbol{u}_i^{\mathrm{T}} \boldsymbol{q}_i. \tag{2.7}$$

De las ecuaciones (2.2) y (2.4) se observa que en las integrales del principio de trabajos virtuales sólo intervienen las primeras derivadas de los desplazamientos, lo que implica que se requiere continuidad de clase C_0 en la aproximación de elementos finitos.

2.1.3. Discretización en elementos finitos

La figura 2.1 muestra el dominio Ω de un problema, el cual se analiza bajo las hipótesis de elasticidad bidimensional, cuenta además con ciertas condiciones de frontera de Dirichlet Γ_u y de Neumann Γ_q.

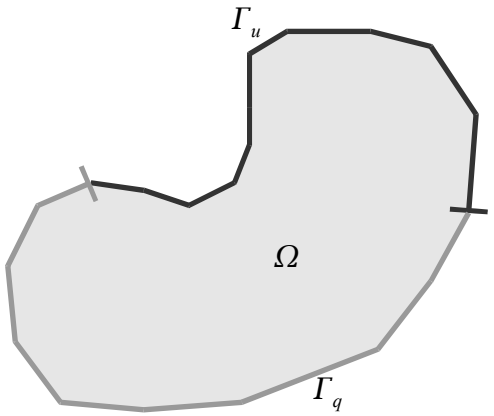


Figura 2.1. Dominio del problema.

Para nuestro desarrollo vamos a utilizar para discretizar el dominio elementos triangulares de tres nodos, los cuales son sencillos de visualizar. En la figura 2.2 vemos la discretización del dominio Ω.

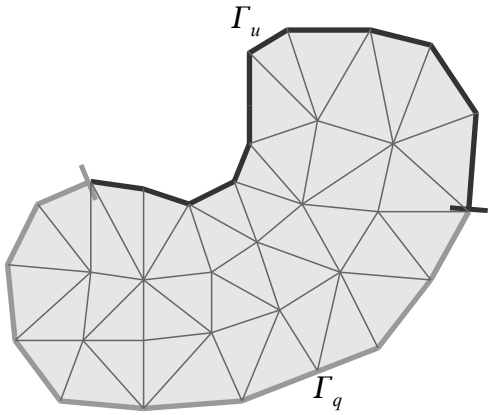


Figura 2.2. Discretización (mallado) del dominio.

La malla de elementos finitos representa una idealización de la geometría real. Por consiguiente, el análisis por elementos finitos reproduce el comportamiento de la malla escogida y no el de la estructura real. Solamente comprobando la convergencia de la solución podemos estimar el grado de aproximación de la solución de elementos finitos a la exacta.

2.1.4. Funciones de forma

Un elemento triangular de tres nodos se caracteriza por los números de sus nodos 1, 2, y 3, con sus respectivas coordenadas. Esta numeración es local y tiene que crearse una correspondencia con la numeración global.

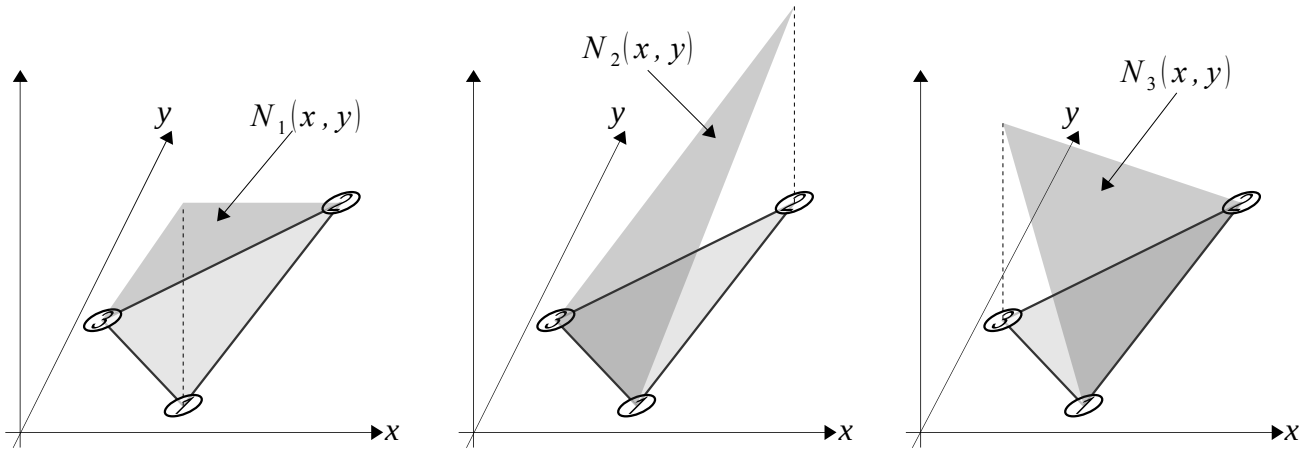


Figura 2.3. Funciones de forma para un elemento finito triangular de tres nodos.

Tomando un elemento aislado, como el de la figura 2.3, podemos expresar los desplazamientos cartesianos de un punto cualquiera en el interior del elemento en función de los desplazamientos de sus nodos introduciendo las funciones de norma N_i, con $i=1,2,3$

$$u(x,y)=\sum_{i=1}^{3} N_i(x,y)u_i,$$

$$v(x,y)=\sum_{i=1}^{3} N_i(x,y)v_i,$$

donde u_i y v_i son los valores discretos de desplazamiento en los nodos. Una función de forma N_i tiene que cumplir la condición de valer uno en la coordenada del nodo *i* y cero en los nodos $j \neq i$.

Para obtener unas funciones de forma más fáciles de trabajar, es conveniente hacer un mapeo a un espacio normalizado, como se muestra en la figura 2.4.

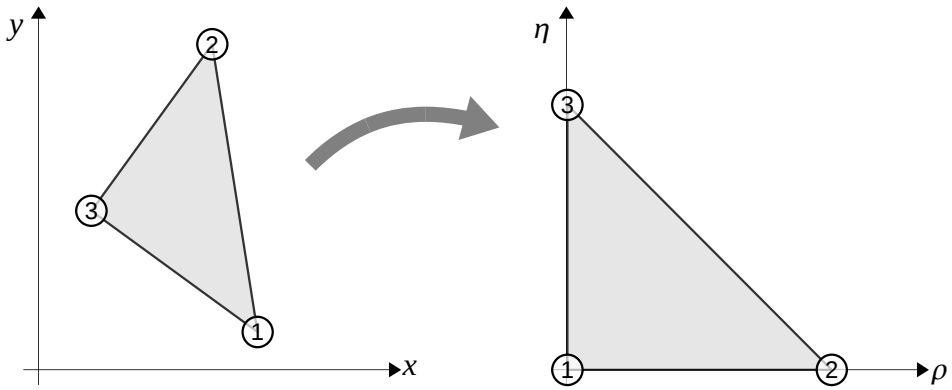


Figura 2.4. Mapeo de un elemento a un espacio normalizado.

Definamos entonces a u y v en términos de las coordenadas normalizadas ρ y η, como sigue

$$u(\rho,\eta)=\sum_{i=1}^{3} N_i(\rho,\eta)\,u_i,$$

$$v(\rho,\eta)=\sum_{i=1}^{3} N_i(\rho,\eta)\,v_i,$$

con lo que podemos definir las funciones de forma como

$$N_1(\rho,\eta)=1-\rho-\eta,$$

$$N_2(\rho,\eta)=\rho,$$

$$N_3(\rho,\eta)=\eta.$$

Las funciones de cambio de coordenada serán

$$x(\rho,\eta)=\sum_{i=1}^{3} N_i(\rho,\eta)\,x_i,$$

$$y(\rho,\eta)=\sum_{i=1}^{3} N_i(\rho,\eta)\,y_i.$$

2.1.5. Discretización de los campo de deformaciones y esfuerzos

Podemos así expresar las componentes de la deformacion (2.2) como

$$\frac{\partial u}{\partial x}=\sum_i \frac{\partial N_i}{\partial x}u_i,\ \frac{\partial u}{\partial x}=\sum_i \frac{\partial N_i}{\partial x}u_i,\ \frac{\partial v}{\partial x}=\sum_i \frac{\partial N_i}{\partial x}v_i,\ \frac{\partial v}{\partial y}=\sum_i \frac{\partial N_i}{\partial y}v_i.$$

Para aplicar un cambio de variable en las primeras derivadas requeriremos del jacobiano

$$\begin{pmatrix} \frac{\partial N_i}{\partial \rho} \\ \frac{\partial N_i}{\partial \eta} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{\partial x}{\partial \rho} & \frac{\partial y}{\partial \rho} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{pmatrix}}_{\boldsymbol{J}^{\mathrm{e}}} \begin{pmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{pmatrix}.$$

Sí $\det \boldsymbol{J}^{\mathrm{e}} \neq 0$, entonces

$$\begin{pmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{pmatrix} = \left(\boldsymbol{J}^{\mathrm{e}}\right)^{-1} \begin{pmatrix} \frac{\partial N_i}{\partial \rho} \\ \frac{\partial N_i}{\partial \eta} \end{pmatrix}.$$

Ahora podemos definir el vector de deformaciones (2.2) como

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} = \sum_i \begin{bmatrix} \frac{\partial N_i}{\partial x} u_i \\ \frac{\partial N_i}{\partial y} v_i \\ \frac{\partial N_i}{\partial y} u_i + \frac{\partial N_i}{\partial x} v_i \end{bmatrix} = \sum_i \underbrace{\begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} \end{bmatrix}}_{\boldsymbol{B}_i} \underbrace{\begin{bmatrix} u_i \\ v_i \end{bmatrix}}_{\boldsymbol{u}_i}, \tag{2.8}$$

donde $\boldsymbol{B}_i$ es la matriz de deformación del nodo i. Visto en forma más compacta

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \boldsymbol{B}_1 \boldsymbol{B}_2 & \cdots & \boldsymbol{B}_n \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_n \end{bmatrix},$$

finalmente como

$$\boldsymbol{\varepsilon} = \boldsymbol{B}\,\boldsymbol{u}. \tag{2.9}$$

Puede verse entonces que la matriz de deformación $\boldsymbol{B}$ está compuesta de tantas sub-matrices $\boldsymbol{B}_i$ como nodos tienen el elemento.

La expresión discretizada del vector de esfuerzos (2.3) se obtiene a partir de (2.4)

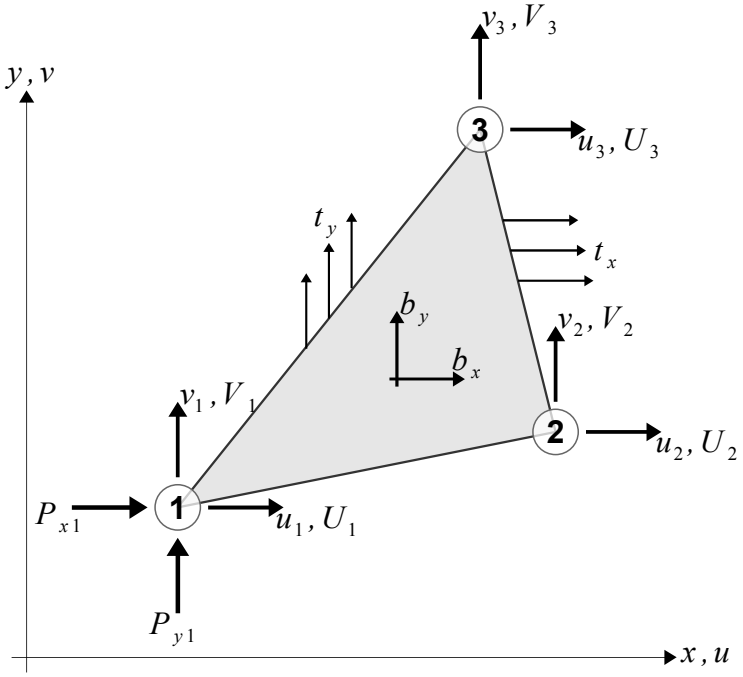
$$\boldsymbol{\sigma} = \boldsymbol{D}\,\boldsymbol{B}\,\boldsymbol{u}.$$

Puede observarse de (2.8) que la matriz de deformación $\boldsymbol{B}$ del elemento triangular de tres nodos es constante, lo que implica que las deformaciones y esfuerzos son constantes en todo el elemento. Esto es consecuencia directa del campo de desplazamientos lineal escogido, cuyos gradientes son obviamente constantes. Por consiguiente, en zonas de alta concentración de esfuerzos será necesaria utilizar una malla tupida para aproximar la solución de esfuerzos con suficiente precisión.

2.1.6. Ecuaciones de equilibrio de la discretización

Para la obtención de las ecuaciones de equilibrio de la discretización partimos de la expresión del principio de trabajos virtuales aplicada al equilibro de un elemento aislado.

Vamos a suponer que sobre un elemento, como el de la figura 2.5, actúan fuerzas másicas repartidas por unidad de área $\boldsymbol{b}$ y en sus lados fuerzas de superficie por unidad de longitud $\boldsymbol{t}$. Las fuerzas de superficie pueden ser de dos tipos: debidas a fuerzas exteriores que actúan sobre los lados del elemento que forman parte del contorno exterior de la estructura o debidas a las fuerzas de interacción entre elementos que se transmite a través de sus lados comunes. Éstas últimas pueden ignorarse desde un inicio, debido a que se anulan cuando se realiza el ensamblado de todos los elementos.



Fuerzas másicas $\boldsymbol{b}=\begin{bmatrix} b_x \\ b_y \end{bmatrix}$

Fuerzas de superficie $\boldsymbol{t}=\begin{bmatrix} t_x \\ t_y \end{bmatrix}$

Fuerzas nodales de equilibrio $\boldsymbol{q}_i=\begin{bmatrix} U_i \\ V_i \end{bmatrix}$

Fuerzas nodales exteriores $\boldsymbol{P}_i=\begin{bmatrix} P_{xi} \\ P_{yi} \end{bmatrix}$

Figura 2.5. Fuerzas sobre un elemento triangular de tres nodos.

Partiendo de la suposición de que el equilibrio del elemento se establece únicamente en los nodos, podemos definir unas fuerzas puntuales de que actúen sobre los nodos que equilibren las fuerzas debidas a la deformación del elemento y al resto de las fuerzas que actúan sobre el mismo. Hacemos entonces uso del principio de trabajos virtuales (2.7) aplicado ahora al elemento

$$\int\int_{A^e} \delta\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma}\, t\, dA^e - \int\int_{A^e} \delta\boldsymbol{u}^{\mathrm{T}}\boldsymbol{b}\, t\, dA^e - \oint_{l^e} \delta\boldsymbol{u}^{\mathrm{T}}\boldsymbol{t}\, t\, ds^e = \delta\boldsymbol{u}^{\mathrm{T}}\boldsymbol{q}^e,$$

reescribíendola utilizando (2.9)

$$\int\int_{A^e} \delta\boldsymbol{u}^{\mathrm{T}}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{\sigma}\, t\, dA^e - \int\int_{A^e} \delta\boldsymbol{u}^{\mathrm{T}}\boldsymbol{b}\, t\, dA^e - \oint_{l^e} \delta\boldsymbol{u}^{\mathrm{T}}\boldsymbol{t}\, t\, ds^e = \delta\boldsymbol{u}^{\mathrm{T}}\boldsymbol{q}^e,$$

tomando además en cuenta que los desplazamientos virtuales son arbitrarios, y el espesor es constante,

$$t\int\int_{A^e} \boldsymbol{B}^{\mathrm{T}}\boldsymbol{\sigma}\, dA^e - t\int\int_{A^e} \boldsymbol{b}\, dA^e - t\oint_{l^e} \boldsymbol{t}\, ds^e = \boldsymbol{q}^e. \tag{2.10}$$

La ecuación (2.10) expresa el equilibrio entre las fuerzas nodales de equilibrio y las fuerzas debidas a la deformación del elemento (la primer integral), las fuerzas másicas (segunda integral) y las de superficie (tercera integral). Sustituyendo el vector de esfuerzos $\boldsymbol{\sigma}$ por su valor en función de los desplazamientos nodales utilizando la forma general (2.5)

$$t\int_{A^e}\int \boldsymbol{B}^{\mathrm{T}}\left[\boldsymbol{D}\left(\boldsymbol{\varepsilon}-\boldsymbol{\varepsilon}_0\right)+\boldsymbol{\sigma}_0\right]dA^{\mathrm{e}}-t\int_{A^e}\int \boldsymbol{b}\,dA^{\mathrm{e}}-t\oint_{l^e}\boldsymbol{t}\,ds^{\mathrm{e}}=\boldsymbol{q}^{\mathrm{e}},$$

separando las integrales

$$t\int_{A^e}\int \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}\,\boldsymbol{\varepsilon}\,dA^{\mathrm{e}}-t\int_{A^e}\int \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}\,\boldsymbol{\varepsilon}_0\,dA^{\mathrm{e}}+t\int_{A^e}\int \boldsymbol{B}^{\mathrm{T}}\boldsymbol{\sigma}_0\,dA^{\mathrm{e}}-t\int_{A^e}\int \boldsymbol{b}\,dA^{\mathrm{e}}-t\oint_{l^e}\boldsymbol{t}\,ds^{\mathrm{e}}=\boldsymbol{q}^{\mathrm{e}},$$

aplicando de nuevo (2.9)

$$\underbrace{t\int_{A^e}\int \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}\,\boldsymbol{B}\,dA^{\mathrm{e}}}_{\boldsymbol{K}^{\mathrm{e}}}\boldsymbol{u}=\underbrace{t\int_{A^e}\int \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}\,\boldsymbol{\varepsilon}_0\,dA^{\mathrm{e}}}_{\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{\varepsilon}}}-\underbrace{t\int_{A^e}\int \boldsymbol{B}^{\mathrm{T}}\boldsymbol{\sigma}_0\,dA^{\mathrm{e}}}_{\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{\sigma}}}+\underbrace{t\int_{A^e}\int \boldsymbol{b}\,dA^{\mathrm{e}}}_{\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{b}}}+\underbrace{t\oint_{l^e}\boldsymbol{t}\,ds^{\mathrm{e}}}_{\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{t}}}+\boldsymbol{q}^{\mathrm{e}}, \qquad (2.11)$$

donde $\boldsymbol{K}^{\mathrm{e}}$ es la matriz de rigidez del elemento y tenemos un conjunto de fuerzas nodales equivalentes debidas a: deformaciones iniciales $\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{\varepsilon}}$, esfuerzos iniciales $\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{\sigma}}$, fuerzas másicas $\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{b}}$, fuerzas en la superficie $\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{t}}$.

Definiendo el vector de fuerzas nodales equivalentes del elemento como

$$\boldsymbol{f}^{\mathrm{e}}=\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{\varepsilon}}-\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{\sigma}}+\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{b}}+\boldsymbol{f}^{\mathrm{e}}_{\boldsymbol{t}},$$

podemos expresar (2.11) como un sistema de ecuaciones para el elemento

$$\boldsymbol{K}^{\mathrm{e}}\boldsymbol{u}=\boldsymbol{f}^{\mathrm{e}}+\boldsymbol{q}^{\mathrm{e}}.$$

La ecuación de equilibrio total de la malla se obtiene estableciendo que la suma de las fuerzas nodales de equilibrio en cada nodo debe ser igual a la fuerza nodal exterior, es decir

$$\sum_{\mathrm{e}}\boldsymbol{q}^{\mathrm{e}}_j=\boldsymbol{q}_j,$$

esta es la suma de las contribuciones de los vectores de fuerzas nodales de equilibrio de los distintos elementos que comparten el nodo con numeración global j, y $\boldsymbol{p}_j$ representa el vector de fuerzas puntuales actuando en dicho nodo.

Las ecuaciones de equilibrio de la malla se obtienen a partir de las contribuciones de las matrices elementales de rigidez y de los vectores de fuerzas nodales equivalentes de los diferentes elementos. Así pues, tras el ensamblaje, la ecuación matricial global se puede escribir como

$$\boldsymbol{K}\,\boldsymbol{u}=\boldsymbol{f},$$

donde $\boldsymbol{K}$ es la matriz de rigidez, $\boldsymbol{u}$ es el vector de desplazamientos nodales y $\boldsymbol{f}$ el vector de fuerzas nodales equivalentes de toda la malla.

Recordemos de nuevo que las fuerzas nodales de equilibrio debidas al las fuerzas de interacción entre los contornos de los elementos adyacentes se anulan en el ensamblaje, debido a que dichas fuerzas tienen igual magnitud y dirección, pero sentidos opuestos en cada elemento. A efectos prácticos, solamente hay que considerar el efecto de las fuerzas de superficie cuando se trate de fuerzas exteriores actuantes sobre lados de elementos que pertenezcan al contorno de la estructura.

2.1.7. Ensamblado de la matriz de rigidez

El primer paso es generar una malla de elementos para el dominio e identificar los nodos.

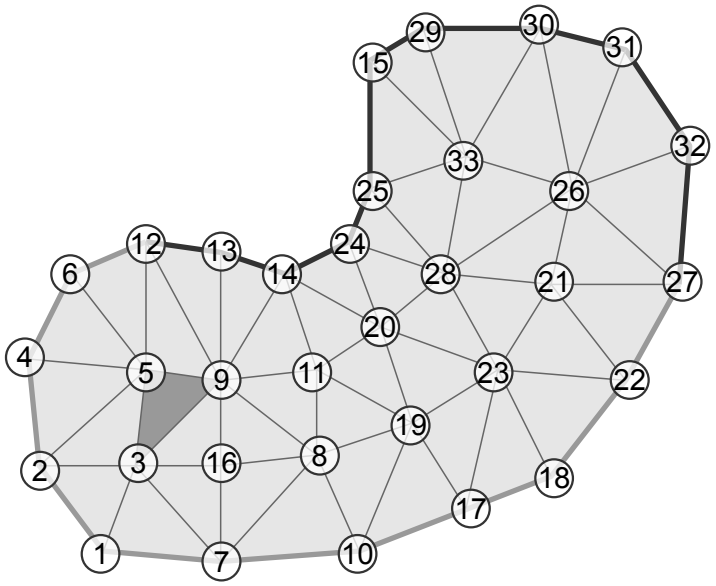


Figura 2.6. Numeración global de los nodos en el dominio.

Se calcula la matriz elemental $\boldsymbol{K}^{e}$ y el vector $\boldsymbol{f}^{e}$ para cada elemento. Por ejemplo, en la figura 2.6 se destaca el elemento con nodos 3, 5 y 9. Su ensamblaje es mostrado en la figura 2.7.

$$
\begin{array}{cc|cccccc}
\text{Global} \rightarrow & & 3_x & 3_y & 9_x & 9_y & 5_x & 5_y \\
\text{Local} \rightarrow & & 1_x & 1_y & 2_x & 2_y & 3_x & 3_y \\
\downarrow & \downarrow & & & & & & \\
3_x & 1_x & K^{e}_{11} & K^{e}_{12} & K^{e}_{13} & K^{e}_{14} & K^{e}_{15} & K^{e}_{16} \\
3_y & 1_y & K^{e}_{21} & K^{e}_{22} & K^{e}_{23} & K^{e}_{24} & K^{e}_{25} & K^{e}_{26} \\
9_x & 2_x & K^{e}_{31} & K^{e}_{32} & K^{e}_{33} & K^{e}_{34} & K^{e}_{35} & K^{e}_{36} \\
9_y & 2_y & K^{e}_{41} & K^{e}_{42} & K^{e}_{43} & K^{e}_{44} & K^{e}_{45} & K^{e}_{46} \\
5_x & 3_x & K^{e}_{51} & K^{e}_{52} & K^{e}_{53} & K^{e}_{54} & K^{e}_{55} & K^{e}_{56} \\
5_y & 3_y & K^{e}_{61} & K^{e}_{62} & K^{e}_{63} & K^{e}_{64} & K^{e}_{65} & K^{e}_{66}
\end{array}
\begin{bmatrix} u^{e}_1 \\ u^{e}_2 \\ u^{e}_3 \\ u^{e}_4 \\ u^{e}_5 \\ u^{e}_6 \end{bmatrix}
=
\begin{bmatrix} f^{e}_1 \\ f^{e}_2 \\ f^{e}_3 \\ f^{e}_4 \\ f^{e}_5 \\ f^{e}_6 \end{bmatrix}
$$

Figura 2.7. Numeración global y local al ensamblar la matriz elemental.

Los valores se sumarán a la matriz de rigidez $\boldsymbol{K}$ y al vector $\boldsymbol{f}$, como se muestra en la figura 2.8.

$$\begin{bmatrix}
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & K^e_{11} & K^e_{12} & \circ & \circ & K^e_{15} & K^e_{16} & \circ & \circ & \circ & \circ & \circ & \circ & K^e_{13} & K^e_{14} & \circ & \cdots \\
\circ & \circ & \circ & \circ & K^e_{21} & K^e_{22} & \circ & \circ & K^e_{25} & K^e_{26} & \circ & \circ & \circ & \circ & \circ & \circ & K^e_{23} & K^e_{24} & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & K^e_{51} & K^e_{52} & \circ & \circ & K^e_{55} & K^e_{56} & \circ & \circ & \circ & \circ & \circ & \circ & K^e_{53} & K^e_{54} & \circ & \cdots \\
\circ & \circ & \circ & \circ & K^e_{61} & K^e_{62} & \circ & \circ & K^e_{65} & K^e_{66} & \circ & \circ & \circ & \circ & \circ & \circ & K^e_{63} & K^e_{64} & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & K^e_{31} & K^e_{32} & \circ & \circ & K^e_{35} & K^e_{36} & \circ & \circ & \circ & \circ & \circ & \circ & K^e_{33} & K^e_{34} & \circ & \cdots \\
\circ & \circ & \circ & \circ & K^e_{41} & K^e_{42} & \circ & \circ & K^e_{45} & K^e_{46} & \circ & \circ & \circ & \circ & \circ & \circ & K^e_{43} & K^e_{44} & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix}
\begin{bmatrix} \circ \\ \circ \\ \circ \\ \circ \\ u^e_1 \\ u^e_2 \\ \circ \\ \circ \\ u^e_5 \\ u^e_6 \\ \circ \\ \circ \\ \circ \\ \circ \\ \circ \\ \circ \\ u^e_3 \\ u^e_4 \\ \circ \\ \vdots \end{bmatrix}
=
\begin{bmatrix} \circ \\ \circ \\ \circ \\ \circ \\ f^e_1 \\ f^e_2 \\ \circ \\ \circ \\ f^e_5 \\ f^e_6 \\ \circ \\ \circ \\ \circ \\ \circ \\ \circ \\ \circ \\ f^e_3 \\ f^e_4 \\ \circ \\ \vdots \end{bmatrix}$$

Figura 2.8. Ensamblaje de la matriz elemental en la matriz de rigidez.

Una condición de frontera de Dirichlet, u_i fijo, implicarán eliminar el i-ésimo renglón y la i-ésima columna de la matriz de rigidez y el i-ésimo renglón del vector . Para poder hace esto, hay que compensar en el vector $\boldsymbol{f}$ de esta forma

$$(\boldsymbol{f})_j \leftarrow (\boldsymbol{f})_j - (\boldsymbol{K})_{ij} (\boldsymbol{u})_j, \ \forall \ j \neq i$$

La matriz de rigidez resultante no tendrá un ancho de banda predefinido.

2.2. Elasticidad tridimensional

La formulación de los elementos finitos tridimensionales es análoga a la formulación bidimensional, por lo cual vamos a describirla brevemente. Trabajaremos ahora con elementos en tres dimensiones, en particular con tetraedros con cuatro nodos.

2.2.1. Esfuerzos y deformaciones

Iniciamos definiendo el vector de desplazamientos

$$\boldsymbol{u}(x,y,z) = \begin{bmatrix} u(x,y,z) \\ v(x,y,z) \\ w(x,y,z) \end{bmatrix}.$$

Introducimos ahora el vector de deformaciones $\boldsymbol{\varepsilon}$, el cual está dado por

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_y \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial w}{\partial z} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \\ \dfrac{\partial v}{\partial z} + \dfrac{\partial w}{\partial y} \\ \dfrac{\partial w}{\partial x} + \dfrac{\partial u}{\partial z} \end{bmatrix},$$

donde ε_x, ε_y y ε_z son las deformaciones normales, mientras que γ_{xy}, γ_{yz} y γ_{zx} son las deformaciones por cizalladura.

El vector de tensiones $\boldsymbol{\sigma}$ se define como

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{bmatrix},$$

donde σ_x, σ_y y σ_z son las tensiones normales y τ_{xy}, τ_{yz} y τ_{zx} las tensiones tangenciales.

Para un medio homogéneo e isotrópico la matriz constitutiva es [Zien05 p196]

$$\boldsymbol{D} = \frac{E(1-v)}{(1+v)(1-2\mathrm{v})} \begin{bmatrix} 1 & \dfrac{v}{(1-v)} & \dfrac{v}{(1-v)} & 0 & 0 & 0 \\ \dfrac{v}{(1-v)} & 1 & \dfrac{v}{(1-v)} & 0 & 0 & 0 \\ \dfrac{v}{(1-v)} & \dfrac{v}{(1-v)} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \dfrac{(1-2\mathrm{v})}{2(1-v)} & 0 & 0 \\ 0 & 0 & 0 & 0 & \dfrac{(1-2\mathrm{v})}{2(1-v)} & 0 \\ 0 & 0 & 0 & 0 & 0 & \dfrac{(1-2\mathrm{v})}{2(1-v)} \end{bmatrix},$$

donde E es el módulo de Young y ν es el coeficiente de Poisson.

2.2.2. Funciones de forma

El vector

$$\boldsymbol{u}_i(x,y,z)=\begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix}$$

representa los desplazamientos de un nodo i.

Introduciendo las funciones de interpolación

$$u(x,y,z)=\sum_i N_i(x,y,z)u_i,$$

$$v(x,y,z)=\sum_i N_i(x,y,z)v_i,$$

$$w(x,y,z)=\sum_i N_i(x,y,z)w_i.$$

Para realizar el mapeo al espacio normalizado, tenemos

$$u(\rho,\eta,\zeta)=\sum_{i=1}^{4} N_i(\rho,\eta,\zeta)u_i,$$

$$v(\rho,\eta,\zeta)=\sum_{i=1}^{4} N_i(\rho,\eta,\zeta)v_i,$$

$$w(\rho,\eta,\zeta)=\sum_{i=1}^{4} N_i(\rho,\eta,\zeta)w_i;$$

donde u, v y w representan los valores discretos de los desplazamientos en los nodos del elemento, N_i son las funciones de forma.

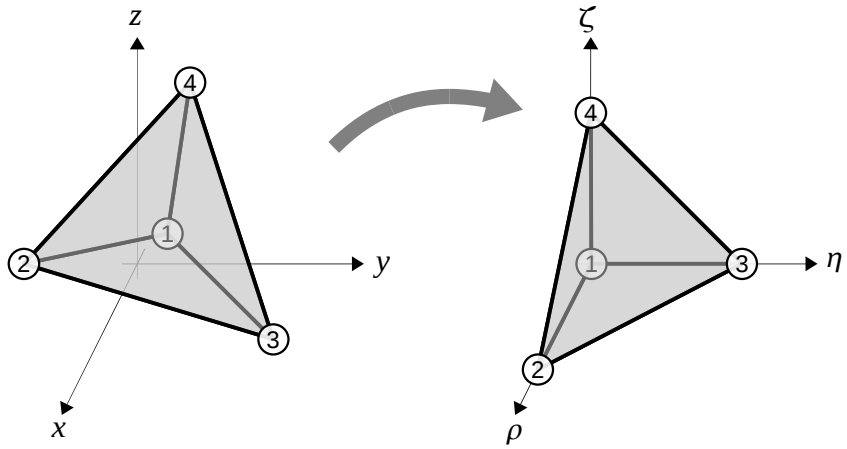


Figura 2.9. Mapeo al espacio normalizado.

Elegimos las siguientes funciones lineales para el mapeo

$$N_1(\rho,\eta,\zeta)=1-\rho-\eta-\zeta,$$

$$N_2(\rho,\eta,\zeta)=\rho,$$

$$N_3(\rho,\eta,\zeta)=\eta,$$
$$N_4(\rho,\eta,\zeta)=\zeta.$$

Las funciones de cambio de coordenada son entonces

$$x(\rho,\eta,\zeta)=\sum_{i=1}^{4} N_i(\rho,\eta,\zeta)\,x_i,$$
$$y(\rho,\eta,\zeta)=\sum_{i=1}^{4} N_i(\rho,\eta,\zeta)\,y_i,$$
$$z(\rho,\eta,\zeta)=\sum_{i=1}^{4} N_i(\rho,\eta,\zeta)\,z_i;$$

donde x, y y z son las coordenadas de los vértices del elemento.

2.2.3. Discretización de los campo de deformaciones y esfuerzos

Las deformaciones entonces son

$$\frac{\partial u}{\partial x}=\sum_i \frac{\partial N_i}{\partial x}u_i,\ \frac{\partial u}{\partial y}=\sum_i \frac{\partial N_i}{\partial y}u_i,\ \frac{\partial u}{\partial z}=\sum_i \frac{\partial N_i}{\partial z}u_i;$$
$$\frac{\partial v}{\partial x}=\sum_i \frac{\partial N_i}{\partial x}v_i,\ \frac{\partial v}{\partial y}=\sum_i \frac{\partial N_i}{\partial y}v_i,\ \frac{\partial v}{\partial z}=\sum_i \frac{\partial N_i}{\partial z}v_i;$$
$$\frac{\partial w}{\partial x}=\sum_i \frac{\partial N_i}{\partial x}w_i,\ \frac{\partial w}{\partial y}=\sum_i \frac{\partial N_i}{\partial y}w_i,\ \frac{\partial w}{\partial z}=\sum_i \frac{\partial N_i}{\partial z}w_i.$$

Aplicamos la regla de la cadena a primeras derivadas para obtener el jacobiano $\boldsymbol{J}^{\mathrm{e}}$, éste es

$$\begin{Bmatrix} \frac{\partial N_i}{\partial \rho} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{Bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial x}{\partial \rho} & \frac{\partial y}{\partial \rho} & \frac{\partial z}{\partial \rho} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix}}_{\boldsymbol{J}^{\mathrm{e}}} \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix},$$

si $\det \boldsymbol{J}^{\mathrm{e}} \neq 0$, entonces

$$\begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix} = \left(\boldsymbol{J}^{\mathrm{e}}\right)^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \rho} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{Bmatrix},$$

De esta forma

$$\boldsymbol{\varepsilon}=\begin{bmatrix}\varepsilon_x\\ \varepsilon_y\\ \varepsilon_y\\ \gamma_{xy}\\ \gamma_{yz}\\ \gamma_{zx}\end{bmatrix}=\sum_i \begin{bmatrix}\dfrac{\partial N_i}{\partial x}u_i\\ \dfrac{\partial N_i}{\partial y}v_i\\ \dfrac{\partial N_i}{\partial z}w_i\\ \dfrac{\partial N_i}{\partial y}u_i+\dfrac{\partial N_i}{\partial x}v_i\\ \dfrac{\partial N_i}{\partial z}v_i+\dfrac{\partial N_i}{\partial y}w_i\\ \dfrac{\partial N_i}{\partial x}w_i+\dfrac{\partial N_i}{\partial z}u_i\end{bmatrix},$$

ésta expresión se puede escribir como

$$\boldsymbol{\varepsilon}=\sum_i \begin{bmatrix}\dfrac{\partial N_i}{\partial x} & 0 & 0\\ 0 & \dfrac{\partial N_i}{\partial y} & 0\\ 0 & 0 & \dfrac{\partial N_i}{\partial z}\\ \dfrac{\partial N_i}{\partial y} & \dfrac{\partial N_i}{\partial x} & 0\\ 0 & \dfrac{\partial N_i}{\partial z} & \dfrac{\partial N_i}{\partial y}\\ \dfrac{\partial N_i}{\partial z} & 0 & \dfrac{\partial N_i}{\partial x}\end{bmatrix}\begin{bmatrix}u_i\\ v_i\\ w_i\end{bmatrix}=\sum_i \boldsymbol{B}_i\boldsymbol{u}_i$$

y de forma más compacta como

$$\boldsymbol{\varepsilon}=\begin{bmatrix}\boldsymbol{B}_1 & \boldsymbol{B}_2 & \cdots & \boldsymbol{B}_n\end{bmatrix}\begin{bmatrix}\boldsymbol{u}_1\\ \boldsymbol{u}_2\\ \vdots\\ \boldsymbol{u}_n\end{bmatrix},$$

finalmente

$$\boldsymbol{\varepsilon}=\boldsymbol{B}\,\boldsymbol{u}.$$

2.2.4. Ecuaciones de equilibrio de la discretización

Definiendo el vector de fuerzas puntuales

$$\boldsymbol{q}_i = \begin{bmatrix} U_i \\ V_i \\ W_i \end{bmatrix},$$

el vector de fuerzas sobre el contorno

$$\boldsymbol{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

y el vector de fuerzas másicas como

$$\boldsymbol{b} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix},$$

podemos entonces expresar la ecuación del principio de trabajos virtuales como

$$\int_V \int \delta \boldsymbol{\varepsilon}^{\mathrm{T}} \boldsymbol{\sigma} \, dV = \int_V \int \delta \boldsymbol{u}^{\mathrm{T}} \boldsymbol{b} \, dV + \oint_s \delta \boldsymbol{u}^{\mathrm{T}} \boldsymbol{t} \, ds + \sum_i \delta \boldsymbol{u}_i^{\mathrm{T}} \boldsymbol{q}_i .$$

Realizando un desarrollo similar al caso bidimensional, podemos llegar a la ecuación de equilibrio discretizada para un elemento, ésta es

$$\underbrace{\int_{V^{\mathrm{e}}} \int \boldsymbol{B}^{\mathrm{T}} \boldsymbol{D} \boldsymbol{B} \, dV^{\mathrm{e}}}_{\boldsymbol{K}^{\mathrm{e}}} \boldsymbol{u} = \underbrace{\int_{V^{\mathrm{e}}} \int \boldsymbol{B}^{\mathrm{T}} \boldsymbol{D} \boldsymbol{\varepsilon}_0 \, dV^{\mathrm{e}}}_{\boldsymbol{f}_{\boldsymbol{\varepsilon}}^{\mathrm{e}}} - \underbrace{\int_{V^{\mathrm{e}}} \int \boldsymbol{B}^{\mathrm{T}} \boldsymbol{\sigma}_0 \, dV^{\mathrm{e}}}_{\boldsymbol{f}_{\boldsymbol{\sigma}}^{\mathrm{e}}} + \underbrace{\int_{V^{\mathrm{e}}} \int \boldsymbol{b} \, dV^{\mathrm{e}}}_{\boldsymbol{f}_{\boldsymbol{b}}^{\mathrm{e}}} + \underbrace{\oint_{s^{\mathrm{e}}} \boldsymbol{t} \, ds^{\mathrm{e}}}_{\boldsymbol{f}_{\boldsymbol{t}}^{\mathrm{e}}} + \boldsymbol{q}^{\mathrm{e}} .$$

3. Una aplicación de la paralelización con memoria compartida

3.1. Introducción

Nuestra primer aproximación para paralelizar la solución del sistema de ecuaciones resultante del método de elemento finito es utilizar el esquema de paralelización con memoria compartida. Veremos como este tipo de esquema se presta para la paralelización del método de gradiente conjugado.

Vamos a comenzar hablando de la arquitectura del procesamiento en paralelo, esto es porque tanto el *hardware* como el *software* utilizados para el cómputo en paralelo son mucho más complejos y sofisticados que los utilizado en el cómputo serial, lo que hace que en el cómputo en paralelo sea más dificil obtener buenos resultados sobre todo en las primeras implementaciones de los algoritmos.

Es necesario entender, por lo menos conceptualmente, cuales son las características tanto del *hardware* como del *software*, con el fin de poder diseñar algoritmos que saquen ventaja de esta forma de cómputo, de no hacerlo podemos hacer que nuestros algoritmos caigan en los múltiples cuellos de botella que presentan estas arquitecturas.

Podemos decir entonces que la arquitectura del *hardware* de cómputo en paralelo nos servirá de guía para diseñar el *software* para la paralelización de los algoritmos de solución de problemas de elemento finito.

3.2. Arquitectura del procesamiento en paralelo

La paralelización con memoria compartida se refiere a la utilización de computadoras con dos o más unidades de procesamiento que accesan a la misma memoria principal. La arquitectura más usada en este tipo de paralelización es con procesadores *multi-core*, es decir procesadores que tienen más de un núcleo (*core*) o unidad de procesamiento (CPU). También es factible tener computadoras con más de un procesador *multi-core* accesando la misma memoria (figura 3.1).

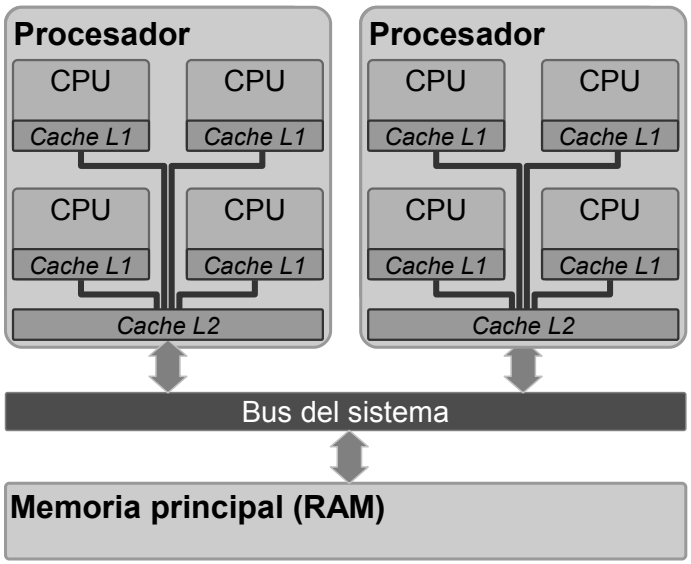


Figura 3.1. Configuración de un sistema multiprocesador multinúcleo

Al poder accesar la misma memoria es entonces posible hacer que los CPUs cooperen en la resolución de un problema. En la taxonomia de Flynn [Flyn72], esta estrategia es conocida como *Multiple Instructions-Multiple Data*, o por sus siglas MIMD.

La velocidad de operación de los procesadores es mucho mayor que la velocidad de acceso de la memoria RAM de las computadoras [Wulf95], esto es debido a que es muy costoso fabricar memoria de alta velocidad. Para solventar esta diferencia en velocidad, los procesadores modernos incluyen memoria *cache*, en diferentes niveles (L1, L2 y en algunos casos L3). Estas memoria *cache* son de alta velocidad aunque de menor capacidad que la memoria RAM del sistema. Su función es la de leer de forma anticipada memoria RAM mientras el CPU está trabajando y modificar la información leida de forma local. Cuando entra nueva información al cache la información ya procesada es almacenada en la memoria RAM. Otra de las ventajas del uso de *caches* es que son además más eficientes cuando leen o escriben localidades de memoria continuas [Drep07 p15].

El uso de caches como un método de acceso intermedio a la memora principal incrementa mucho la eficiencia de los procesadores mientras mantiene bajos los costos de la computadora. La siguiente tabla muestra los ciclos de reloj de CPU necesarios para accesar cada tipo de memoria en un procesador Pentium M:

Acceso a	Ciclos
Registro CPU	≤ 1
L1	~ 3
L2	~ 14
Memoria RAM	~ 240

Sin embargo, mantener coherencia en la información cuando varios procesadores accesan la misma memoria RAM es complejo, los detalles se pueden consultar en [Drep07]. Lo importante a notar es que para lograr buenos algoritmos en paralelo con memoria compartida es necesario que cada CPU trabaje en localidades de memoria diferentes, ya que si dos CPU modifican la misma dirección de memoria se crea una “falla” en la coherencia entre los *caches*, lo que obliga a los CPU a accesar a la memoria RAM, lo cual como vimos es muy costoso. Es muy importante tener en cuenta la arquitectura de *caches* de los sistemas *multi-core* al momento de diseñar algoritmos en paralelo cuyo requerimiento sea ser muy eficientes.

Otra de las desventajas del esquema de procesamiento en paralelo con memoria compartida es que existe un cuello de botella en el acceso a la memoria principal, ya que sólo un procesador puede accesarla la vez, esto será más y mas notorio cuando el sistema tenga más procesadores.

3.3. Paralelización con *threads*

Veamos ahora lo que significa en cuanto a la programación el procesamiento con *threads*. Un programa serial ejecuta sólo una secuencia de instrucciones, en cambio, un programa paralelizado puede contener diferentes secuencias de instrucciones que se ejecutan simutáneamente. A la ejecución de cada secuencia de instrucciones se le conoce como un hilo de procesamiento o *thread*. Cada *thread* posee sus propios registros de control y su propio *stack* de datos, mientas que comparte el uso de la memoria del montículo (*heap*) con los demás *threads* del programa. En las computadoras *multi-core* logra la mejor eficiencia cuando cada CPU ejecuta sólo un *thread*.

Comunmente el manejo de la programación con *threads* se hace a través de librerías de *software*, entre las más comunes están Windows Threads y POSIX Threads. En los últimos años se han incluido extensiones a los lenguajes de programación (C, C++ o Fortran) que simplifican la programación de las librerías de *threads*, como es el caso de OpenMP [Chap08]. Éste incorpora directivas para el compilador que indican de forma simple que partes del código deben paralelizarse, la declaración de memoria compartida o local, balance de carga, etc.

Un programa puede entonces crear varios *threads* para procesar un bloque del algoritmo, después ejecutar un bloque de forma serial para después regresar a procesar en paralelo, como se muestra en la figura 3.2. Por ejemplo, si se tiene que hacer una multiplicación de una matriz por vector en varias ocasiones. En este caso cada *thread* accesa cierto rango de renglones de la matriz, todos los *threads* accesan al vector por el cual se multiplica.

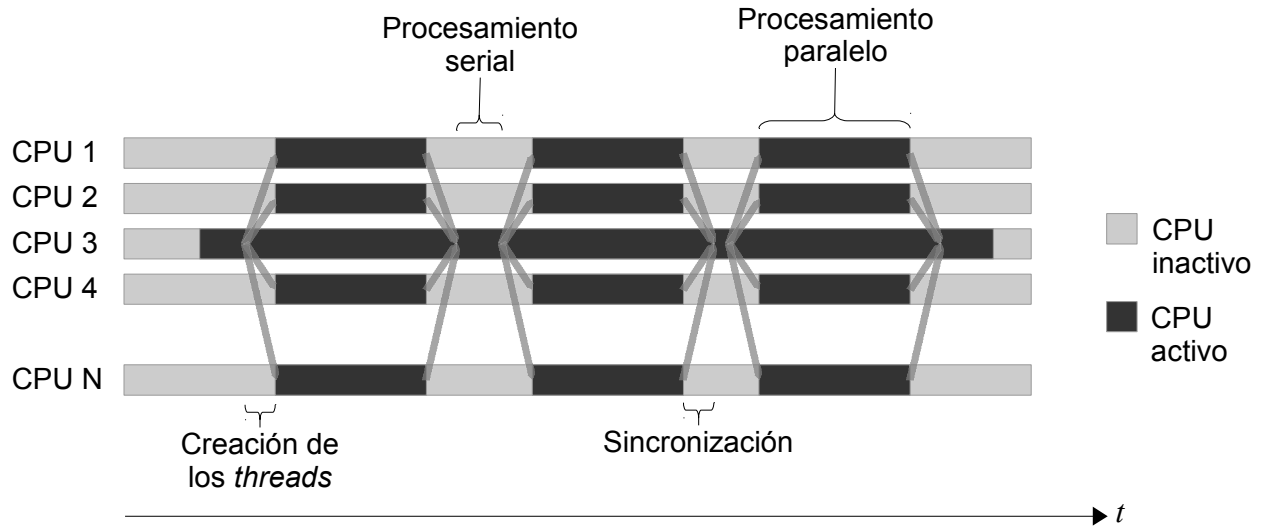


Figura 3.2. Visualización temporal de un proceso ejecutando threads en diferentes procesadores

Hay varias casos en los cuales se puede ver limitada la eficiencia del procesamiento con threads. Por ejemplo, la dependencia de datos, ésta se da cuando un *thread* tiene que esperar a que otro termine una

operación para poder continuar. Entre más dependencia exista entre los datos, más difícil será paralelizar un algoritmo. Por ejemplo, el algoritmo de sustitución hacia atrás para resolver sistemas de ecuaciones triangulares presenta mucha dependencia para encontrar la solución de cada incógnita.

Un caso más complejo sería cuando dos *threads* modifican los mismos datos, un *thread* podría entonces alterar un dato que otro *thread* espera permanezca constante. Se pierde entonces la sincronía en los algoritmos. Es necesario implementar sistemas de bloqueo de datos para poder modificar los datos de forma ordenada. Esto provoca que un thread tenga que parar y esperar a que otro termine para poder en su momento modificar o leer el valor.

Al repartir un trabajo entre varios *threads* puede suceder que la carga de trabajo no esté balanceada, provocando que uno o más procesadores terminen antes y estén inactivos, símplemente esperando. Entonces nuestro algoritmo será tan rápido como el más lento de los *threads*. Resultando en un desperdicio de poder de cómputo.

Después de ver las características de la programación con *threads* y memoria compartida, podemos buscar entonces un tipo de algoritmo que sea adecuado para funcionar en la paralelización de la solución de sistemas de ecuaciones. En particular, uno que se ajusta bastante bien a este esquema es el método de gradiente conjugado, el cual, como veremos tiene la ventaja de ser fácilmente paralelizable utilizando este esquema de procesamiento en parelelo con memoria compartida.

3.4. Algoritmo de gradiente conjugado

El algoritmo de gradiente conjugado es un método iterativo para minimizar funciones cuadráticas convexas de la forma

$$f(\mathbf{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{b}, \tag{3.1}$$

donde $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^n$ y $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ es una matriz simétrica positiva definida.

Para minimizar $f(\mathbf{x})$ calculamos primero el gradiente de (3.1),

$$\nabla f(\mathbf{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}.$$

Buscando igualar a cero el gradiente, podemos ver el gradiente conjugado como un método iterativo para resolver systemas de ecuaciones lineales

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

aplicando el concepto de vectores conjugados a una matriz

A partir de una matriz $\boldsymbol{A}$ simétrica positiva definida, podemos definir un producto interno como

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{y} = \langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\boldsymbol{A}}.$$

Ahora, decimos que un vector $\boldsymbol{x}$ es conjugado a otro vector $\boldsymbol{y}$ con respecto a una matriz $\boldsymbol{A}$ si

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\boldsymbol{A}} = 0, \text{ con } \boldsymbol{x} \neq \boldsymbol{y}.$$

La idea del algoritmo es utilizar direcciones conjugadas para el descenso en la búsqueda del punto óptimo $\boldsymbol{x}^*$ [Noce06 p103] , es decir

$$\boldsymbol{x}^* = \alpha_1 \boldsymbol{p}_1 + \alpha_2 \boldsymbol{p}_2 + \ldots + \alpha_n \boldsymbol{p}_n,$$

los coeficientes están dados a partir de la combinación lineal

$$\boldsymbol{A}\,\boldsymbol{x}^{*}=\alpha_1\boldsymbol{A}\,\boldsymbol{p}_1+\alpha_2\boldsymbol{A}\,\boldsymbol{p}_2+\ldots+\alpha_n\boldsymbol{A}\,\boldsymbol{p}_n=\boldsymbol{b},$$

con

$$\alpha_k=\frac{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{b}}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{A}\,\boldsymbol{p}_k}=\frac{\langle \boldsymbol{p}_k,\boldsymbol{b}\rangle}{\langle \boldsymbol{p}_k,\boldsymbol{p}_k\rangle_{\boldsymbol{A}}}.$$

A partir de una matriz $\boldsymbol{A}$ de rango n sólo se pueden definir n vectores $\boldsymbol{A}$-conjugados, por lo tanto el algoritmo de gradiente conjugado garantiza la obtención de una solución en un máximo de n iteraciones.

De la fórmula de actualización

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\alpha\,\boldsymbol{p}_k,$$

tomando $\boldsymbol{p}$ como una dirección de descenso.

Definamos

$$\boldsymbol{g}_k=\nabla f(\boldsymbol{x}_k),$$

el tamaño de paso α que minimiza la función $f(\boldsymbol{x})$ a lo largo de la dirección $\boldsymbol{x}_k+\alpha\,\boldsymbol{p}_k$ es

$$\alpha_k=-\frac{\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{p}_k}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{A}\,\boldsymbol{p}_k}.$$

Si definimos $\boldsymbol{p}_{k+1}$ como la dirección más cercana al gradiente $\boldsymbol{g}_k$ bajo la restricción de ser conjugado. Esta dirección está dada por la proyección de $\boldsymbol{g}_k$ en el espacio ortogonal a $\boldsymbol{p}_k$ con respecto al producto interno inducido por $\boldsymbol{A}$, así

$$\boldsymbol{p}_{k+1}=-\boldsymbol{g}_k+\frac{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{A}\,\boldsymbol{g}_k}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{A}\,\boldsymbol{p}_k}\boldsymbol{p}_k.$$

Al utilizar el negativo del gradiente tendremos una dirección de descenso.

sean
  $\boldsymbol{x}_0$, coordenada inicial
  $\boldsymbol{g}_0 \leftarrow \boldsymbol{A}\,\boldsymbol{x}_0-\boldsymbol{b}$, gradiente inicial
  $\boldsymbol{p}_0 \leftarrow -\boldsymbol{g}_0$, dirección inicial de descenso
  ε, tolerancia
  $k \leftarrow 0$
inicio
  mientras $\boldsymbol{g}_k \neq \boldsymbol{0}$, es decir $k<\operatorname{rank}(\boldsymbol{A})\wedge\|\boldsymbol{g}_k\|>\varepsilon$
    $\alpha_k \leftarrow -\dfrac{\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{p}_k}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{A}\,\boldsymbol{p}_k}$
    $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k+\alpha_k\,\boldsymbol{p}_k$
    $\boldsymbol{g}_{k+1} \leftarrow \boldsymbol{A}\,\boldsymbol{x}_{k+1}-\boldsymbol{b}$
    $\beta_k \leftarrow \dfrac{\boldsymbol{g}_{k+1}^{\mathrm{T}}\boldsymbol{A}\,\boldsymbol{p}_k}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{A}\,\boldsymbol{p}_k}$
    $\boldsymbol{p}_{k+1} \leftarrow -\boldsymbol{g}_{k+1}+\beta_{k+1}\boldsymbol{p}_k$
    $k \leftarrow k+1$
  fin_mientas
fin

Algoritmo 3.1. Gradiente conjugado.

Una fórma más económica del algoritmo 3.1, en la cual se reduce la cantidad de productos matriz-vector [Noce06 p112] es la mostrada en el algoritmo 3.2.

sean
$\boldsymbol{x}_0$, coordenada inicial
$\boldsymbol{g}_0 \leftarrow \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}$, gradiente inicial
$\boldsymbol{p}_0 \leftarrow -\boldsymbol{g}_0$, dirección inicial de descenso
ε, tolerancia
$k \leftarrow 0$
inicio
mientras $\boldsymbol{g}_k \neq \boldsymbol{0}$, es decir $k < \text{rank}(\boldsymbol{A}) \wedge \|\boldsymbol{g}_k\| > \varepsilon$
$\boldsymbol{w} \leftarrow \boldsymbol{A}\boldsymbol{p}_k$
$\alpha_k \leftarrow -\dfrac{\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{g}_k}{\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{w}}$
$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
$\boldsymbol{g}_{k+1} \leftarrow \boldsymbol{g}_k + \alpha \boldsymbol{w}$
$\beta_k \leftarrow \dfrac{\boldsymbol{g}_{k+1}^{\mathrm{T}}\boldsymbol{g}_{k+1}}{\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{g}_k}$
$\boldsymbol{p}_{k+1} \leftarrow -\boldsymbol{g}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$
$k \leftarrow k+1$
fin_mientas
fin

Algoritmo 3.2. Gradiente conjugado práctico.

3.5. Formulación en paralelo

Al realizar la paralelización del algoritmo es importante considerar que hay un costo en tiempo de procesamiento cada vez que se realiza una sincronización entre los *threads* utilizados, esta sincronización es manejada tanto por el sistema operativo como por la librería de manejo de *threads*. Si el algoritmo 3.2 lo implementamos calculando individualmente cada producto punto, suma de vectores, vector por escalar y matriz por vector, tendrémos más de una decena de puntos de sincronización.

La versión paralelizada del algoritmo anterior reduciendo los puntos de sincronización:

sean
  $\boldsymbol{x}_0$, coordenada inicial
  $\boldsymbol{g}_0 \leftarrow \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}$, gradiente inicial
  $\boldsymbol{p}_0 \leftarrow -\boldsymbol{g}_0$, dirección inicial de descenso
  ε, tolerancia
  $n \leftarrow$ Renglones$(\boldsymbol{x})$
  $k \leftarrow 0$
inicio
  mientras $\boldsymbol{g}_k \neq \boldsymbol{0}$, es decir $k < \text{rank}(\boldsymbol{A}) \wedge \|\boldsymbol{g}_k\| > \varepsilon$
    $q \leftarrow 0$, guardará $\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{w}$
    $g \leftarrow 0$, guardará $\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{g}_k$
    paralelizar para $i \leftarrow 1 \ldots n$
      $(\boldsymbol{w})_i \leftarrow 0$
      para $j \leftarrow 1 \ldots n$
        $(\boldsymbol{w})_i \leftarrow (\boldsymbol{w})_i + (\boldsymbol{A})_{ij}(\boldsymbol{p}_k)_j$
      fin_para
      $q \leftarrow q + (\boldsymbol{p}_k)_i(\boldsymbol{w})_i$
      $g \leftarrow g + (\boldsymbol{g}_k)_i(\boldsymbol{g}_k)_i$
    fin_para
    $\alpha_k \leftarrow -\dfrac{g}{q}$
    $h \leftarrow 0$, guardará $\boldsymbol{g}_{k+1}^{\mathrm{T}}\boldsymbol{g}_{k+1}$
    paralelizar para $i \leftarrow 1 \ldots n$
      $(\boldsymbol{x}_{k+1})_i \leftarrow (\boldsymbol{x}_k)_i + \alpha_k(\boldsymbol{p}_k)_i$
      $(\boldsymbol{g}_{k+1})_i \leftarrow (\boldsymbol{g}_k)_i + \alpha_k(\boldsymbol{w})_i$
      $h \leftarrow h + (\boldsymbol{g}_{k+1})_i(\boldsymbol{g}_{k+1})_i$
    fin_para
    $\beta_k \leftarrow \dfrac{h}{g}$
    paralelizar para $i \leftarrow 1 \ldots n$
      $(\boldsymbol{p}_{k+1})_i \leftarrow -(\boldsymbol{g}_{k+1})_i + \beta_{k+1}(\boldsymbol{p}_k)_i$
    fin_para
    $k \leftarrow k+1$
  fin_mientas
fin

Algoritmo 3.3. Gradiente conjugado en paralelo.

Se han agrupado las operaciones algebraicas en tres ciclos, quedando sólo dos puntos de sincronización, uno para calcular α_k y otro para β_k. Es posible reordenar el algoritmo para disminuir aún más los puntos de sincronización [DAze93] manteniendo la estabilidad numérica. Para nuestro programa elegimos mantener en su escencia el algoritmo 3.2, mostramos un extracto en el algoritmo 3.4.

```
Vector<T> G(rows); // Gradient
Vector<T> P(rows); // Descent direcction
Vector<T> W(rows); // A*P

omp_set_num_threads(threads);

T gg = 0.0;
#pragma omp parallel for default(shared) reduction(+:gg)
schedule(guided)
for (int i = 1; i <= rows; ++i)
{
    T sum = 0.0;
    int km = A.RowSize(i);
    for (register int k = 0; k < km; ++k)
    {
        sum += A.Entry(i, k)*X(A.Index(i, k));
    }
    G(i) = sum - Y(i); // G = AX - Y;
    P(i) = -G(i); // P = -G
    gg += G(i)*G(i); // gg = G'*G
}

T epsilon = tolerance*tolerance;
int step = 0;
while (step < max_steps)
{
    // Test termination condition
    if (gg <= epsilon) // Norm(Gn) <= tolerance
    {
        break;
    }

    T pw = 0.0;
    #pragma omp parallel for default(shared) reduction(+:pw)
schedule(guided)
    for (int i = 1; i <= rows; ++i)
    {
        T sum = 0.0;
        int km = A.RowSize(i);
        for (register int k = 0; k < km; ++k)
        {
            sum += A.Entry(i, k)*P(A.Index(i, k));
        }
        W(i) = sum; // W = AP
        pw += P(i)*W(i); // pw = P'*W
    }

    T alpha = gg/pw; // alpha = (G'*G)/(P'*W)

    T gngn = 0.0;
    #pragma omp parallel for default(shared) reduction(+:gngn)
    for (int i = 1; i <= rows; ++i)
    {
        X(i) += alpha*P(i); // Xn = X + alpha*P
        G(i) += alpha*W(i); // Gn = G + alpha*W
        gngn += G(i)*G(i); // gngn = Gn'*Gn
    }

    T beta = gngn/gg; // beta = (Gn'*Gn)/(G'*G)

    #pragma omp parallel for default(shared)
    for (int i = 1; i <= rows; ++i)
    {
        P(i) = beta*P(i) - G(i); // Pn = -G + beta*P
    }

    gg = gngn;
    ++step;
}
```

Algoritmo 3.4. Sección del código en C++ del algoritmo de gradiente conjugado en paralelo.

3.6. Implementación con matrices ralas

La parte más costosa del algoritmo 3.3 es la multiplicación matriz-vector, la cual tiene que realizarse una vez en cada iteración. Para ahorrar tanto memoria como tiempo de procesamiento sólo almacenaremos los elementos de la matriz $\boldsymbol{A}$ que sean distintos de cero. Lo cual es conveniente, dado que las matrices de rigidez resultantes de problemas de elemento finito son ralas, es decir, la mayor parte de las entradas de la matriz $\boldsymbol{A}$ son cero.

Hay varias estratégias de almacenamiento en memoria de matrices ralas, dependiendo de la forma en que se accesarán las entradas. El método *Compressed Row Storage* [Saad03 p362] es adecuado para el caso del algoritmo 3.3, en el cual se accesarán las entradas de cada renglón de la matriz $\boldsymbol{A}$ en secuencia.

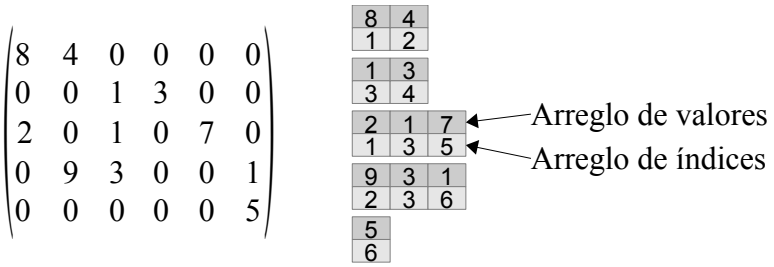


Figura 3.3. Almacenamiento con Compressed Row Storage.

Con este método, por cada renglón de la matriz se guardan dos arreglos. Uno conteniendo los índices y otro los valores de los elementos de ese renglón cuyo valor sea diferente a cero. Si buscamos en un renglón una entrada con cierto índice de columna, se tendrá un costo de búsqueda del elemento de orden $O(n)$ en el peor caso. Sin embargo, tenemos la ventaja que, para el caso de multiplicación matriz vector el órden de búsqueda es $O(1)$, esto es porque no se hace una búsqueda sino que se toman los elementos de cada renglón uno tras otro.

Otra de las ventajas de utilizar *Compressed Row Storage* es que los datos de cade renglón de la matriz de rigidez son accesados en secuencia uno tras otro, esto producirá una ventaja de acceso al entrar el bloque de memoria de cada renglón en el *cache* del CPU.

Para problemas de elemento finito, es posible conocer la estructura de la matriz de rigidez antes de llenarla, ya que son conocidas las conectividades de los nodos en el mallado. Si un nodo i está conectado con un nodo j entonces la matriz rala tendrá las entradas (i,j), (j,i), (i,i) y (j,j) distintas de cero. Conocer de antemano la estructura hará más eficiente la reserva de memoria de cada arreglo de la matriz rala. Para problemas con m grados de libertad por nodo las entradas de la matriz diferentes de cero serán $(i*m-k, j*m-l)$, $(j*m-k, i*m-l)$, $(i*m-k, i*m-k)$ y $(j*m-k, j*m-k)$, con $m, l=0\ldots2$.

3.7. Resultados

Los siguientes resultados se refieren a encontrar la deformación de un sólido tridimensional de 2'120,123 elementos y 381,674 nodos, figura 3.4. Con tres grados de libertad por nodo, tenemos entonces, un sistema de 1'145,022 ecuaciones.

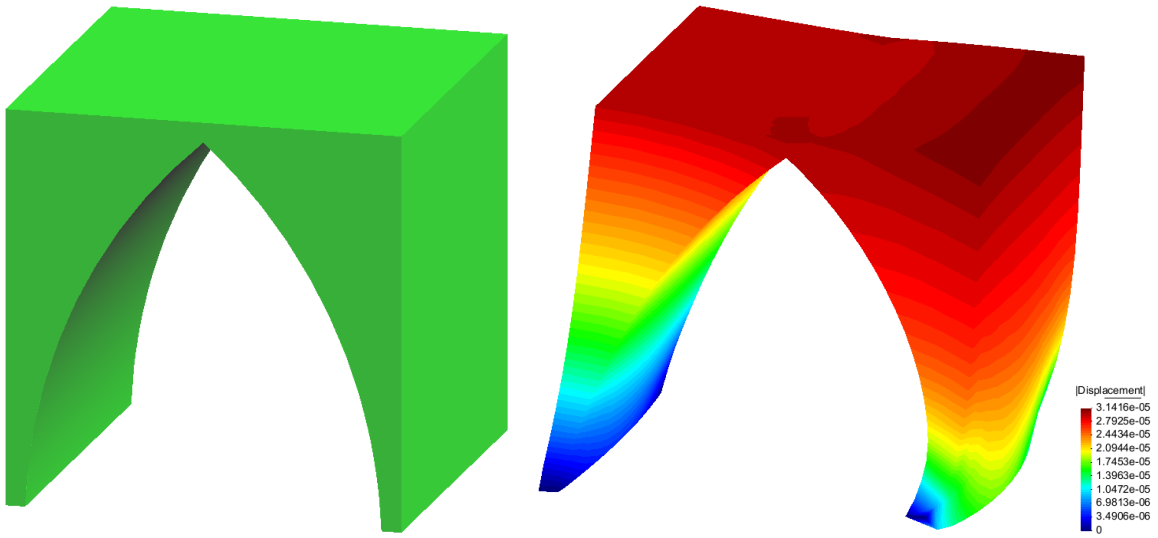
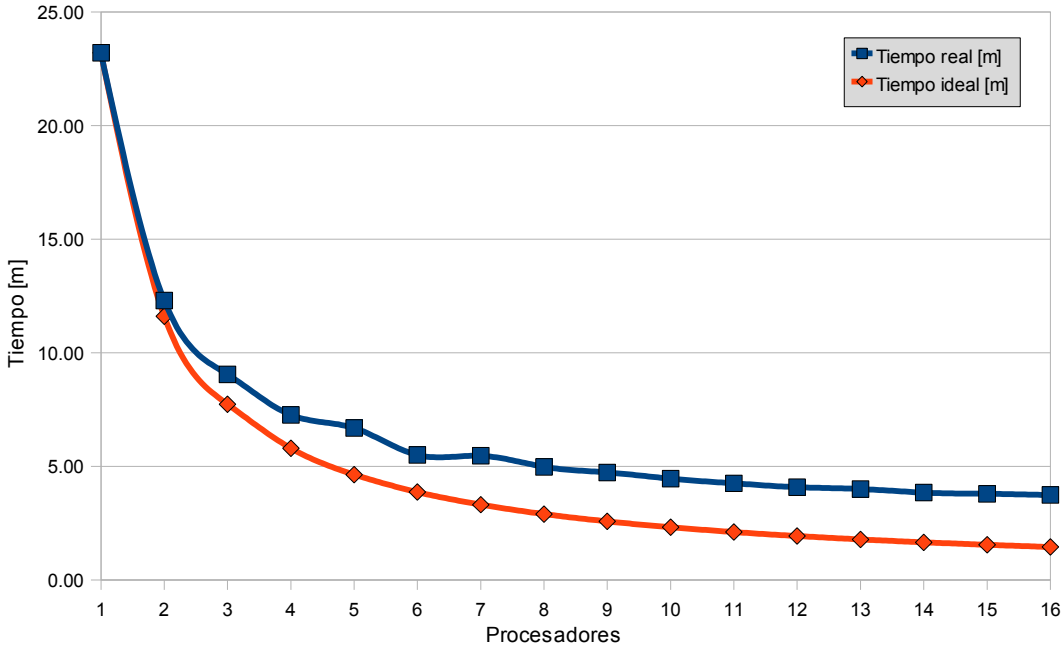


Figura 3.4. Resultado del problema de deformación de sólidos tridimensional.

Para estas pruebas se utilizó una computadora MacPro 4 con ocho procesadores Intel Xeon a 2.26GHz, con hyperthreading habilitado (es decir 16 CPUs). La gráfica 3.1 muestra los el tiempo que el programa tardó en resolver el problema paralelizando con una cantidad diferente de CPUs en cada una de las 16 pruebas. La primer prueba fue con un CPU, la segunda con dos, y así sucesivamente.

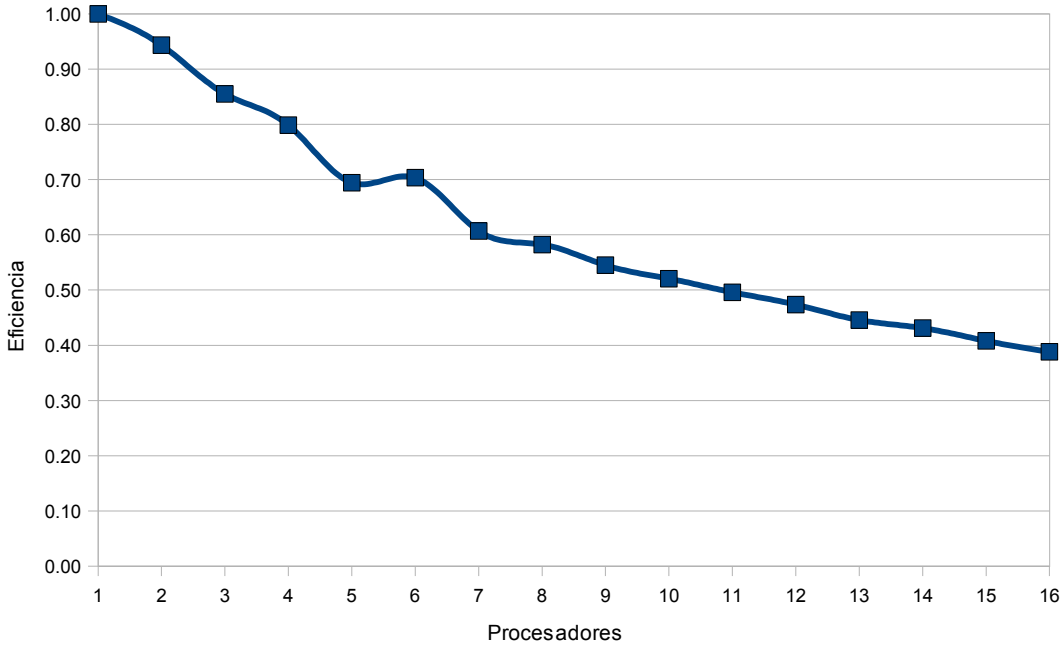


Gráfica 3.1. Comparación entre tiempo de ejecución real y el ideal esperado.

Sea t_1 el tiempo que tardó el resolverse el problema con un CPU, entonces, en la gráfica anterior el tiempo ideal es t_1/n, done n es el número de procesadores utilizado. Podemos dar una medida de eficiencia E del algoritmo

$$E = \frac{t_1}{n t_n},$$

con t_n el tiempo de ejecución que le tomó al programa en terminar el proceso.



Gráfica 3.2 Eficiencia de la paralelización del gradiente conjugado.

Es notorio en la gráfica 3.2 como la eficiencia de la paralelización disminuye conforme se aumenta el número de CPUs. Esto es debido a que se crea un cuello de botella cuando más de un CPU trata de accesar a la memoría RAM de forma simultánea, lo cual es inevitable para este tipo de arquitecturas. El que los procesadores utilicen las líneas de *cache* disminuye este efecto, pero el hecho de que se trate de problemas que utilizan una gran cantidad de memoria hara que la ayuda del *cache* sea sobrepasada. Una alternativa podría ser realizar este proceso utilizando GPUs, las cuales están diseñadas para paralelizar mejor el acceso a memoria.

Sin embargo, podemos decir que para un número reducido de CPUs (4 o menos) la paralelización del gradiente conjugado funciona bien ya que mantiene un rendimiento por arriba del 80%.

4. Una aplicación de la paralelización con memoria distribuída

4.1. Paralelización con memoria distribuida

Este esquema de paralelización implica que un programa ahora será ejecutado multiples veces en varias computadoras conectadas en red. A cada instancia del programa (proceso) se le asigna una parte del trabajo, los resultados son intercambiados entre los procesos a fin de colaborar para lograr un resultado global.

Para la comunicación entre los procesos utilizamos el estándar *Message Passing Interface* (MPI) [MPIF08], el cual consiste en un conjunto de librerías y programas que se encargan de facilitar la implementación y administración de programas que requieren transmitir información con gran eficiencia. Estos programas se ejecutarán simultáneamente en varias computadoras interconectadas por medio de una red de cómputo.

La figura 4.1 muestra un esquema de la arquitectura de procesamiento con memoria distribuída. En este modelo se tiene que cada computadora puede tener uno o más procesadores que localmente pueden trabajar bajo el esquema de memoria compartida o si se requiere bajo un esquema de memoria distribuída dentro de la misma computadora.

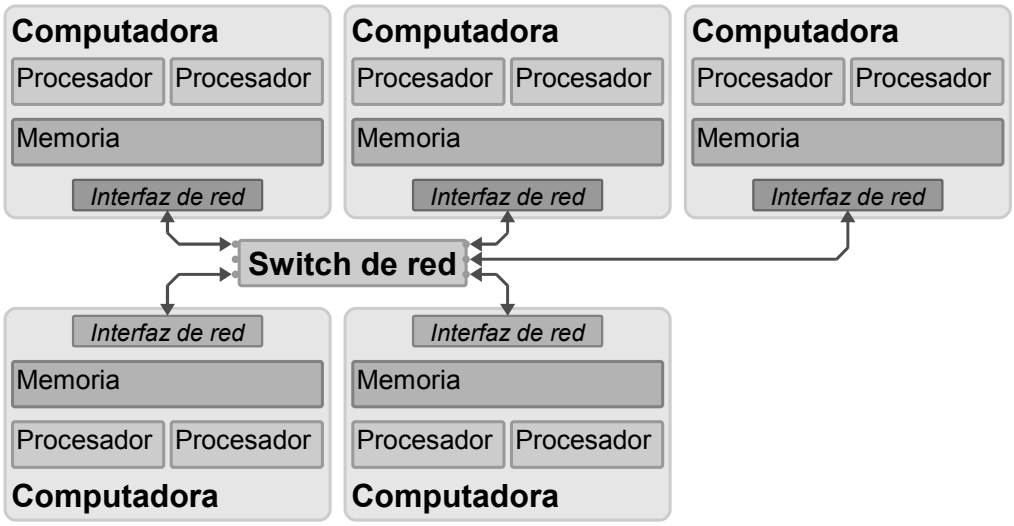


Figura 4.1. Esquema de procesamiento con memoria distribuida.

En su forma básica, el modelo MPI permite ejecutar un mismo programa en varias computadoras y en varios procesadores dentro de cada computadora. A cada instancia del programa (proceso) se le asigna un

número de rango que va de 0 a $N-1$, donde N es el número de instancias del programa. Visto de forma conceptual, la operación de MPI es una red de comunicación donde todos los procesos pueden mandar datos (o mensajes) a todos los procesos, ver figura 4.2.

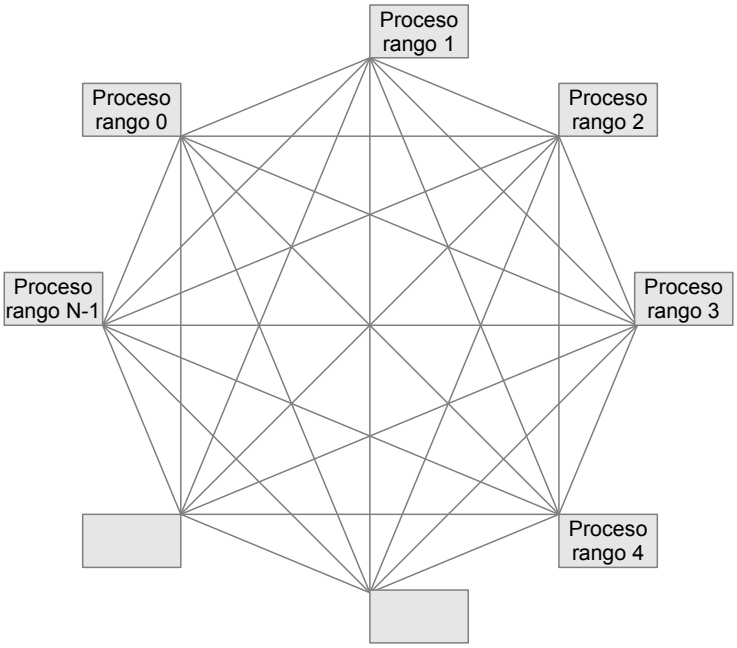


Figura 4.2. Diagrama conceptual de la comunicación con MPI.

Ningun proceso tiene prioridad sobre otro, aunque se suele dejar al proceso con rango cero el control de todos los demás. La comunicación entre los procesos es transparente para el usuario y se selecciona de forma automática, ésta puede ser establecida utilizando sockets TCP/IP en el caso de que los procesos estén en diferentes computadoras, o utilizando memoria compartida monitoreada con *polling* en el caso de que los procesos residan en la misma computadora.

4.2. Descomposición de dominios

Al discretizar sólidos en varios millones de elementos, se utiliza tal cantidad de información que hace que el cálculo de la solución requiera tiempos de procesamiento y/o cantidades de memoria tales que no es posible resolver el problema utilizando una sola computadora en un tiempo razonable. Es necesario dividir el dominio en particiones, para resolver cada una independientemente y después combinar las soluciones locales de forma iterativa.

Hay dos formas de trabajar la descomposición de dominio, con particiones con traslape y sin traslape. En este trabajo elegimos utilizar la versión en paralelo el método alternante de Schwarz [Smit96], que es un método con traslape entre las particiones. Un tratamiento profundo de la teoría de los algoritmos de Schwarz puede consultarse en [Tose05].

4.2.1. Algoritmo alternante de Schwarz

El algoritmo es conocido como el método alternante de Schwarz en paralelo. La figura 4.3 muestra un dominio $\boldsymbol{\Omega}$ con frontera $\partial\boldsymbol{\Omega}$, el cual va a ser dividido en dos particiones $\boldsymbol{\Omega}_1$ y $\boldsymbol{\Omega}_2$.

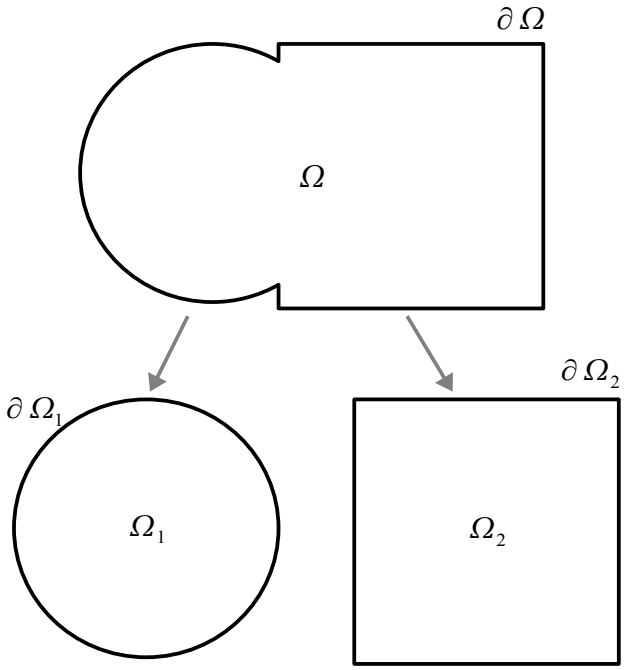


Figura 4.3. Descomposición de un dominio en dos particiones.

La figura 4.4 muestra las particiones traslapadas, en las cuales $\boldsymbol{\Omega}=\boldsymbol{\Omega}_1\cup\boldsymbol{\Omega}_2$.

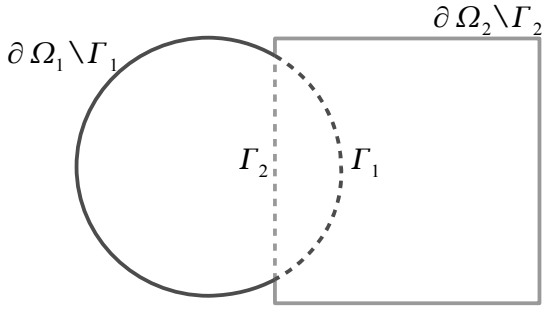


Figura 4.4. Particiones traslapadas.

Las fronteras Γ_1 y Γ_2 son fronteras artificiales y son la parte de las fronteras de $\boldsymbol{\Omega}_1$ y $\boldsymbol{\Omega}_2$ que están en el interior de $\boldsymbol{\Omega}$. Para resolver una ecuación diferencial, siendo $\boldsymbol{L}$ un operador diferencial, tenemos

$$\boldsymbol{L}\,\boldsymbol{x}=\boldsymbol{y} \text{ en } \boldsymbol{\Omega},$$

$$\boldsymbol{x}=\boldsymbol{b} \text{ sobre } \partial\boldsymbol{\Omega}.$$

El método alternante de Schwarz consiste en resolver de cada partición de forma independiente, fijando condiciones de Dirichlet en las fronteras artificiales de cada partición con los valores de la iteración previa de la partición adyacente.

$\boldsymbol{x}_1^0$, $\boldsymbol{x}_2^0$ aproximaciónes iniciales
ε tolerancia
i número de iteración
mientras $\left\|\boldsymbol{x}_1^i - \boldsymbol{x}_1^{i-1}\right\| > \varepsilon$ y $\left\|\boldsymbol{x}_2^i - \boldsymbol{x}_2^{i-1}\right\| > \varepsilon$

resolver		resolver	
$\boldsymbol{L}\,\boldsymbol{x}_1^i = \boldsymbol{y}$	en $\boldsymbol{\Omega}_1$	$\boldsymbol{L}\,\boldsymbol{x}_2^i = \boldsymbol{y}$	en $\boldsymbol{\Omega}_2$
$\boldsymbol{x}_1^i = \boldsymbol{b}$	en $\partial\boldsymbol{\Omega}_1 \setminus \boldsymbol{\Gamma}_1$	$\boldsymbol{x}_2^i = \boldsymbol{b}$	en $\partial\boldsymbol{\Omega}_2 \setminus \boldsymbol{\Gamma}_2$
$\boldsymbol{x}_1^i \leftarrow \boldsymbol{x}_2^{i-1}\big\vert_{\boldsymbol{\Gamma}_1}$	en $\boldsymbol{\Gamma}_1$	$\boldsymbol{x}_2^i \leftarrow \boldsymbol{x}_1^{i-1}\big\vert_{\boldsymbol{\Gamma}_2}$	en $\boldsymbol{\Gamma}_2$

$i \leftarrow i+1$
fin_mientras

Algoritmo 4.1. Método alternante de Schwarz.

El algoritmo deberá iterar hasta que se satisfaga las condiciones de tolerancia.

4.2.2. Aplicación con un problema de elemento finito

Cuando el operador $\boldsymbol{L}$ tiene una representación como matriz, el algoritmo 4.1 corresponde a una generalización (debido al traslape) del método iterativo tradicional Gauss-Seidel por bloques [Smit96 p13].

En problemas de elemento finito, el traslape se realiza aumentando a cada partición elementos de la partición adyacente a partir de la frontera entre las particiones. La figura 4.5 muestra un ejemplo simple.

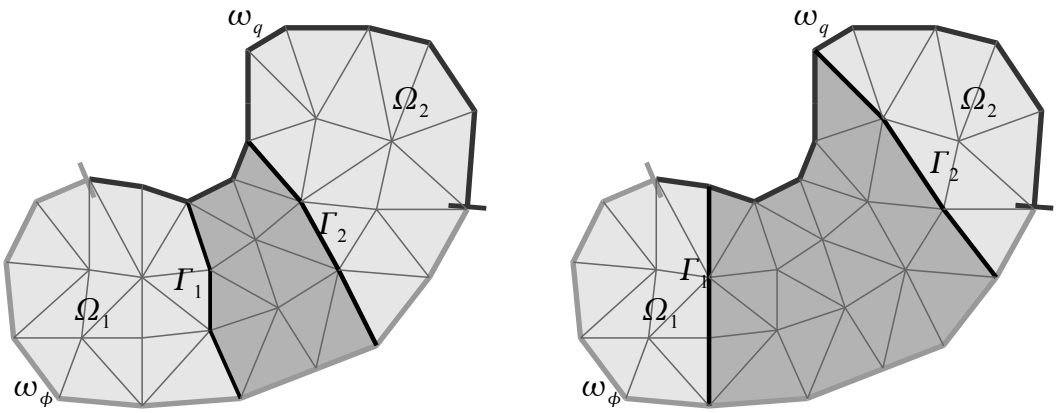


Figura 4.5. Descomposición con traslape de un dominio $\boldsymbol{\Omega}$ con diferentes capas de traslape.

Las fronteras virtuales se formarán con los nodos que están en la parte más exterior del traslape. El intercambio de valores será entonces con los resultados encontrados en unos nodos "fantasma" que se encontrarán en la partición adyacente.

4.2.3. Velocidad de convergencia

La velocidad de convergencia al utilizar descomposición de dominios se deteriora conforme se aumenta el número de particiones [Smith96 p53]. Esto puede verse de forma heurística como sigue. Considerese el dominio $\boldsymbol{\Omega}$ de la figura 4.6, el cual está dividido en N particiones.

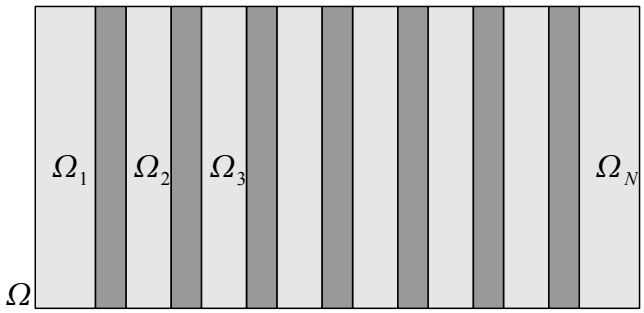


Figura 4.6. Dominio dividido en N particiones.

En cada iteración del metodo alternante de Schwarz solo transferirá información entre las particiones adyacentes. Entonces, si se tiene una condición de frontera diferente de cero en la primer partición, y se inicia en la iteración 0, le tomará N iteraciones para que la solución local en la partición N sea diferente de cero. Por tanto, el algoritmo alternante de Schwarz impone límites en la velocidad en la cual la información es transferida globalmente a través de todo el dominio.

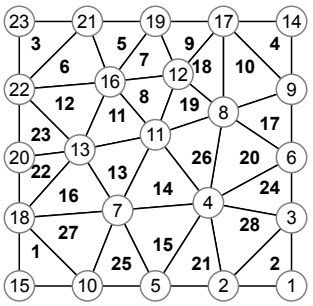
El algoritmo de Schwarz típicamente converge a una velocidad que es independiente (o ligeramente dependiente) de la densidad de la malla y de la partición, cuando el traslape entre las particiones es suficientemente grande [Smith96 p74].

4.3. Particionamiento del dominio

Nuestra idea es entonces particionar el dominio para resolver cada partición en una computadora del cluster, cada partición será tratada como un problema individual de tamaño reducido. Localmente cada problema individual será resuelto utilizando el método de gradiente conjugado paralelizado con memoria compartida visto en el capítulo 3. Utilizaremos el esquema MPI de memoria distribuída para intercambiar el resultado en las fronteras artificiales entre las particiones, esto se hara iterativamente siguiendo el método alternante de Schwartz. Tendremos así un sistema híbrido que combina los esquemas de procesamiento con memoria compartida y distribuída.

Vamos a mostrar como se realiza la partición del dominio con traslape necesario para implementar el método alternante de Schwarz, utilizando para ello un ejemplo sencillo de un problema de elemento finito en dos dimensiones con una malla de 28 elementos triangulares y 23 nodos. El dominio se dividirá en dos particiones con una capa de traslape.

La malla es generada por el módulo de pre-procesamiento del programa GiD, el cual permite generar la geometría y entrega una tabla de conectividades, la tabla indica los nodos n_1, n_2, n_3 que pertenecen a cada elemento E, figura 4.7. Denotaremos con negritas los números que corresponden a elementos y sin negritas los que corresponen a nodos.

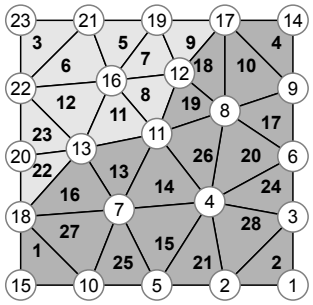


E	n	n	n
1	15	10	18
2	1	3	2
3	23	22	21
4	14	17	9
5	19	21	16
6	16	21	22
7	19	16	12
8	12	16	11
9	19	12	17
10	17	8	9
11	11	16	13
12	13	16	22
13	11	13	7
14	11	7	4

E	n	n	n
15	4	7	5
16	7	13	18
17	6	9	8
18	8	17	12
19	8	12	11
20	6	8	4
21	5	2	4
22	20	18	13
23	22	20	13
24	3	6	4
25	10	5	7
26	11	4	8
27	18	10	7
28	2	3	4

Figura 4.7. Mallado y su representación con conectividades.

Para realizar la partición inicial del dominio nuestro programa utiliza la librería de software METIS [Kary99], ésta recibe como entrada la tabla de conectividades y un número que indica las particiones requeridas. La figura 4.8 muestra el resultado entregado por esta librería, indicaremos los elementos de la primer partición como EP_1 y los de la segunda como EP_2. La librería indica a que partición pertenece cada nodo, solo divide los elementos.

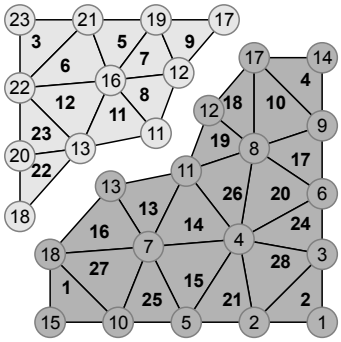


EP_1
3
5
6
7
8
9
11
12
22
23

EP_2	
1	**18**
2	**19**
4	**20**
10	**21**
13	**24**
14	**25**
15	**26**
16	**27**
17	**28**

Figura 4.8. Separación de los elementos del mallado en dos particiones.

Buscando en la tabla de conectividades determinamos que nodos pertenecen a cada elemento en cada partición. Indicaremos los nodos de la primer partición como nP_1 y los de la segunda como nP_2. Los nodos pueden pertenecer a más de una partición, como se observa en la figura 4.9.



nP_1
11
12
13
16
17
18
19
20
21
22
23

nP_2	
1	10
2	11
3	12
4	13
5	14
6	15
7	17
8	18
9	

Figura 4.9. Identificación de los nodos de cada partición.

Para definir qué nodos pertencen a la frontera entre las particiones, buscamos los nodos que aparezcan en más de una partición. En el caso de la figura 4.10, que aparezcan tanto en nP_1 como en nP_2.

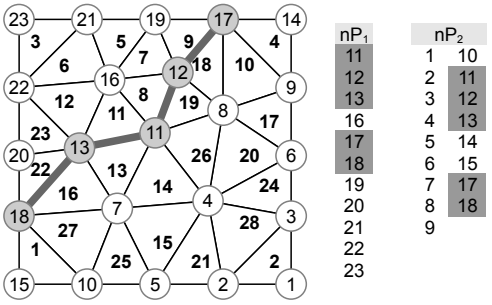


Figura 4.10. Definición de los nodos frontera entre las particiones.

A partir de este momento dividiremos la malla en el número de particiones, la frontera definida anteriormente seguirá existiendo en cada sub-malla tendrá una frontera, tal como se muestra en la figura 4.11, existen dos fronteras FP_1 y FP_2, en esta etapa ambas son iguales.

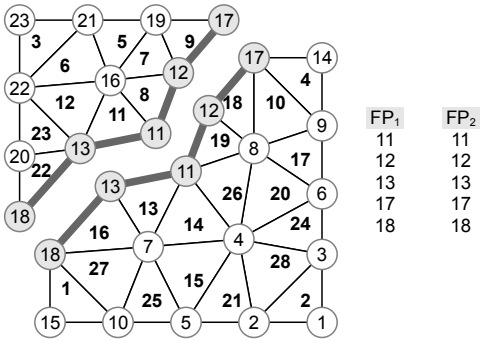


Figura 4.11. División de la malla en dos sub-mallas con dos fronteras.

Aumentar una capa de traslape significa que se agregarán a la sub-partición los elementos que compartan nodo en la frontera, pero que no estén en dicha sub-partición. Los nodos de estos nuevos elementos que no estén en la sub-partición se agregarán y formarán la nueva frontera, suplantando a la anterior. En la figura 4.12 se observa cómo se agrega una capa de traslape a cada partición, se denotan los elementos agregados y la nueva frontera formada.

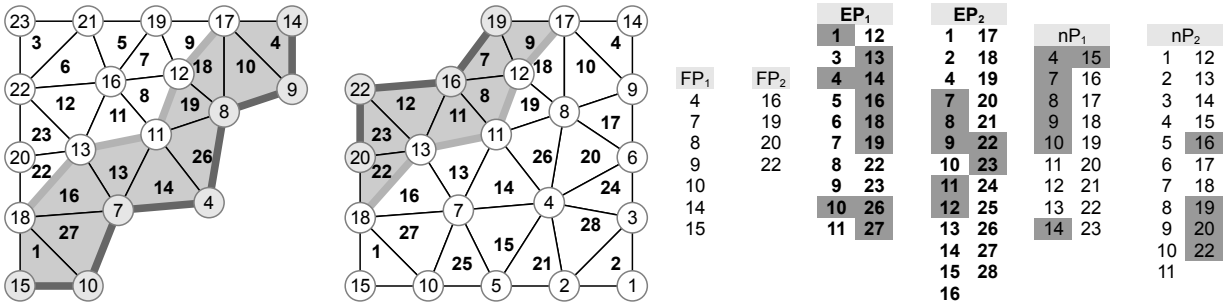
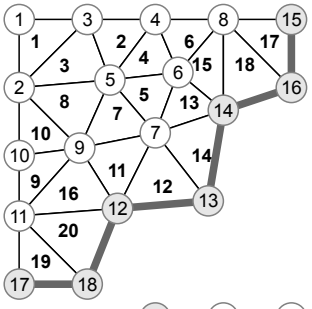


Figura 4.12. Aumento de una capa de traslape en cada partición.

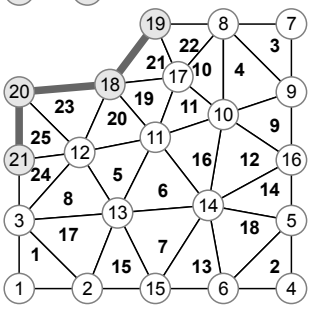
Este proceso se puede repetir tantas veces como capas de traslape se deseen agregar a cada partición. La frontera final será la frontera artificial del método alternante de Schwarz, en la cual se impondrán condiciones de Dirichlet antes de solucionar el sistema local de ecuaciones de la partición.

Ahora es necesaria la renumeración local a cada partición de elementos y nodos. Primero se renumeran los elementos y en base a estos los nodos. Este paso es necesario ya que cada sub-malla se procesará por separado como si se tratase de un problema independiente. La figura 4.13 muestra el reordenamiento para nuestro ejemplo, las tablas indican la numeración local y global, tanto de elementos como de nodos.



EP_1	E	EP_1	E
1	3	11	13
2	5	12	14
3	6	13	19
4	7	14	26
5	8	15	18
6	9	16	16
7	11	17	4
8	12	18	10
9	22	19	1
10	23	20	27

nP_1	n	nP_1	n
1	23	10	20
2	22	11	18
3	21	12	7
4	19	13	4
5	16	14	8
6	12	15	14
7	11	16	9
8	17	17	15
9	13	18	10



EP_2	E	EP_2	E
1	1	14	24
2	2	15	25
3	4	16	26
4	10	17	27
5	13	18	28
6	14	19	8
7	15	20	11
8	16	21	7
9	17	22	9
10	18	23	12
11	19	24	22
12	20	25	23
13	21		

nP_2	n	nP_2	n
1	15	12	13
2	10	13	7
3	18	14	4
4	1	15	5
5	3	16	6
6	2	17	12
7	14	18	16
8	17	19	19
9	9	20	22
10	8	21	20
11	11		

Figura 4.13. Reenumeración de elementos y nodos.

Esta numeración funciona bien para solvers iterativos (por ejemplo con gradiente conjugado), pero es inadecuada para *solvers* directos como la factorización Cholesky. Cuando se aplica la factorización Cholesky a una matriz, la cantidad de nodos no cero en la matriz factor $\boldsymbol{L}$ depende del órden de los nodos en la malla o lo que es equivalente, el órden de los renglones y columnas en la matriz de rigidez. Cuando se utiliza el solver con factorización Cholesky será necesario agregar un paso extra que imponga una enumeración de los nodos más adecuada, para una descripción más extensa de este caso consultar el capítulo 5.

El paso final es crear enlaces entre los nodos en la frontera artificial de cada sub-malla con su correspondiente nodo fantasma en la otra sub-malla. Estos enalces serán utilizados para intercambiar información entre las particiones. Se tendrán una lista de nodos fantasma para cada partición. Cada enlace guarda dos datos, el índice del nodo en la frontera artificial con la numeración local y el índice del nodo fantasma con la numeración correspondiente en la partición adyacente. En la figura 4.14 se muestran los enlaces formados entre los nodos en la frontera artificial y los nodos fantasma en la otra partición.

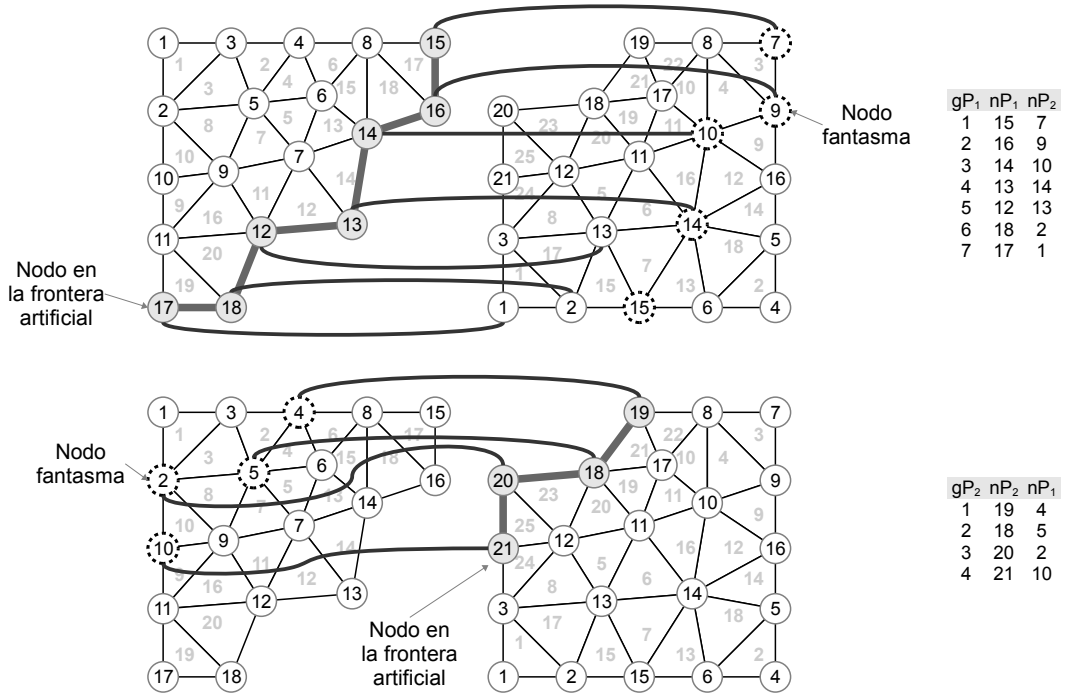


Figura 4.14. Creación de los enlaces hacia los nodos fantasma

Para realizar particionamientos con más de dos particiones se requerirá guardar una lista de enlaces por cada partición con la que se tenga una frontera artificial.

El ejercicio de realizar el particionamiento del dominio en tres dimensiones es más complicado de visualizar, pero se siguen exactamente los mismos pasos, la única diferencia es la cantidad de nodos por elemento. La figura 4.15 muestra el ejemplo de un dominio en tres dimensiones dividido en 16 particiones con traslape.

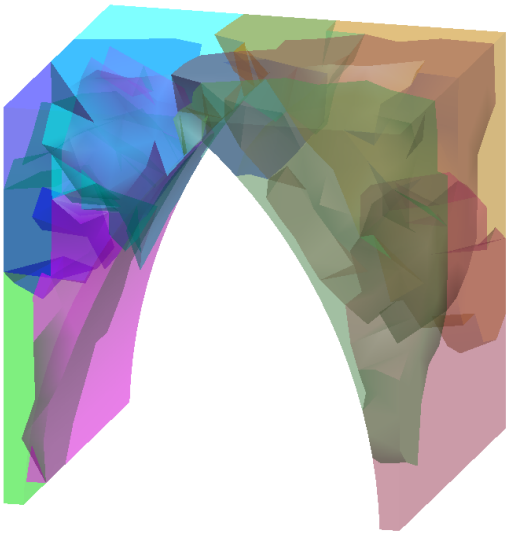


Figura 4.15. Ejemplo tridimensional de particionamiento con traslape.

4.4. Implementación con MPI

Vamos a explicar de forma simplificada las rutinas de MPI [MPIF08] utilizadas en nuestra implementación del método alternante de Schwarz. Vamos a mostrar el funcionamiento utilizando la geometría de un arco bidimensional al cual se le ha impuesto un desplazamiento en la parte superior derecha. Por brevedad no vamos a presentar la sintaxis completa de las rutinas de MPI. Las rutinas de MPI de envio y recepción de datos tienen la siguiente estructura

$$\text{MPI\_Function}(i, D),$$

donde i es el nodo al cual se envia o se quiere leer el dato o mensaje D. Las funciones que cuyo nombre comienza con “I” indican que no se espera a que los datos estén listos para enviar o recibir,

$$\text{MPI\_Ifunction}(i, D)$$

esto permite continuar con la ejecución del programa. La sincronización de estos datos se dará cuando se llame a la función

$$\text{MPI\_Waitall}(D).$$

1. El programa inicia con una geometría, una tabla de conectividades y ciertas condiciones de frontera, y un número P que indica la cantidad de particiones. Para el ejemplo de figura 4.16 el programa se instancia 5 veces, asignándose el proceso con rango 0 como el maestro y los procesos 1 a 4 como esclavos, es decir, tendremos 4 particiones. Cada nodo entra en el esquema de MPI llamando la rutina MPI_Init.

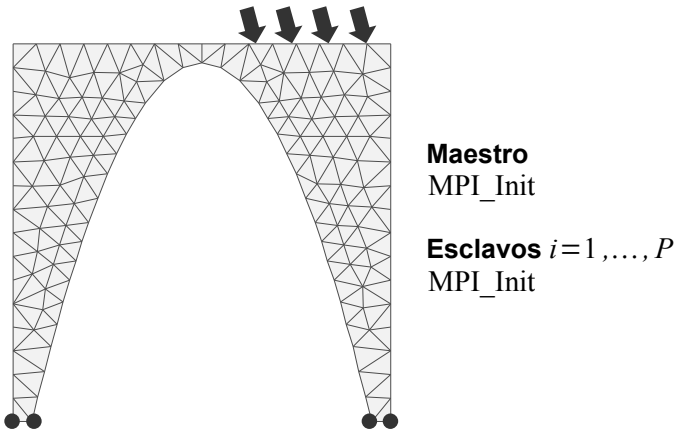


Figura 4.16. Entrada del programa.

2. En el nodo maestro se realiza un particionamiento con traslape siguiendo el procedimiento descrito en la sección anterior. Se generan las particiones con traslape, se reenumeran elementos y nodos, finalmente se generan tablas con los enlaces de los nodos fantasma, figura 4.17.

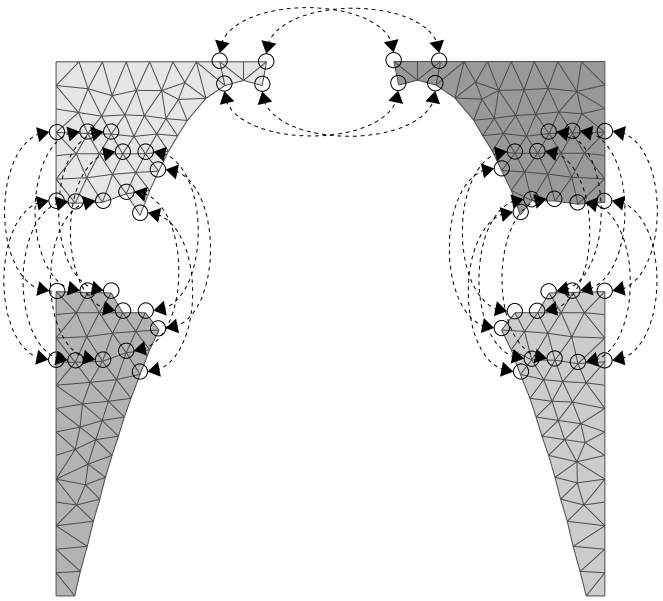


Figura 4.17. Particionamiento del dominio con traslape.

3. Ya con las particiones independientes se generan los problemas individuales, cada uno tiene sus propias condiciones de frontera. El nodo maestro entra en un ciclo $i=1,\ldots,P$ en el cual se envian los siguientes datos correspondientes a cada partición i utilizando la función MPI_Send: $\boldsymbol{N}^i$ coordenadas de los nodos, $\boldsymbol{E}^i$ tabla de conectividades de los elementos, $\boldsymbol{C}^i$ condiciones de frontera; se envian en un ciclo $j=1,\ldots,P$ con $j\neq i$ los $\boldsymbol{L}^{i\,j}$ enlaces con los indices de los nodos de frontera artificial y su correspondiente nodo fantasma en la partición j. Con la función MPI_Irecv$\left(i,s^i\right)$ se crea una petición sin bloqueo del estatus del nodo i, esto le permite al nodo maestro seguir enviando datos sin esperar a que los esclavos estén listos. Por su parte cada esclavo utilizará la función MPI_Recv para recibir todos estos datos. Cada esclavo generará su sistema de ecuaciones y cuando esté listo enviará su estatus s^i al nodo maestro. Todo esto es indicado en la figura 4.18.

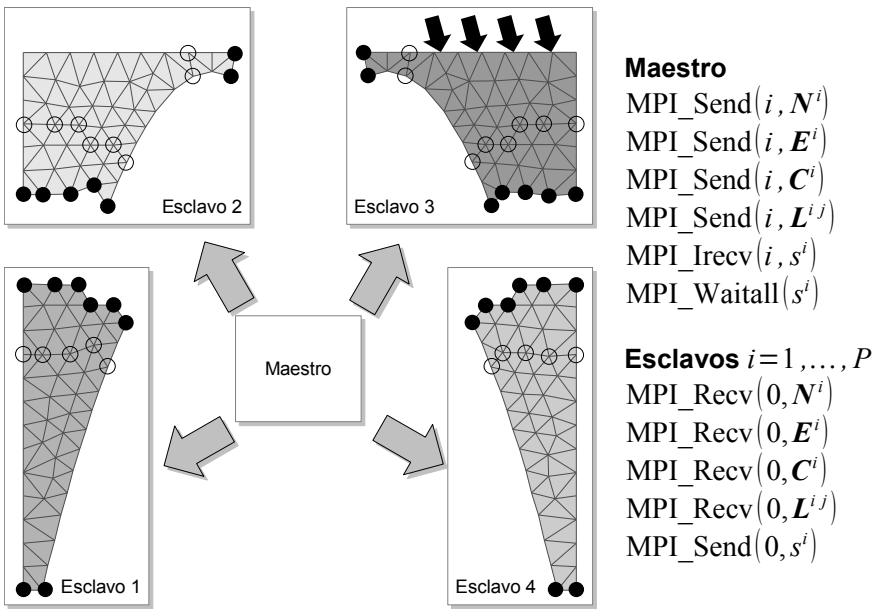


Figura 4.18. Creación de problemas individuales.

Con la función MPI_Waitall$\left(s^i\right)$ el nodo maestro espera el estatus de todos los esclavos antes de continuar.

4. En este punto el programa comienza a iterar hasta que se logre la convergencia del algoritmo 4.1. En cada esclavo el sistema de ecuaciones es resuelto localmente. Al terminar cada esclavo envia la diferencia entre la solución anterior y la actual d^i al nodo maestro, el cual evalua la convergencia global del problema. En caso de que el problema llegue a una convergencia se continuará al paso 6. Ver figura 4.19.

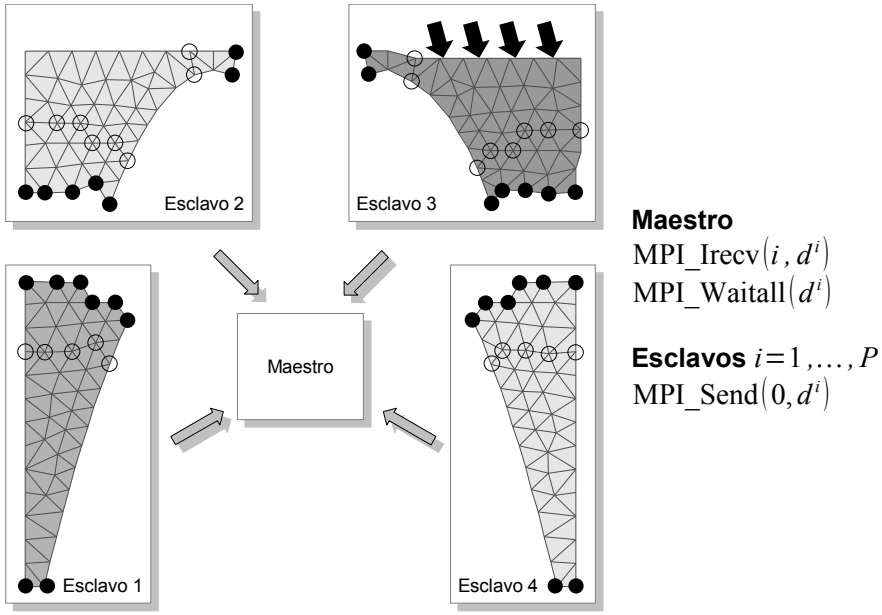


Figura 4.19. Reporte del estado de la convergencia al nodo maestro.

5. Si aún no se ha logrado la convergencia global, entonces los esclavos solicitarán los valores en los nodos "fantasma" a cada una de las particiones adyacentes con la función MPI_Irecv$(j, \boldsymbol{G}^{ij})$, con $j=1,\ldots,P$, $i \neq j$, al mismo tiempo enviara los valores de los nodos fantasma que soliciten las otras particiones con el la función MPI_Isend$(j, \boldsymbol{G}^{ji})$, $j=1,\ldots,P$, $i \neq j$. Con la función MPI_Wait esperaran a que todas las transacciones con las particiones adyacentes hayan concluido. Nótese en la figura 4.20 que el nodo maestro no interviene en este paso.

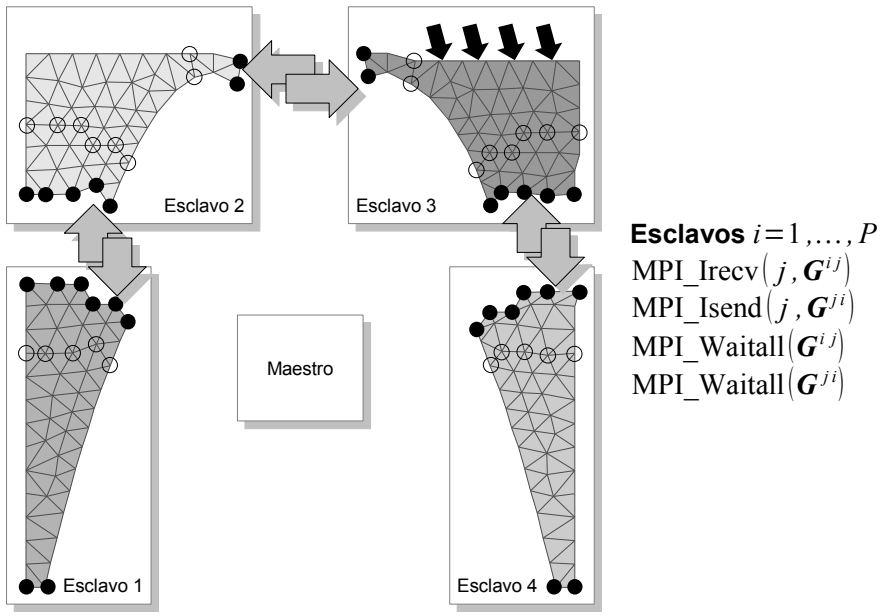


Figura 4.20. Generación de la soluciones locales e intercambio de valores en las fronteras artificiales con sus respectivos nodos fantasma.

Se regresa al paso 4 para continuar a la siguiente iteración.

6. Una vez que se logre la convergencia el nodo maestro entrará en un ciclo $i=1,\ldots,P$ y solicitará uno a uno a los esclavos los resultados de desplazamiento $\boldsymbol{u}^i$, deformación $\boldsymbol{\varepsilon}^i$, y esfuerzos $\boldsymbol{\sigma}^i$ de cada partición. Esto se indica en la figura 4.21.

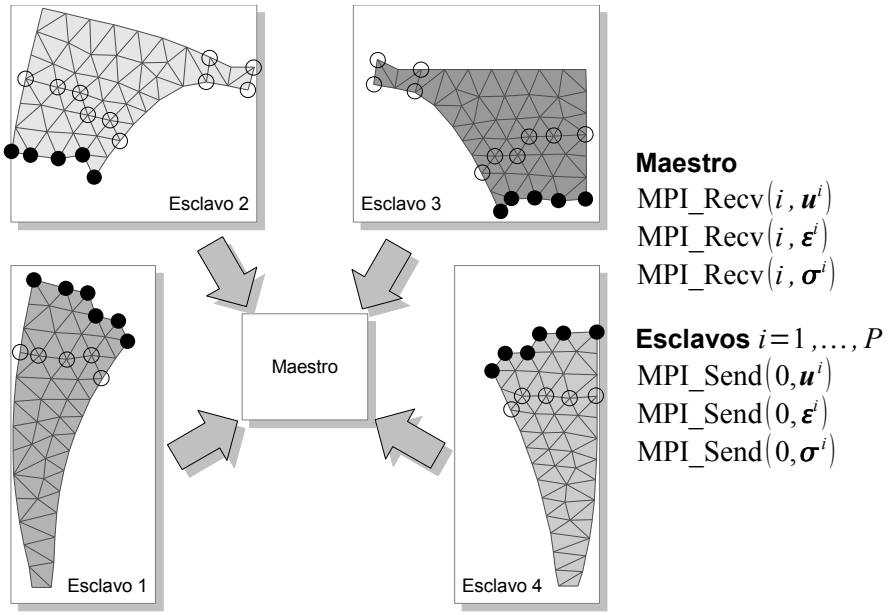


Figura 4.21. Envio de resultados locales al nodo maestro.

7. El nodo maestro generará una solución global conjuntando los resultados de todas las particiones, figura 4.22. Los nodos salen del esquema MPI llamando la función MPI_Finalize.

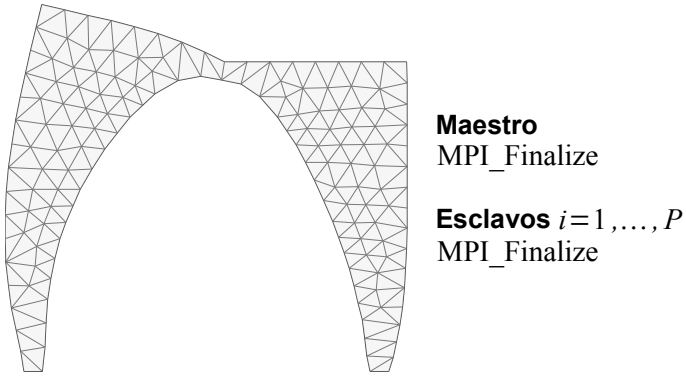


Figura 4.22. Consolidación de un resultado global.

5. Factorización Cholesky simbólica para matrices ralas

5.1. Cómo lograr un solver directo eficiente

El algoritmo factorización Cholesky para resolver sistemas de ecuaciones es computacionalmente mas caro que el método de gradiente conjugado, tanto en tiempo como en utilización de memoria. Sin embargo, al trabajar con el algoritmo alternante de Schwarz obtendremos la ventaja, de que una vez factorizado el sistema de ecuaciones, resolver el sistema en cada iteración de Schwarz consistirá simplemente en hacer una sustitución hacia adelante y una hacia atrás, lo cual es bastante rápido.

La tabla 5.1 muestra los diferentes costos de operación y almacenamiento para matrices simétricas [Piss84 p63], siendo n el rango de la matriz, b el ancho de banda y r_i el número de elementos no cero de la columna i de $\boldsymbol{L}$.

Matriz	Multiplicaciones y divisiones	Sumas	Almacenamiento
Simétrica completa	$\frac{1}{6}n^3+\frac{1}{2}n^2-\frac{2}{3}n$	$\frac{1}{6}n^3-\frac{1}{6}n$	$\frac{1}{2}n^2+\frac{1}{2}n$
Simétrica bandada	$\frac{1}{2}b(b+3)n-\frac{1}{3}b^3-b^2-\frac{2}{3}b$	$\frac{1}{2}b(b+1)n-\frac{1}{3}b^3-\frac{1}{2}b^2-\frac{1}{6}b$	$(b+1)n-\frac{1}{2}b^2-\frac{1}{2}b$
Simétrica rala	$\sum_{i=1}^{n-1} r_i(r_i+3)/2$	$\sum_{i=1}^{n-1} r_i(r_i+1)/2$	$n+\sum_{i=1}^{n-1} r_i$

Tabla 5.1. Costo de operación y almacenamiento para diferentes tipos de matrices.

Para el caso del algoritmo de sustitución hacia atrás, los costos operación se muestran en la tabla 5.2 [Piss84 p64].

Matriz	Multiplicaciones y divisiones	Sumas
Simétrica completa	n^2	n^2-n
Simétrica bandada	$(2b+1)n-b^2-b$	$2b\,n-b^2-b$
Simétrica rala	$n+2\sum_{i=1}^{n-1} r_i$	$2\sum_{i=1}^{n-1} r_i$

Tabla 5.2. Costo de operación del algoritmo de sustitución hacia atrás.

Debido a la dependencia entre las variables, no es sencillo paralelizar la factorización Cholesky [Heat91], sin embargo, podemos aumentar el número de particiones para así obtener sistemas individuales de menor tamaño, lo que resultará a su vez en una factorización más rápida.

Utilizaremos dos estrategias para disminuir el uso de tiempo y memoria de la factorización Cholesky. La primer estrategia es reordenar los renglones y columnas de la matriz del sistema de ecuaciones para reducir el tamaño de las matrices resultantes de la factorización. La segunda estrategia es utilizar la

factorización Cholesky simbólica para obtener la factorización exacta y formar con esta matrices ralas que no tengan elementos cero. La combinación de ambas estrategias reducirá tanto el tiempo de ejecución como la memoria utilizada.

5.2. Factorización clásica de Cholesky

Como antecedente vamos a describir la factorización Cholesky tradicional [Quar00 p80].

Un sistema de ecuaciones

$$\boldsymbol{A}\,\boldsymbol{x}=\boldsymbol{y}, \tag{5.1}$$

con una matriz $\boldsymbol{A}\in\mathbb{R}^{n\times n}$ simétrica positiva definida puede ser resuelto aplicando a esta matriz la factorización Cholesky

$$\boldsymbol{A}=\boldsymbol{L}\,\boldsymbol{L}^{\mathrm{T}}, \tag{5.2}$$

donde $\boldsymbol{L}$ es una matriz triangular inferior. Ésta factorización existe y es única [Quar00 p80].

Las fórmulas para determinar los valores de $\boldsymbol{L}$ son

$$L_{ij}=\frac{1}{L_{jj}}\left(A_{ij}-\sum_{k=1}^{i-1}L_{ik}L_{jk}\right),\ \text{para } i>j \tag{5.3}$$

$$L_{ii}=\sqrt{A_{ii}-\sum_{k=1}^{i-1}L_{ik}^{2}}. \tag{5.4}$$

Sustituyendo (5.2) en (5.1), tenemos

$$\boldsymbol{L}\,\boldsymbol{L}^{\mathrm{T}}\,\boldsymbol{x}=\boldsymbol{y},$$

hagamos $\boldsymbol{z}=\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}$ y entonces tendremos dos sistemas de ecuaciones

$$\boldsymbol{L}\,\boldsymbol{z}=\boldsymbol{y}, \tag{5.5}$$

$$\boldsymbol{L}^{\mathrm{T}}\,\boldsymbol{x}=\boldsymbol{z}, \tag{5.6}$$

con (5.5) se resuelve para $\boldsymbol{z}$ haciendo una sustitución hacia adelante con

$$z_i=\frac{1}{L_{i,i}}\left(y_i-\sum_{k=1}^{i-1}L_{i,k}\,z_k\right)$$

y con (5.6) resolvemos para $\boldsymbol{x}$ sustituyendo hacia atrás con

$$x_i=\frac{1}{L_{i,i}^{\mathrm{T}}}\left(z_i-\sum_{k=i+1}^{n}L_{i,k}^{\mathrm{T}}\,x_k\right).$$

5.3. Reordenamiento de renglones y columnas

5.3.1. Descripción del problema

Al utilizar la factorización Cholesky para resolver sistemas de ecuaciones, donde $\boldsymbol{A}$ es una matriz simétrica y positiva definida, el primer paso es reordenar los renglones y las columnas de tal forma que se reduzca el número de entradas no cero de la matriz factor $\boldsymbol{L}$. Definamos la notación $\eta(\boldsymbol{L})$, que indica el número de elementos no cero de $\boldsymbol{L}$.

La figura 5.1 muestra los elementos no cero (en negro) de un sistema de ecuaciones para un problema de elemento finito en dos dimensiones, el cual no ha sido reordenado. A la izquierda está la matriz de rigidez $\boldsymbol{A}$, con $\eta(\boldsymbol{A})=1810$, a la derecha la matriz triangular inferior $\boldsymbol{L}$, con $\eta(\boldsymbol{L})=8729$, resultante de la factorización Cholesky de $\boldsymbol{A}$.

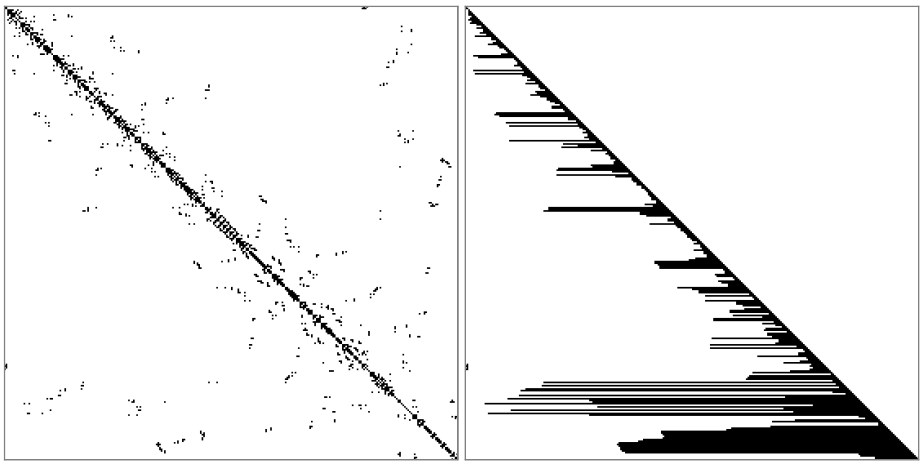


Figura 5.1. Representación de los elementos no cero de una matriz $\boldsymbol{A}$ (izquierda) y su correspondiente factorización $\boldsymbol{L}$ (derecha).

Ahora veamos en la figura 5.2 tenemos que la matriz de rigidez $\boldsymbol{A}'$ con $\eta(\boldsymbol{A}')=1810$ (con la misma cantidad de elementos no nulos que $\boldsymbol{A}$) y su factorización $\boldsymbol{L}'$ tiene $\eta(\boldsymbol{L}')=3215$. Ambas factorizaciones permiten resolver el mismo sistema de ecuaciones. Para determinar este reordenamiento utilizamos las rutinas de la librería METIS [Kary99].

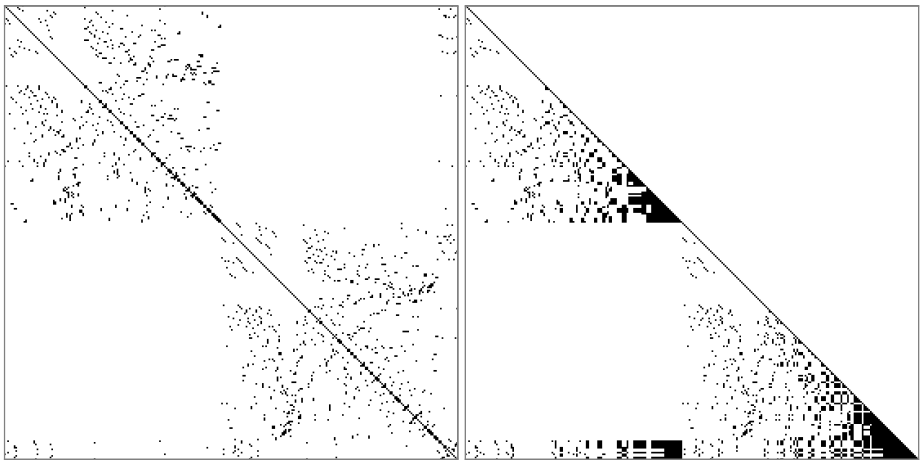


Figura 5.2. Representación de los elementos no cero de una matriz reordenada $\boldsymbol{A}'$ (izquierda) y su correspondiente factorización $\boldsymbol{L}'$ (derecha).

5.3.2. Matrices de permutación

Dada $\boldsymbol{P}$ una matriz de permutación, las permutaciones (reordenamientos) de columnas del tipo

$$\boldsymbol{A}' \leftarrow \boldsymbol{P}\,\boldsymbol{A},$$

o de renglón

$$\boldsymbol{A}' \leftarrow \boldsymbol{A}\,\boldsymbol{P}$$

solas destruyen la simetría de $\boldsymbol{A}$ [Golu96 p148]. Para preservar la simetría de $\boldsymbol{A}$ solamente podemos considerar reordenamiento de las entradas de la forma

$$\boldsymbol{A}' \leftarrow \boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{P}^{\mathrm{T}}.$$

Es de notar que esta permutaciones no mueven los elementos fuera de la diagonal a la diagonal. La diagonal de $\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{P}^{\mathrm{T}}$ es un reordenamiento de la diagonal de $\boldsymbol{A}$.

Dado que $\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{P}^{\mathrm{T}}$ es además simétrica y positiva definida para cualquier permutación de la matriz $\boldsymbol{P}$, podemos entonces resolver el sistema reordenado

$$\left(\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{P}^{\mathrm{T}}\right)\left(\boldsymbol{P}\,\boldsymbol{x}\right)=\left(\boldsymbol{P}\,\boldsymbol{y}\right).$$

La elección de $\boldsymbol{P}$ tendrá un efecto determinante en el tamaño de las entradas no cero de $\boldsymbol{L}$. Calcular un "buen" reordenamiento de la matriz $\boldsymbol{A}$ que minimice las entradas no cero de $\boldsymbol{L}$ es un problema NP completo [Yann81], sin embargo existen heurísticas que generan un reordenamiento aceptable en un tiempo reducido.

5.3.3. Representación de matrices ralas como grafos

Vamos a introducir algunas nociones de teoría de grafos para hacer un análisis de las operaciones con matrices ralas. Un grafo $G=(X, E)$ consiste en un conjunto finito de nodos o vértices X junto con un conjunto E de aristas, los cuales son pares no ordenados de vértices. Un ordenamiento (o etiquetado) α de G es simplemente un mapeo del conjunto $\{1,2,\ldots, N\}$ en X, donde N denota el número de nodos de G. El grafo ordenado por α será denotado como $G^{\alpha}=(X^{\alpha}, E^{\alpha})$.

Sea $\boldsymbol{A}$ una matriz simétrica de $N\times N$, el grafo ordenado de $\boldsymbol{A}$, denotado por $G^{\boldsymbol{A}}=(X^{\boldsymbol{A}}, E^{\boldsymbol{A}})$ en el cual los N vértices de $G^{\boldsymbol{A}}$ están numerados de 1 a N, y $\{x_i, x_j\}\in E^{\boldsymbol{A}}$ si y solo sí $a_{ij}=a_{ji}\neq 0$, $i\neq j$. Aquí x_i denota el nodo de $X^{\boldsymbol{A}}$ con etiqueta i. La figura 5.3 muestra un ejemplo.

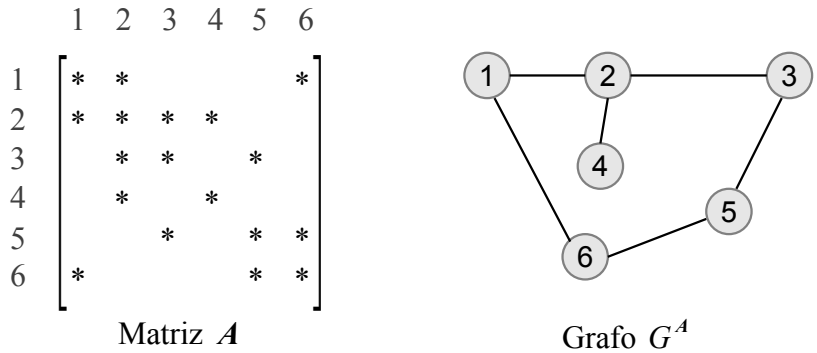
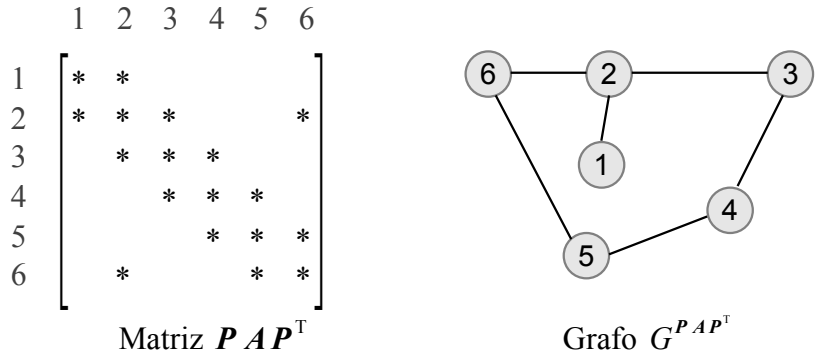


*Figura 5.3. Una matriz y su grafo ordenado, con * se indican las entradas no cero de $\boldsymbol{A}'$.*

Para cualquier matriz de permutación $\boldsymbol{P}\neq\boldsymbol{I}$, los grafos no ordenados (o etiquetados) de $\boldsymbol{A}$ y $\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{P}^{\mathrm{T}}$ son los mismos pero su etiquetado asociado es diferente. Así, un grafo no etiquetado de $\boldsymbol{A}$ representa la

estructura de $\boldsymbol{A}$ sin sugerir un orden en particular. Esta representa la equivalencia de las clases de matrices $\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{P}^{\mathrm{T}}$. Entonces, encontrar una "buena" permutación de $\boldsymbol{A}$ equivale a encontrar un "buen" ordenamiento de su grafo [Geor81]. La figura 5.4 muestra un ejemplo.



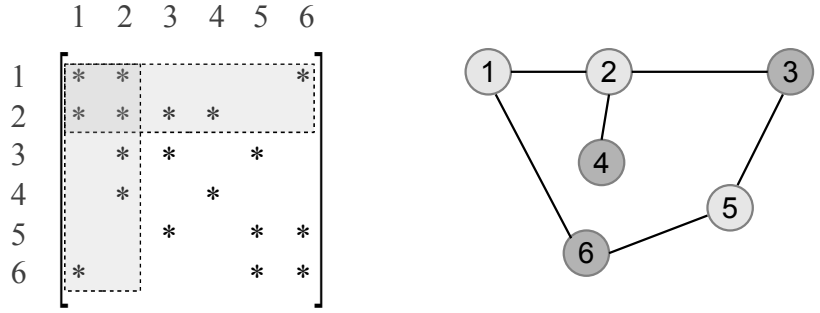
Matriz $\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{P}^{\mathrm{T}}$ Grafo $G^{\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{P}^{\mathrm{T}}}$

Figura 5.4. El gráfico de la figura 5.3 con diferente ordenamiento.
***P** denota una matriz de permutación.*

Dos nodos $x\,,\,y\in X$ en un grafo $G(X\,,E)$ son adyacentes si $\{x\,,\,y\,r\}\in E$. Para $Y\subset X$, el conjunto adyacente de Y, denotado como $\mathrm{ady}(Y)$, es

$$\mathrm{adj}(Y)=\{x\in X-Y\,|\,\{x\,,\,y\}\in E \text{ para algún } y\in Y\}.$$

En otras palabras, $\mathrm{ady}(Y)$ es simplemente el conjunto de nodos en G que no están en Y pero son adyacentes en al menos un nodo de Y. La figura 5.5 muestra un ejemplo.



$$Y=\{x_1\,,\,x_2\}\qquad \mathrm{adj}(Y)=\{x_3\,,\,x_4\,,\,x_6\}\qquad \mathrm{gr}(x_2)=3$$

Figura 5.5. Ejemplo de adyacencia de un conjunto $Y\subset X$.

Para $Y\subset X$, el grado de Y, denotado por $\mathrm{gr}(Y)$, es simplemente el número $|\mathrm{ady}(Y)|$, donde $|S|$ denota el número de miembros del conjunto S. En el caso de que se trate de un solo elemento, consideraremos $\mathrm{gr}(\{x_2\})\equiv\mathrm{gr}(x_2)$.

5.3.4. Algoritmos de reordenamiento

Vamos a hablar muy brevemente de los algoritmos de reordenamiento. La heurística más común utilizada para realizar el reordenamiento es el algoritmo de grado mínimo. El algoritmo 5.1 muestra una versión básica de éste [Geor81 p116].

Dada un matriz $\boldsymbol{A}$ y su correspondiente grafo G_0
$i \leftarrow 1$
repetir
  En el grafo de eliminación $G_{i-1}\left(X_{i-1}, E_{i-1}\right)$, elegir un nodo x_i que tenga grado mínimo.
  Formar el grafo de eliminación $G_i\left(X_i, E_i\right)$ como sigue:
    Eliminar el nodo x_i de G_{i-1} y sus aristas inicidentes
    Agregar aristas al grafo tal que los nodos $\text{adj}\left(x_1\right)$ sean pares adyacentes en G_i.
  $i \leftarrow i+1$
mientras $i > \left|X\right|$

Algoritmo 5.1. Método de grado mínimo para reordenar grafos no dirigidos.

Cuando el grado mínimo se presenta en varios nodos, usualmente se elige uno de forma arbitraria. El ejemplo del reordenamiento obtenido en la figura 5.4 se obtiene aplicando el algoritmo de grado mínimo con la secuencia mostrada en la tabla 5.3.

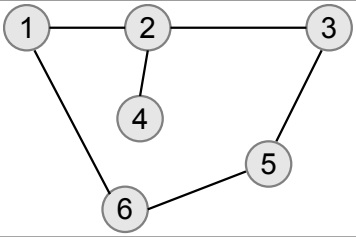
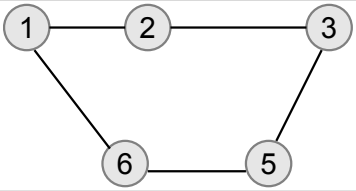
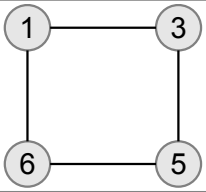
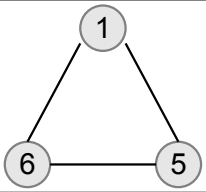
ii	Grafo de eliminación G_{i-1}	Nodo elegido	Grado mínimo
1		4	1
2		2	2
3		3	2
4		5	2
5		1	1
6		6	0

Tabla 5.3. Numeración usando el algoritmo de grado mínimo.

Versiones más avanzadas de este algoritmo pueden consultarse en [Geor89].

Ahora vamos a revisar brevemente el método de disección anidada, el cual es más apropiado para matrices resultantes de problemas de diferencias finitas y elemento finito. La principal ventaja de este

algoritmo comparado con el de grado mínimo es la velocidad y el poder predecir las necesidades de almacenamiento. La ordenación producida es similar a la del algoritmo de grado mínimo.

Vamos a introducir la definición de separador. El conjunto $S \subset X$ es un separador del grafo conectado G si el grafo sección $G(X-S)$ está desconectado. Por ejemplo, en la figura 5.6 se muestra que $S=\{x_3, x_4, x_5\}$ es un separador de G, dado que $G(X-S)$ tiene tres componentes, los conjuntos de nodos $\{x_1\}$, $\{x_2\}$ y $\{x_6, x_7\}$.

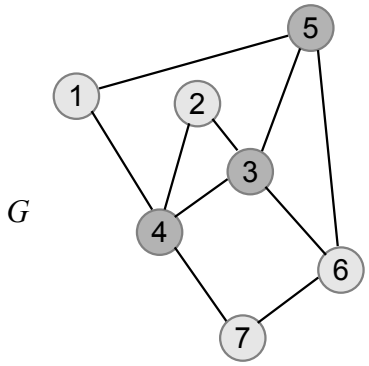


Figura 5.6. Ejemplo de un conjunto separador S.

Sea $\boldsymbol{A}$ una matriz simétrica y G^A su grafo no dirigido asociado. Consideremos un separador S en G^A, cuya remoción desconecta el grafo en dos conjuntos de nodos C_1 y C_2. Si los nodos en S son numerados después de aquellos de C_1 y C_2, entonces se inducirá una partición en la correspondiente matriz ordenada. La observación principal es que los bloques cero en la matriz continúan siendo cero después de la factorización. Cuando es elegido apropiadamente, una submatriz "grande" está garantizada de permanecer cero. La idea puede ser aplicada recursivamente, de tal forma que los ceros puedan ser preservados en la misma forma en las submatrices.

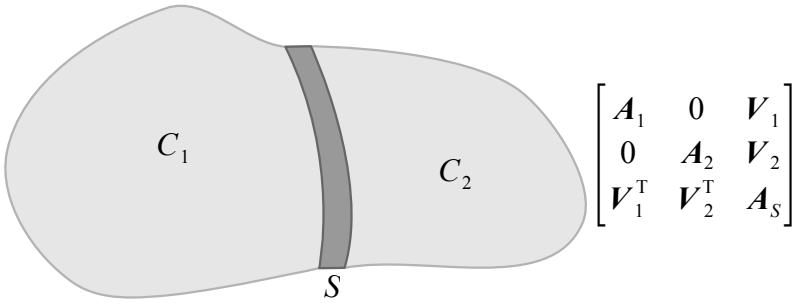


Figura 5.7. El efecto de un conjunto separador S en una matriz.

Esta procedimiento aplicado de forma de forma recursiva se conoce como algoritmo de disección anidada generalizado [Lipt77].

La idea es tratar de dividir el grafo tratando de que sean de igual tamaño con un separador pequeño. Para encontrar este separador se busca generar una estructura grande y entonces elegir un separador de un nivel medio. Este es un algoritmo recursivo que emplea la estrategia de divide y vencerás, a continuación describimos el algoritmo.

Sea S una clase de grafos cerrados en los cuales se cumple el teorema del separador $\sqrt{n}$ [Lipt79]. Sean α, β constantes asociadas con el teorema del separador y sea $G(X, E)$ un grafo de n nodos en S. El algoritmo recursivo 5.2 numera los nodos de G tal que la eliminación gaussiana rala (factorización Cholesky rala) es eficiente. El algoritmo supone que l de los nodos de G ya contienen números asignados, cada uno de los

cuales es más grande que b (se explica más adelante). El objetivo es numerar los nodos restantes de G consecutivamente de a a b.

Sea dado $G(X, E)$
inicio
  si $|G| \leq \left(\frac{\beta}{(1-\alpha)}\right)^2$
    Los nodos son ordenados arbitrariamente de a a b (puede utilizarse el algoritmo de grado mínimo)
  si_no
    Encontrar conjuntos A, B y C que satisfagan el teorema del separador $\sqrt{n}$, donde C es el conjunto separador. Al remover C se divide el resto de G en dos conjuntos A y B los cuales no tienen que ser conexos. Sea A conteniendo i nodos no numerados, B contiene j y C contiene k nodos no numerados.
    Numerar los nodos no numerados en C de forma arbitraria de $b-k+1$ a b, es decir, estamos asignando a los nodos de C los números más grandes.
    Eliminar todos las aristas cuyas conexiones estén ambas en C.
    Aplicar el algoritmo recursivamente al subgrafo inducido por $B \cup C$ para numerar los nodos no numerados en B de $a \leftarrow b-k-j+1$ a $b \leftarrow b-k$.
    Aplicar el algoritmo recursivamente al subgrafo inducido por $A \cup C$ para numerar los nodos no numerados en B de $a \leftarrow b-k-j-i+1$ a $b \leftarrow b-k-j$.
  fin_si
fin

Algoritmo 5.2. Algoritmo de disección anidada generalizado.

Se inicia el algoritmo 5.2 con todos los nodos de G no numerados, con $a \leftarrow 1$, $b \leftarrow n$ y $l \leftarrow 0$. Esto enumerará los nodos en G de 1 a n. En este algoritmo los nodos en el separados son incluidos en la llamada recursiva pero son no numerados.

Una versión mejorada de este algoritmo es la empleada en la librería METIS [Kary99], ésta es la que hemos utilizado en la implementación de nuestro programa.

5.4. Factorización Cholesky simbólica

Cuando trabajamos con matrices grandes y ralas, es muy costoso calcular directamente $\boldsymbol{L}$ utilizando (5.3) y (5.4), un mejor método es determinar que elementos de $\boldsymbol{L}$ son distintos de cero y llenarlos utilizando entonces (5.3) y (5.4). El algoritmo para determinar los elementos distintos cero en $\boldsymbol{L}$ se le denomina factorización simbólica [Gall90 p86-88].

Para una matriz rala $\boldsymbol{A}$, definamos

$$\boldsymbol{a}_j \stackrel{\text{def}}{=} \{k > j \mid A_{kj} \neq 0\},\ j = 1 \ldots n \tag{5.7}$$

como el conjunto de los índices de los elementos no nulos de la columna j de la parte estrictamente triangular inferior de $\boldsymbol{A}$.

De forma análoga definimos para la matriz $\boldsymbol{L}$, los conjuntos

$$\boldsymbol{l}_j \stackrel{\text{def}}{=} \{k > j \mid L_{kj} \neq 0\},\ j = 1 \ldots n. \tag{5.8}$$

Requeriremos de conjuntos $\boldsymbol{r}_j$ que serán usados para registrar las columnas de $\boldsymbol{L}$ cuyas estructuras afectarán a la columna j de $\boldsymbol{L}$.

para $j \leftarrow 1 \ldots n$
  $\boldsymbol{r}_j \leftarrow \emptyset$
  $\boldsymbol{l}_j \leftarrow \boldsymbol{a}_j$
  para $i \in \boldsymbol{r}_j$
    $\boldsymbol{l}_j \leftarrow \boldsymbol{l}_j \cup \boldsymbol{l}_i \setminus \{j\}$
  fin para
  $p \leftarrow \begin{cases} \min\{i \in \boldsymbol{l}_j\} & \text{si } \boldsymbol{l}_j \neq \emptyset \\ j & \text{otro caso} \end{cases}$
  $\boldsymbol{r}_p \leftarrow \boldsymbol{r}_p \cup \{j\}$
fin para

Algoritmo 5.3. Factorización Cholesky simbólica.

Esta algoritmo de factorización simbólica es muy eficiente, la complejidad en tiempo y espacio es de orden $O(\eta(\boldsymbol{L}))$.

Vamos ahora a mostrar visualmente como funciona la factorización simbólica, ésta puede ser vista como una secuencia de grafos de eliminación [Geor81 pp92-100]. Dado $\boldsymbol{H}_0 = \boldsymbol{A}$, podemos establecer una correspondencia entre una transformación de $\boldsymbol{H}_0$ a $\boldsymbol{H}_1$ como los cambios correspondientes en sus grafos respectivos. Denotamos $\boldsymbol{H}_0$ por $G^{\boldsymbol{H}_0}$ y $\boldsymbol{H}_1$ por $G^{\boldsymbol{H}_1}$. Dado un ordenamiento α implicado por $G^{\boldsymbol{A}}$, denotemos el nodo $\alpha(i)$ por x_i. Como se muestra en la figura 5.8, el grafo de $\boldsymbol{H}_1$ es obtenido de $\boldsymbol{H}_0$ por:

1) Eliminar el nodo x_1 y sus aristas incidentes

2) Agregar las aristas a el grafo tal que los nodos en $\text{adj}(x_1)$ sean pares adyacentes en $G^{\boldsymbol{H}_1}$.

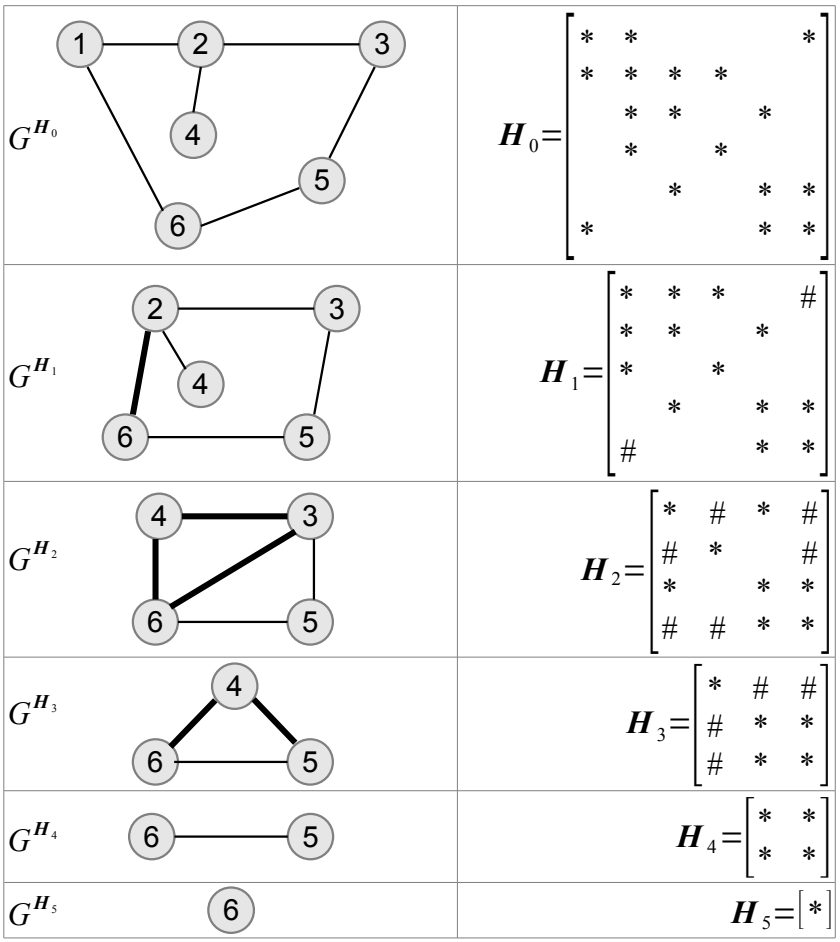


Figura 5.8.Secuencia de grafos de eliminación.

El grafo llenado y su matriz correspondiente se muestran en la figura 5.9, las entradas nuevas se indican con #. Sea $\boldsymbol{L}$ la matriz triangular factor de la matriz $\boldsymbol{A}$. Definamos el grafo llenado de $G^{\boldsymbol{A}}$ como el grafo simétrico $G^{\boldsymbol{F}} = (X^{\boldsymbol{F}}, E^{\boldsymbol{F}})$, donde $\boldsymbol{F} = \boldsymbol{L} + \boldsymbol{L}^{\mathrm{T}}$. Así el conjunto de aristas $E^{\boldsymbol{F}}$ consiste de todas las aristas en $E^{\boldsymbol{A}}$ junto con todos las aristas agregadas durante la factorización. Obviamente $X^{\boldsymbol{F}} = X^{\boldsymbol{A}}$.

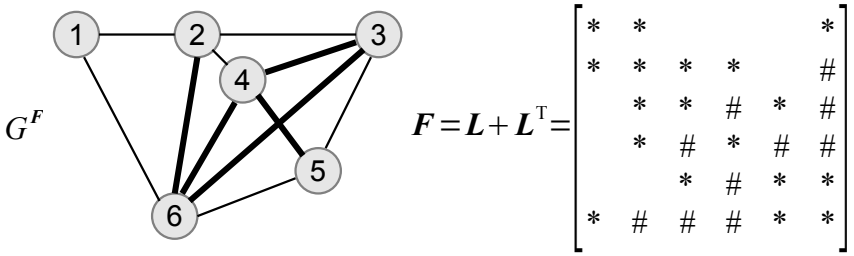


Figura 5.9. Resultado de la secuencia de eliminación.

5.5. Implementación

5.5.1. En dos dimensiones

La gráfica 5.1 muestra los resultados obtenidos con la implementación del algoritmo de factorización Cholesky simbólica para la solución de un problema de elemento finito en dos dimensiones. Como comparación extra se muestran los tiempos de solución utilizando el algoritmo de gradiente conjugado sin paralelizar. Al igual que en la implementación del método del gradiente conjugado, se utiliza el método de *compressed row storage* para el almacenamiento de las matrices ralas. Para realizar estas mediciones se utilizó una malla regular, como la de la figura 5.10, con diferentes densidades de malla.

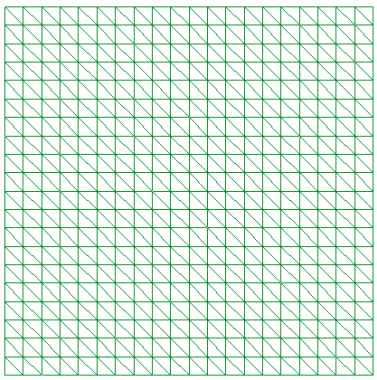
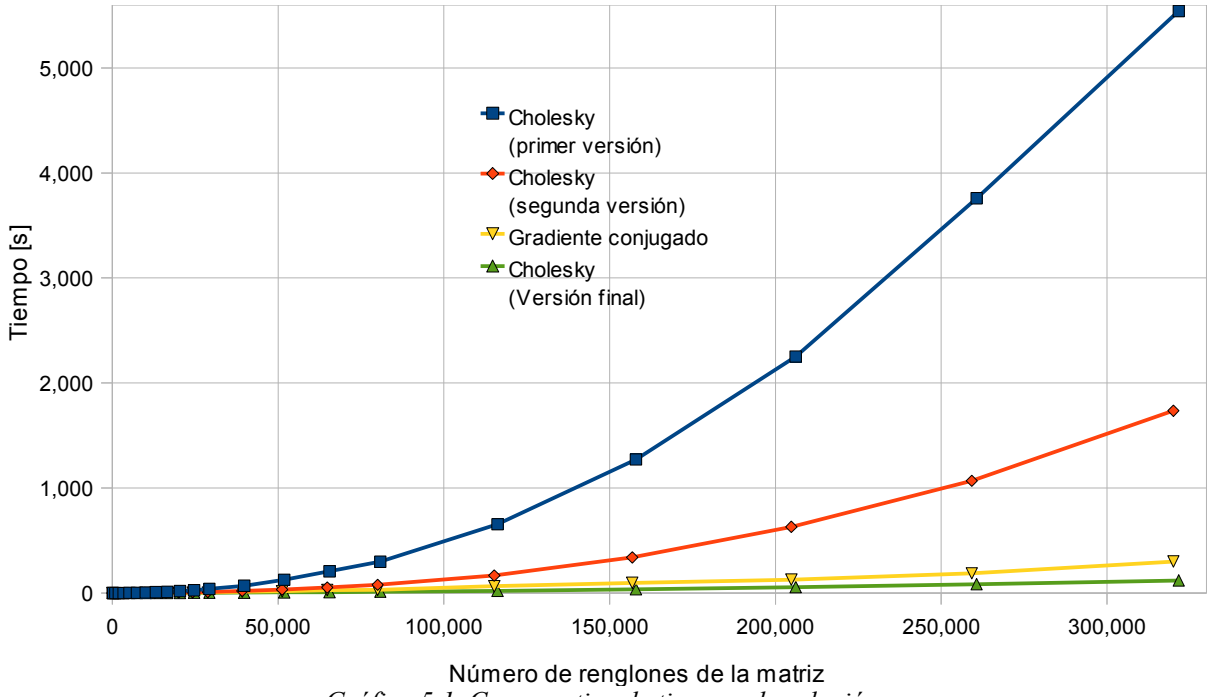


Figura 5.10. Mallado de ejemplo.

No es posible mostrar un comparativo con una factorización clásica de Cholesky para un matriz completa, dado que la cantidad de memoria utilizada aún para matrices de tamaño reducido es excesiva. Por ejemplo, para una matriz de 100,000x100,000 entradas se necesitarían 40 gigabytes de memoria (con doble precisión).

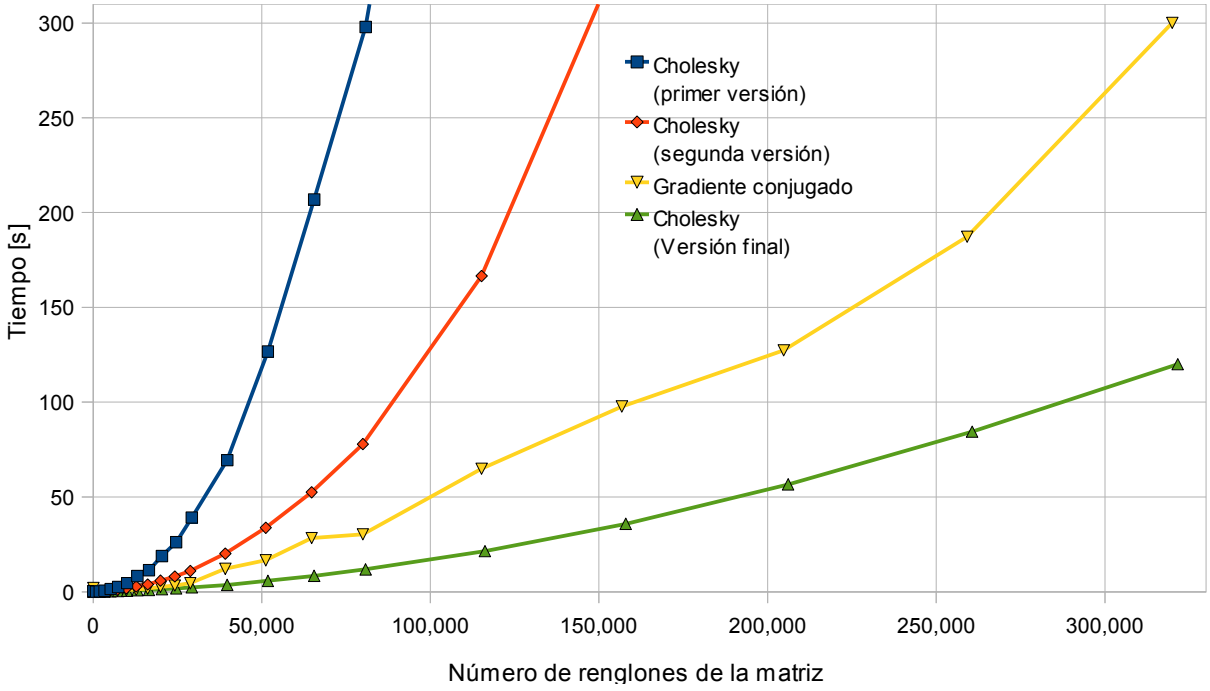


Gráfica 5.1. Comparativo de tiempos de solución.

Una vez implementada la primer versión se realizó un análisis de los cuellos de botella del algoritmo 5.3. El primer cuello de botella fue en el llenado de las columnas $\boldsymbol{l}_j$. Para reducir el tiempo tratando de minimizar el uso de memoria usamos una matriz de bits triangular inferior que almacena valores *true* si la entrada existirá en $\boldsymbol{L}$ y *false* en caso contrario. Como mejora posterior se cambió el formato de ésta matriz a *sky-line*, con lo que se logró una reducción en el uso de memoria en la factorización simbólica en aproximadamente un 66%.

La segunda versión del algoritmo se toma en cuenta la modificación del cuello de botella que daba el mayor tiempo de procesamiento, la ecuación (5.3) en la búsqueda de las entradas L_{jk}. Para mejorar la búsqueda se optó por reordenar las entradas de los los elementos de las matrices ralas en base a sus índices. Si los índices de las entradas por renglón no son ordenadas, se tendrá un costo de búsqueda de la entrada de orden $O(n)$ en el peor caso. Al ordenar y aplicar un algoritmo de búsqueda binaria, se redujo el costo de la búsqueda a un orden $O(\log_2 n)$ en el peor caso. Tenemos la ventaja que, para el caso de multiplicación matriz vector, el orden de búsqueda sigue siendo $O(1)$. Esto es porque no se hace una búsqueda sino que se toman los elementos de cada renglón uno tras otro.

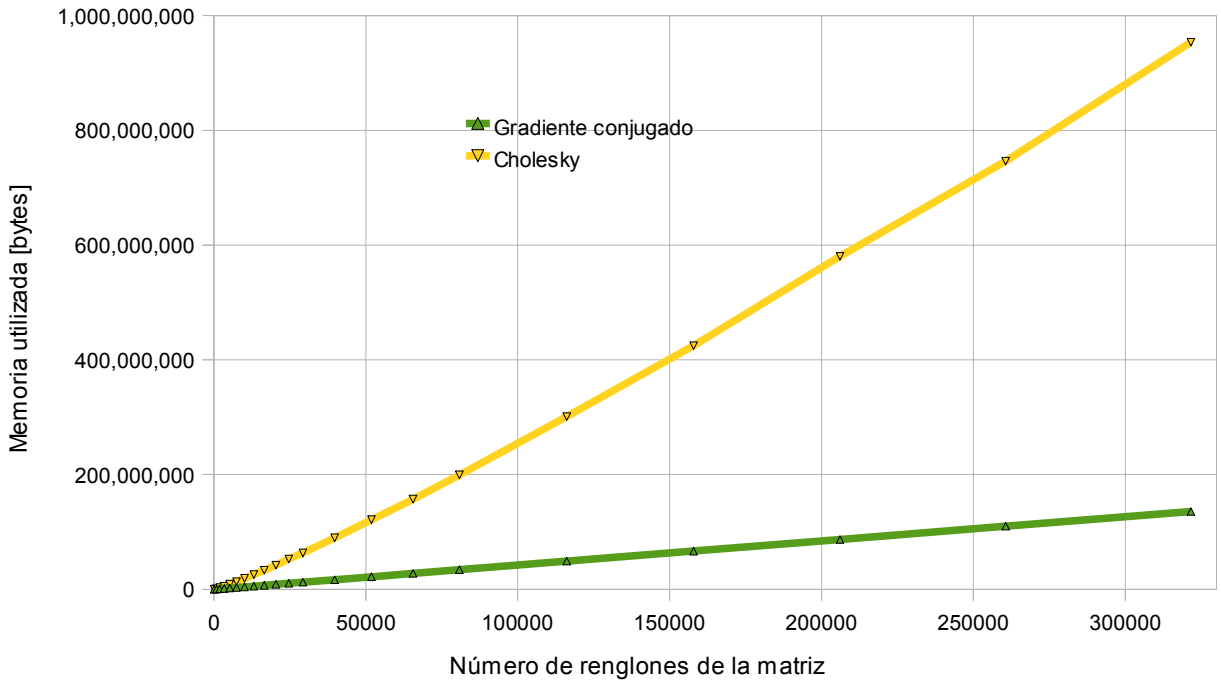
La versión final del algoritmo tiene considerable mejora con respecto a la segunda versión. Se modificó el acceso a las entradas L_{ik} y L_{jk} de tal forma que no se hicieran búsquedas. El algoritmo final recorre los índices de los renglones i y j de la matriz rala $\boldsymbol{L}$, tomando en cuenta sólo los índices comunes para calcular (5.3). Una vista a detalle de la gráfica 5.1 es mostrada en la gráfica 5.2.



Gráfica 5.2. Detalle del comparativo de tiempos de solución.

La identificación de estos cuellos de botella y su corrección reducen el tiempo de solución del algoritmo como se muestra en la gráfica 5.1. Se logró que la solución de los sistemas de ecuaciones utilizando la factorización Cholesky simbólica sea más rápida que la solución con gradiente conjugado.

La gráfica 5.3 indica la utilización de memoria comparando el algoritmo final de factorización Cholesky simbólica con respecto al gradiente conjugado.



Gráfica 5.3. Comparativo de utilización de memoria de los algoritmos de gradiente conjugado y Cholesky.

Es claro que el algoritmo de factorización Cholesky utiliza mucha más memoria que el gradiente conjugado.

6. Resultados

6.1. Preparación

La tabla 6.1 resume las características del cluster de cómputo con el cual se realizaron las pruebas que se muestran a continuación.

Nodo Maestro	Procesador: AMD Quad Core Opteron 2350 HE (8 cores) Memoria: 32 GB Disco Duro: SATA 250 GB, 1000GB
Cluster 1 **16 Nodos Esclavos**	Procesador: 2 x AMD Quad Core Opteron 2350 HE (8 cores) Memoria: 12 GB Disco Duro: SATA 160 GB
Cluster 2 **15 Nodos Esclavos**	Procesador: Intel(R) Xeon(R) CPU E5502 (4 cores) Disco Duro: SATA 160 GB Memoria: 16 GB
Red	Switch SMC: 48 ptos 1 Gbps
Sistema operativo	Rocks Cluster Linux 5.3 (64 bits)
Compiladores	GCC 4.4.3 (con soporte para OpenMP) C, C++, Fortran
Librería MPI	Open MPI 1.4.1
Total	Núcleos de procesamiento (CPU): 192 Capacidad en memoria: 444 GB Capacidad en disco : 6120 GB

Tabla 6.1. Características del cluster de prueba.

A fin de poder valorar la escalabilidad de las estratégias de solución se utilzó un solo problema para hacer pruebas, éste es mostrado en la figura 6.1.

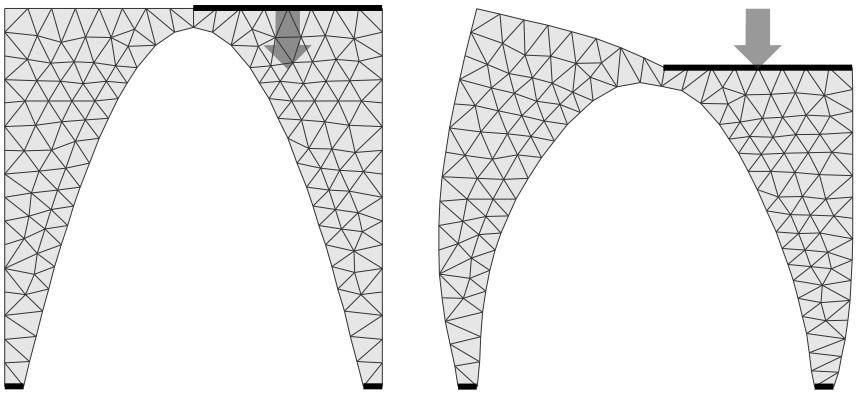


Figura 6.1. Descripción del problema patrón (iquierda problema, derecha solución).

El problema consiste en arco en dos dimensiones compuesto de un solo material, las condiciones de frontera impuestas son: las bases del arco están fijas, en la parte superior derecha se impone un

desplazamiento vertical. Al problema se le generarán mallas de elemento finito con diferentes grados de refinamiento y se resolverá utilizando diversas estratégias, las cuales se describen a continuación.

6.2. Descomposición de dominio y gradiente conjugado

La siguiente prueba se efectuó utilizando una de las computadoras del "Cluster 2" del CIMAT. El siguiente resultado consiste en el problema de la figura 6.1 mallado en 3'960,100 elementos con 1'985,331 nodos, es decir un sistema de 3'970,662 de ecuaciones. El resultado de este problema se muestra en la figura 6.2.

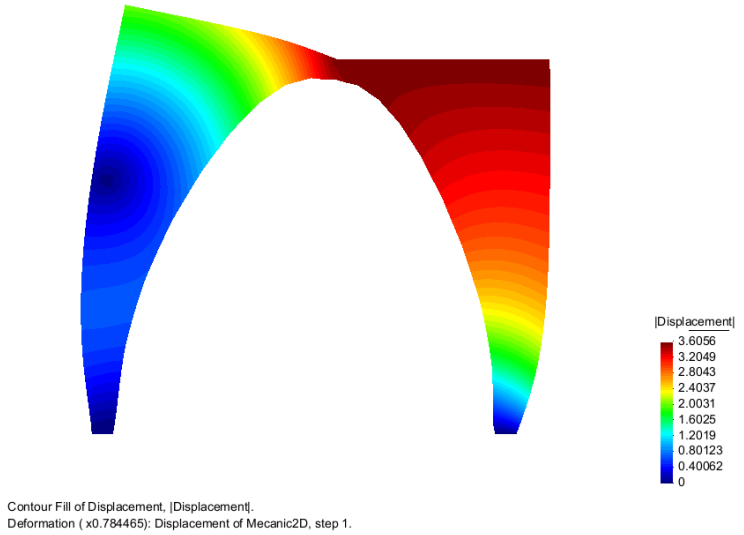


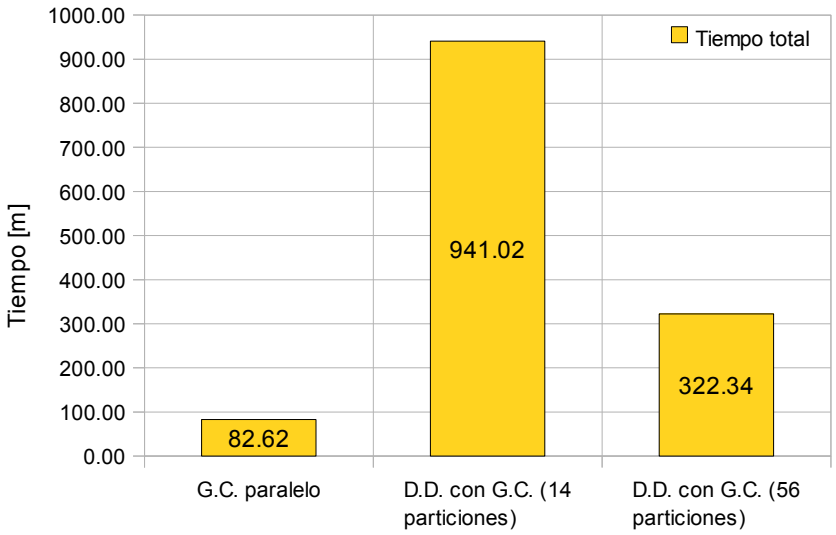
Figura 6.2. Diagrama del resultado del problema de ejemplo en dos dimensiones.

La primer columna de la gráfica 6.1 muestra el resultado de este problema, utilizando una sola computadora del cluster (sin descomposición de dominio) resolviendo el sistema de ecuaciones completo utilizando gradiente conjugado paralelizado con 4 CPUs con memoria compartida, el tiempo que tardó en resolver el sistema fue de 82.62 minutos. Este resultado nos servirá como una referencia de cuánto más eficiente es la solución del problema utilizando descomposición de dominios.

6.2.1. Gradiente conjugado paralelizado

En las siguientes pruebas se siguió utilizando el "Cluster 2" del CIMAT, utilizamos las 14 computadoras del cluster, cada una con cuatro procesadores, lo que nos da un total de 56 CPUs.

En la segunda columna de la gráfica 6.1 mostramos el resultado de utilizar el esquema híbrido en una descomposición de dominios de 14 particiones (una por nodo) con 20 capas de traslape, utilizando como solver el gradiente conjugado paralelizado en cada computadora con 4 CPUs con memoria compartida. El tiempo total para resolver el sistema fue de 941.02 minutos. Es decir, 11.38 veces más lento que el resultado de la primer columna.



Gráfica 6.1. Comparación de estrategias.

El problema que encontramos con este esquema es que el tiempo que requiere el gradiente conjugado para converger es muy diferente en cada una de las particiones. Las gráficas de la figura 6.3 muestran la carga de procesamiento durante 21 iteraciones de Schwarz en dos nodos del cluster, el más eficiente y el menos eficiente, a la izquierda y a la derecha respectivamente. En el nodo más eficiente se ve que después de terminar de resolver el sistema tiene periodos de baja actividad, durante este tiempo este nodo está esperando que los otros nodos terminen. En comparación, el nodo menos eficiente tarda más en resolver el sistema de ecuaciones, al ser el más lento no muestra periodos de espera, en la gráfica se aprecia que está trabajando continuamente. El nodo menos eficiente alenta a todos los nodos del cluster.

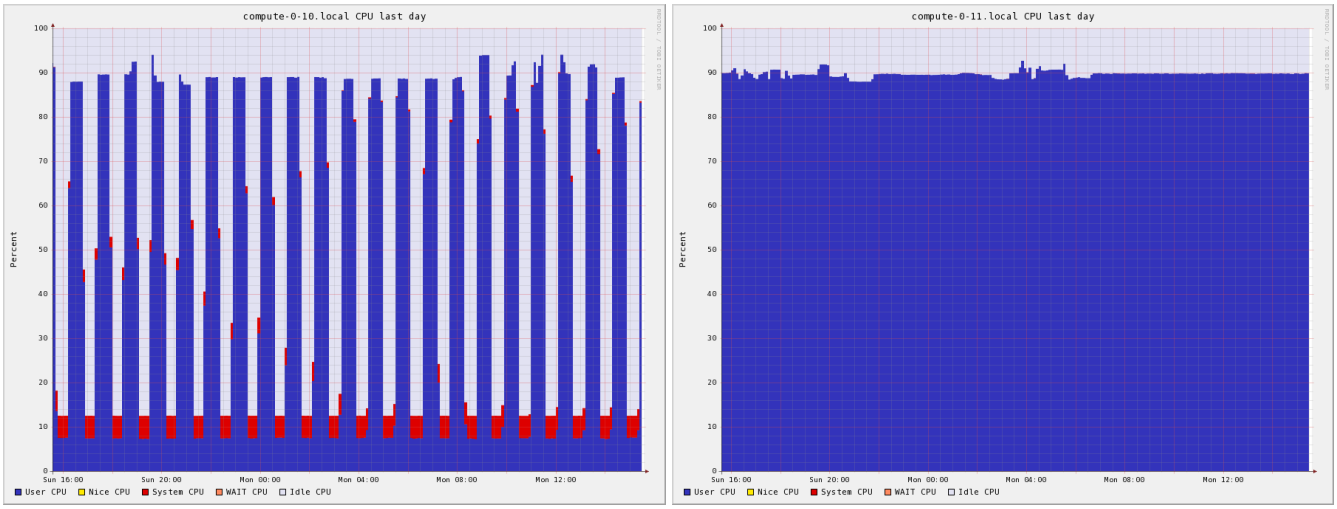


Figura 6.3. Diferencia de carga en los nodos más eficiente (izquierda) y menos eficiente (derecha)

Como resultando tenemos un gran desbalance de carga que hace que la mayoría de los procesos estén inactivos casi un 50% del tiempo Esto provoca que la eficiencia de operación del cluster se reduzca significativamente, como se aprecia en la figura 6.4, la cual muestra el nivel de operación de todo del cluster.

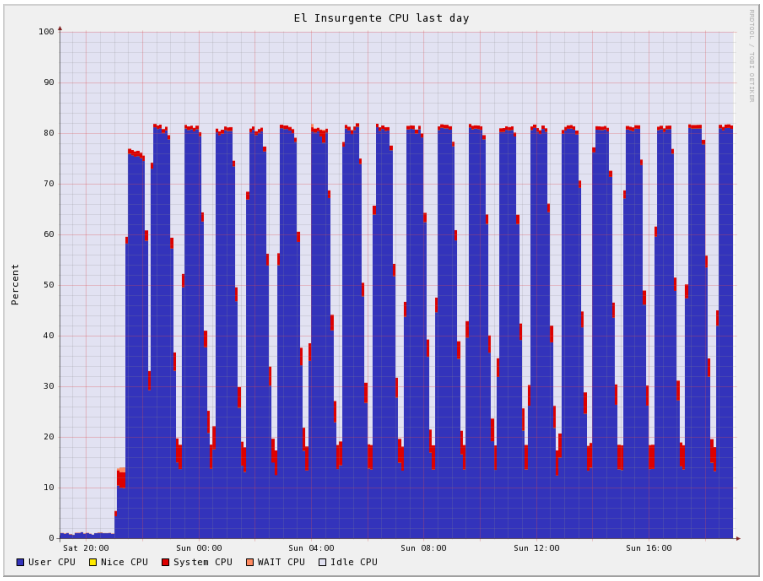


Figura 6.4. Detalle del trabajo de CPU general del cluster.

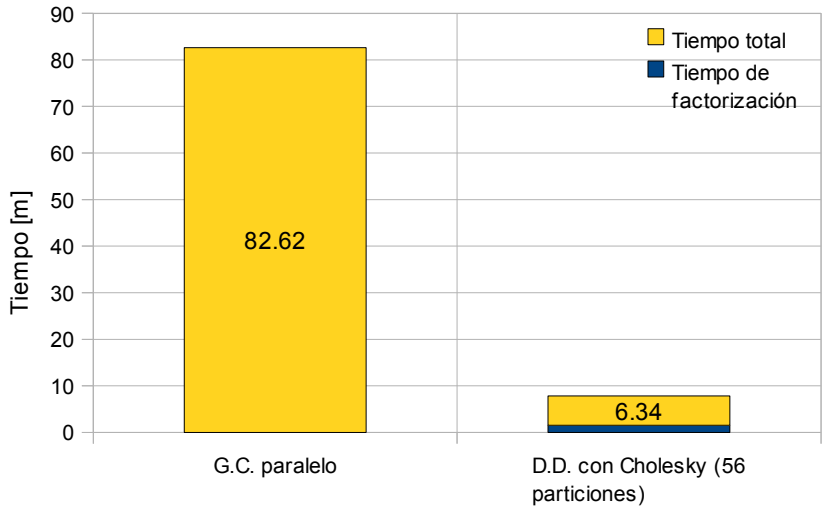
6.2.2. Gradiente conjugado sin paralelizar

Ahora con una configuración diferente, vamos a utilizar la descomposición de dominio utilizando el gradiente conjugado no paralelizado, para esto dividiremos el dominio en 56 particiones con 20 capas de traslape, un solver por cada CPU del cluster. La tercer columna de la gráfica 6.1 muestra el tiempo obtenido de 322.34 minutos. Casi la tercera parte del tiempo con en comparación del con gradiente conjugado paralelizado, pero aún cuatro veces más lento que la versión sin descomposición de dominio. Al tener particiones más pequeñas, y por lo tanto sistemas de ecuaciones más pequeños el desbalance de carga se hace menor, aunque sigue siendo la causa principal de la poca eficiencia de esta estrategia.

6.3. Descomposición de dominio y factorización Cholesky simbólica

Entre más pequeño sea el sistema de ecuaciones más eficiente será la resolución por factorización Cholesky, es por eso que en vez de utilizar 14 particiones, como en el caso con gradiente conjugado paralelizado, utilizaremos 56 particiones (una por cada procesador del cluster) con 20 capas de traslape.

La gráfica 6.2 muestra dos resultados, la columna de la izquierda es la misma que la de la primer columna de la gráfica 6.1 (es la solución del sistema sin descomposición de dominio utilizando una sola computadora con 4 procesadores y aplicando el algoritmo de gradiente conjugado en paralelo). A la derecha el resultado de utilizar descomposición de dominios utilizando como solver la factorización Cholesky para matrices ralas mostrada en el capítulo 5.



Gráfica 6.2. Comparación de estrategias.

El tiempo utilizando gradiente conjugado en paralelo en una computadora fue de 82.62 minutos, el tiempo con descomposición de dominio y factorización Cholesky en 14 computadoras fue de 6.34 minutos. Se tuvo la solución aproximadamente 13 veces más rápido. Esto se debe a que el tiempo por iteración Schwarz es muy reducido, ya que solo se resuelven dos matrices triangulares con sustitución hacia atrás y hacia adelante en cada partición. El ahorro en tiempo es notable.

6.4. Evolución y convergencia

A continuación, en la figura 6.5, mostramos un ejemplo de la evolución de las iteraciones del método alternante de Schwarz en el problema de la figura 6.1 dividido en 370 elementos y 239 nodos, utilizando cuatro particiones con dos capas de traslape.

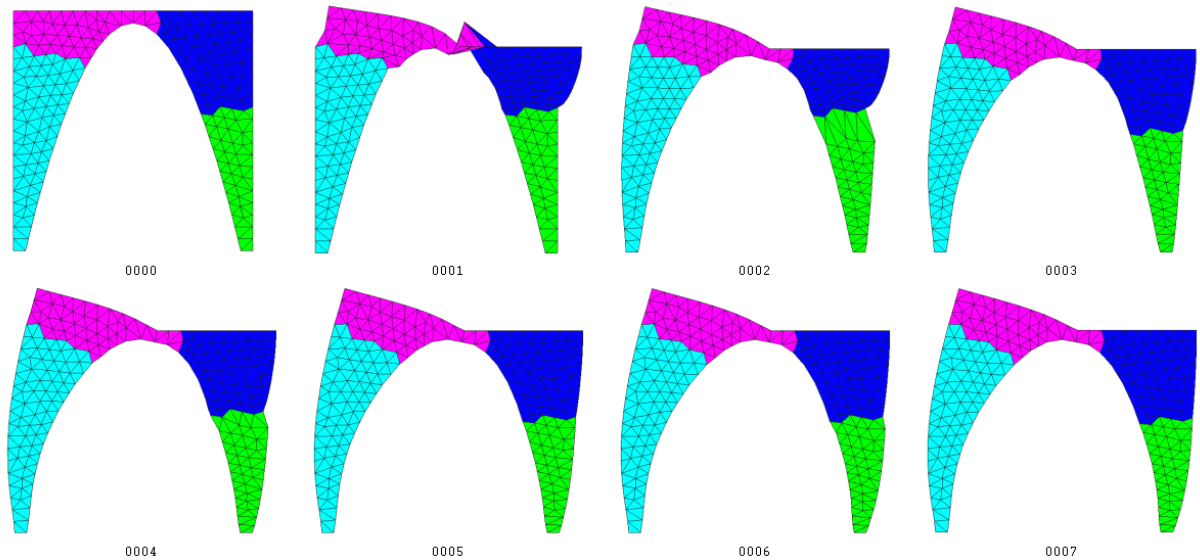
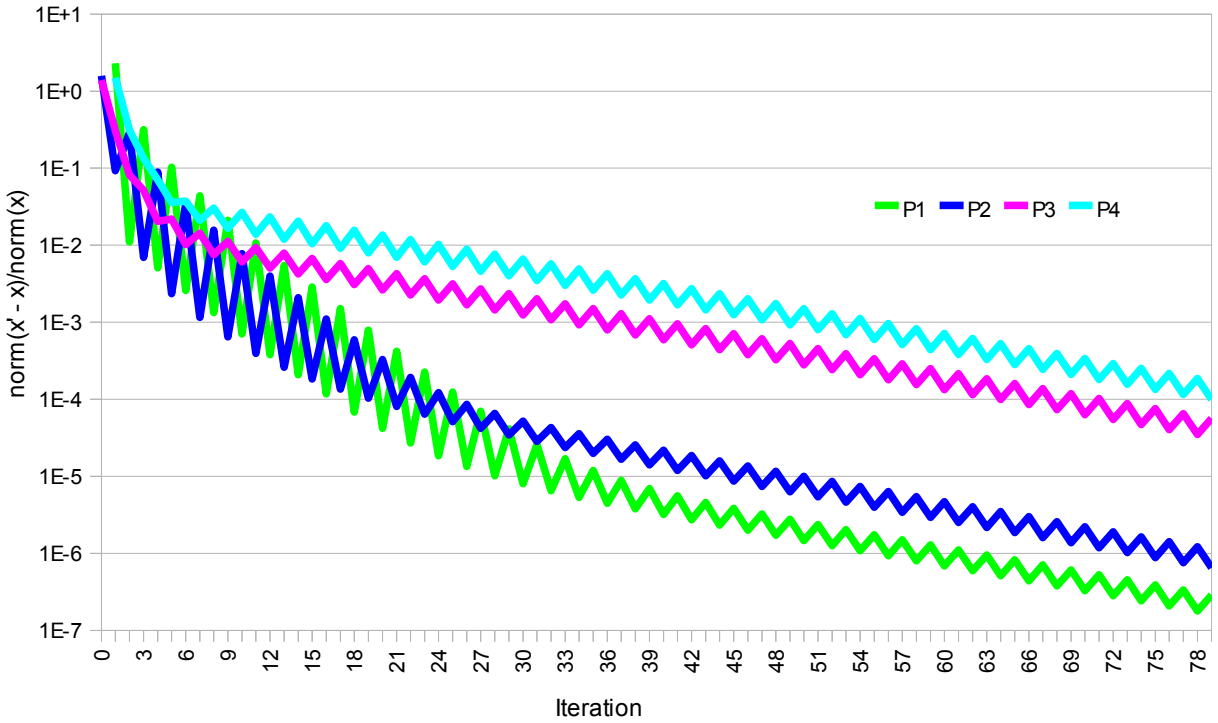


Figura 6.5. Ejemplo de evolución en las primeras ocho iteraciones.

La gráfica 6.3 muestra la convergencia d^i de cada partición, medida como la norma ponderada de la diferencia entre la solución actual y la anterior

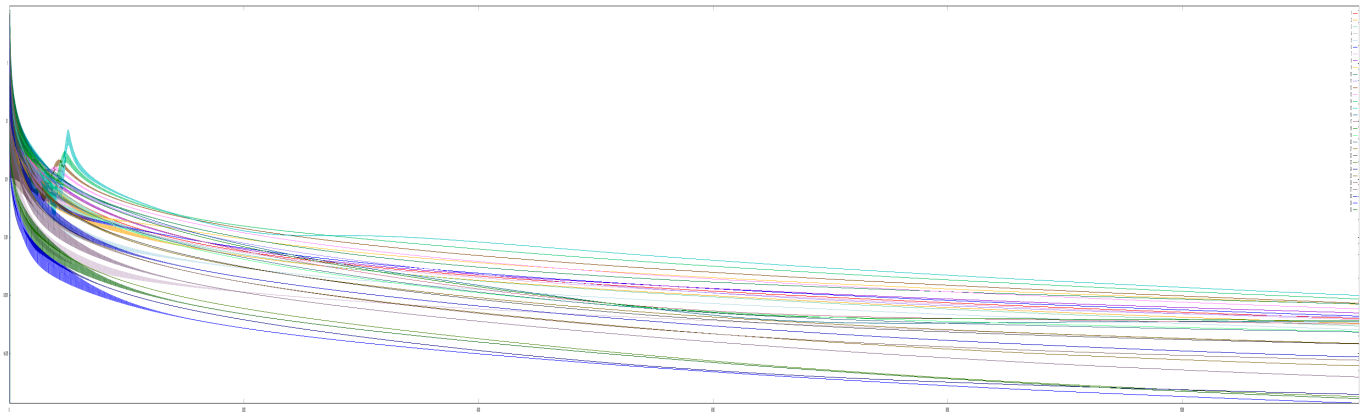
$$d^i = \frac{\left\| \boldsymbol{u}_{t-1}^i - \boldsymbol{u}_t^i \right\|}{\left\| \boldsymbol{u}_t^i \right\|},$$

donde $\boldsymbol{u}_t^i$ es el vector de desplazamiento resultante de resolver el sistema de ecuaciones de la partición i. El criterio de paro establecido para el método alternante de Schwarz es cuando $d^i < 1\,x\,10^{-4}$ para $i = 1, \ldots, 4$.



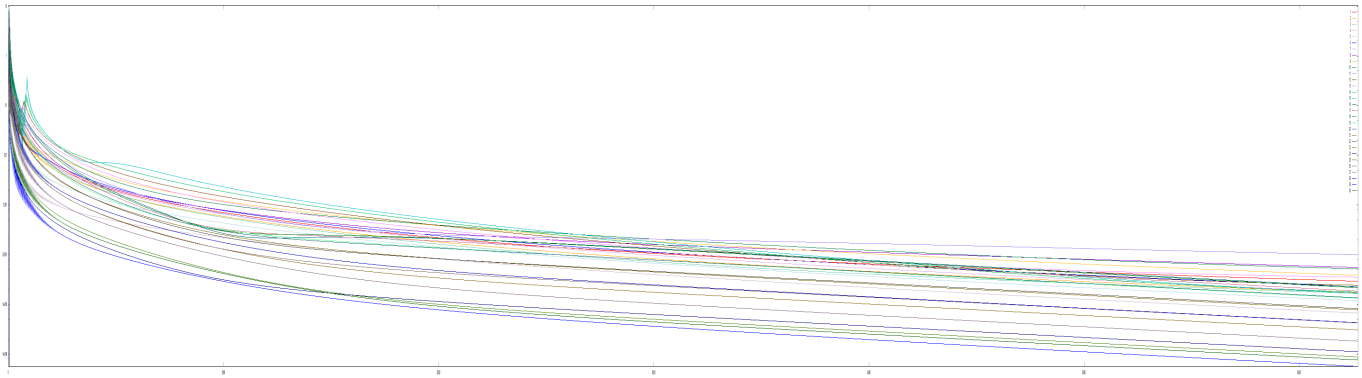
Gráfica 6.3. Evolución de la convergencia.

En problemas con mallas más grandes se ven otro tipo de efectos. Los siguientes resultados muestran la evolución de la convergencia del problema de la figura 6.1 dividido en 3’960,100 elementos y 1’985,331 nodos (3’970,662 ecuaciones), utilizando 30 particiones. En la gráfica 6.4 se ve la evolución con una capa de traslape. Esta es interesante, se ve que en las primeras iteraciones hay unos picos donde empeora la convergencia en algunas particiones, para después mejorar.



Gráfica 6.4. Evolución de la convergencia con una capa de traslape.

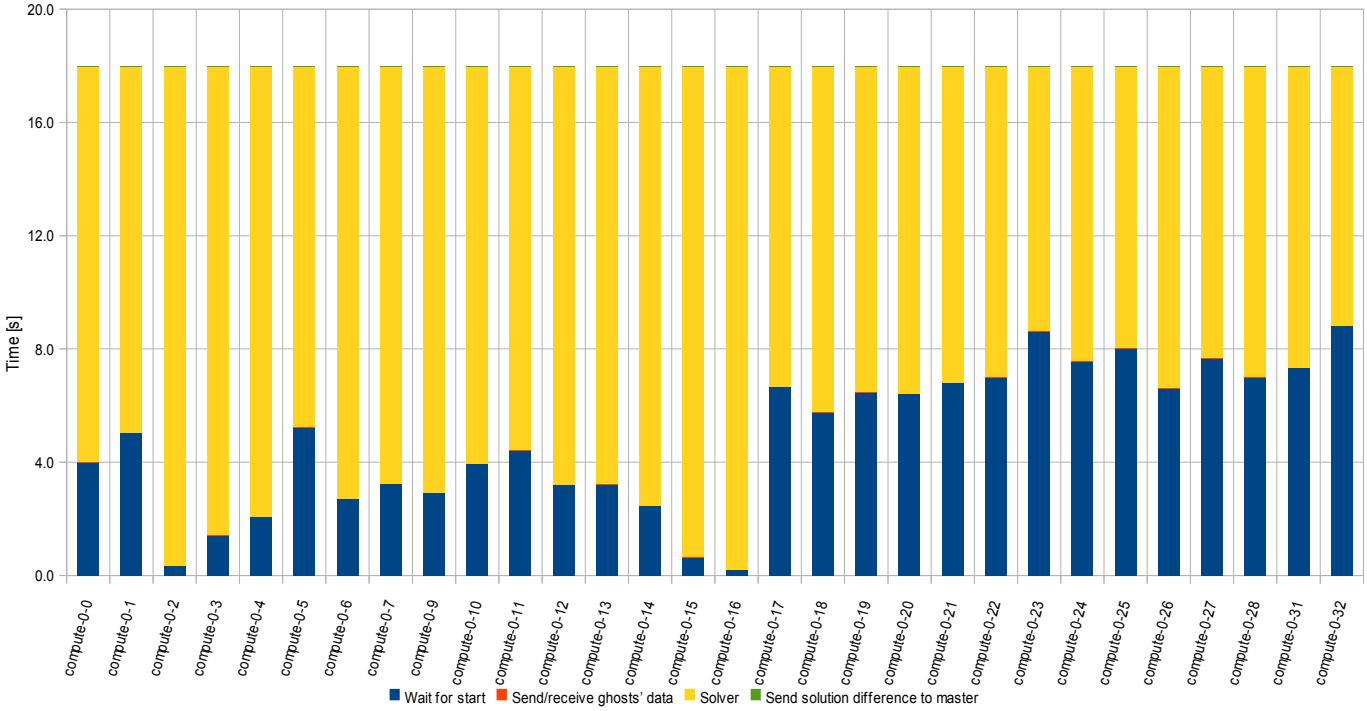
La gráfica 6.5 se ve la evolución del mismo problema, pero utilizando siete capas de traslape. En ésta se observa que los valores de la convergencia d^i presentan los mismos picos pero de forma más temprana.



Gráfica 6.5. Evolución de la convergencia con siete capas de traslape.

6.5. Distribución de tiempo

La gráfica 6.6 muestra la distribución de tiempos en dada iteración de Schwarz con el problema de la figura 6.1 dividido en 3’960,100 elementos y 1’985,331 nodos (3’970,662 ecuaciones), utilizando 33 particiones. El tiempo necesario para la transmisión de datos no es notorio.



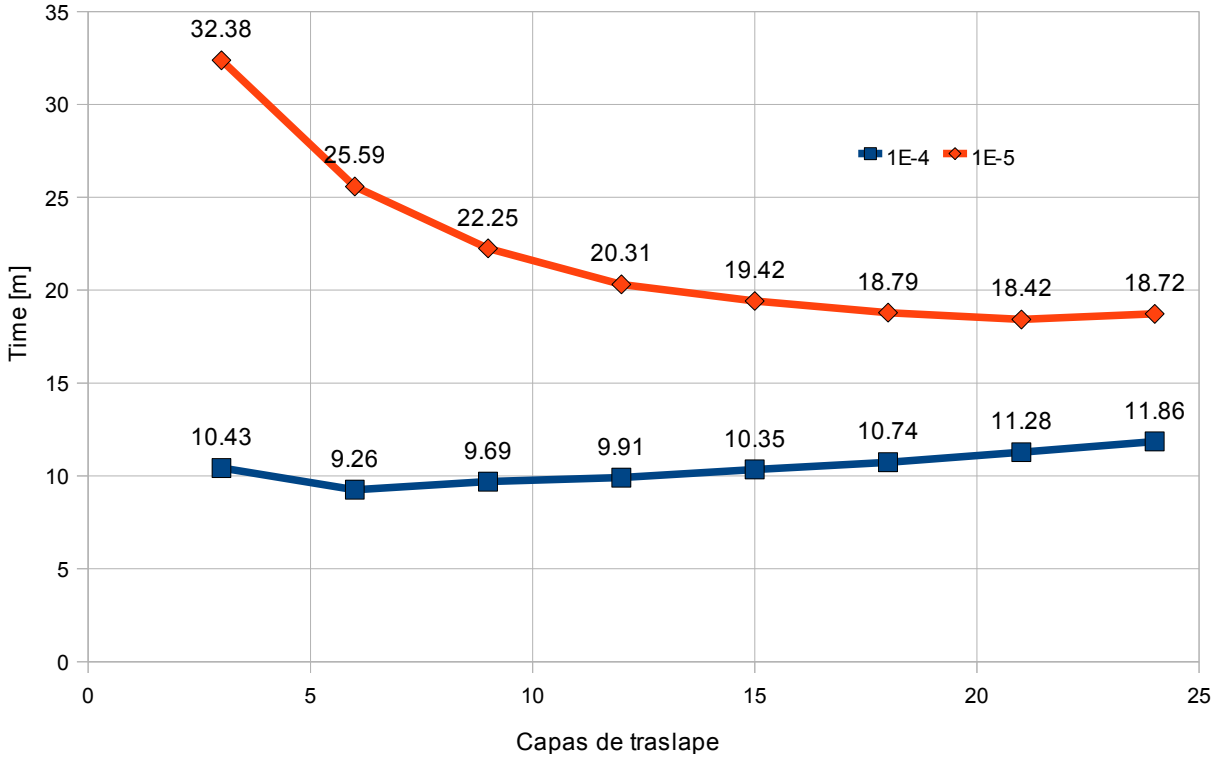
Gráfica 6.6. Distribución de tiempo del algoritmo.

Se observa que el tiempo utilizado por el solver varía en cada nodo, todos los nodos tienen que esperar al solver más lento, el del nodo 16 en este caso.

6.6. Traslape

Los siguientes resultados son para un problema de dos dimensiones con 3’960,100 elementos y 1’985,331 nodos (3’970,662 ecuaciones), dividido en 52 particiones resuelto con factorización Cholesky simbólica.

Traslape	Tolerancia	
	$1x10^{-4}$	$1x10^{-5}$
3	10.43	32.38
6	9.26	25.59
9	9.69	22.25
12	9.91	20.31
15	10.35	19.42
18	10.74	18.79
21	11.28	18.42
24	11.86	18.72



Gráfica 6.7. Eficiencia en tiempo variando las capas de traslape entre las particiones.

6.7. Afinidad del CPU

Uno de los problemas más importantes que encontramos al realizar pruebas asignando procesos a todos los CPU de una computadora es que la intercomunicación con OpenMPI entre estos procesos es realizada utilizando memoria compartida monitoreada con *polling*. A fin de ser eficiente, OpenMPI hace un *polling* muy intenso, en el orden de microsegundos por petición. Para mantener este monitoreo, OpenMPI crea

varios *threads*, el problema es que estos *threads*, que pueden llevar el uso del CPU al 100%, son asignados por el sistema operativo para ejecutarse en los otros CPU de la computadora, lo que alenta en gran medida los procesos asignados previamente a estos CPU. Como todos los procesos crean threads para comunicarse con los otros procesos en la misma computadora se produce una caída de la eficiencia considerable al interferirse entre ellos.

La solución que encontramos fue establecer la afinidad a los procesos a un CPU, de esta forma los threads creados por este proceso se ejecutarán en el mismo CPU que el proceso. De tal forma que si un proceso espera datos no interferirá con los procesos que aún están realizando cálculos.

Encontramos dos formas de establecer la afinidad de CPU, una es utilizando el programa "taskset", que es parte de las utilerías del kernel de Linux. Este programa permite asignar a un proceso un CPU determinado, "encerrando" sus *threads* en el mismo CPU. La segunda forma es utilizando una modalidad de OpenMPI en la cual es posible asignar a cada proceso, utilizando su número de rango, un CPU en particular del cluster.

Este problema no aparece cuando se implementa el esquema híbrido de descomposición de dominios utilizando como solver el gradiente conjugado paralelizado con memoria compartida. Esto se debe a que el OpenMPI sólo se crea un proceso en cada computadora no se requiere comunicación con memoria compartida y *polling*.

6.8. Sistemas "grandes"

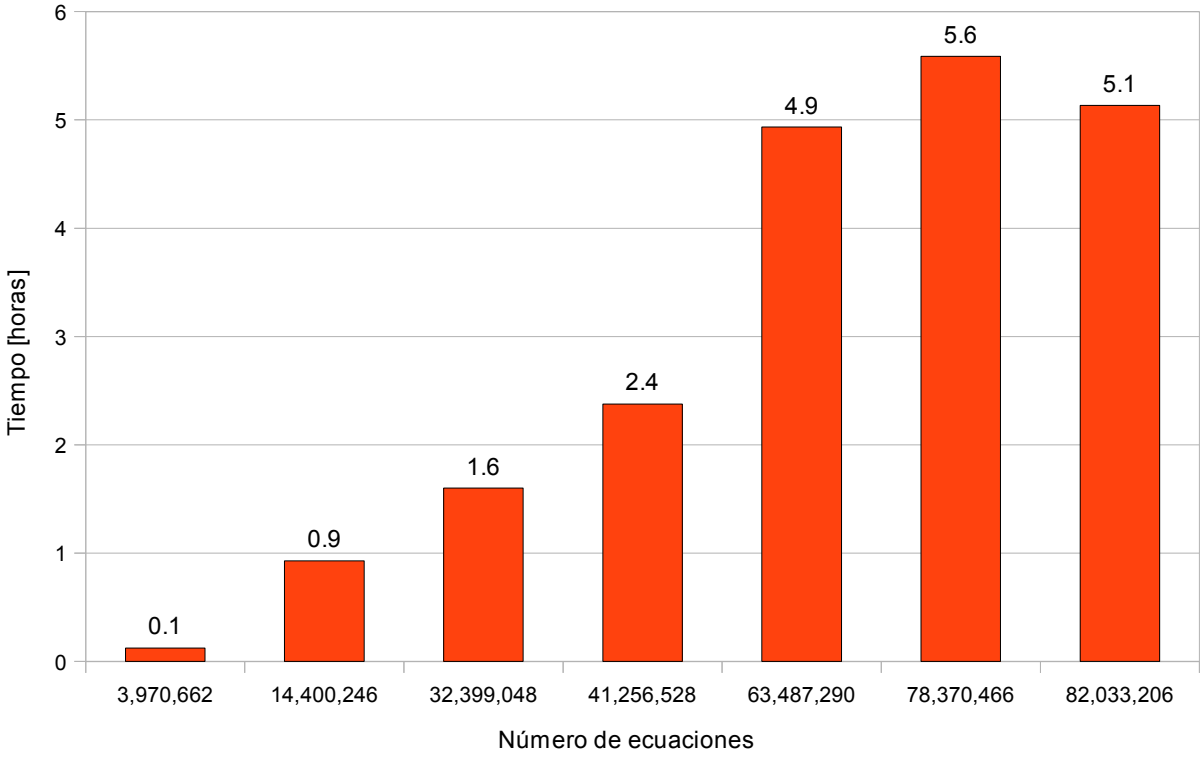
Los siguientes resultados se siguen refiriendo al problema de la figura 6.1. La tabla 6.2 muestra diversos resultados al generar mallas cada vez más refinadas, lo que significa sistemas de ecuaciones cada vez más grandes.

Los tiempos son dados en horas. La columna de la memoria se refiere a la memoria total utilizada por todos los nodos, tanto el nodo maestro (que es dónde se carga y distribuye el problema) como los nodos esclavo (dónde se resuelven los problemas individuales). Cada partición (o problema individual) se resolvió utilizando un CPU. El número de capas de traslape se eligió de forma eurística haciendo varias pruebas tratando de encontrar un "buen" valor que diese un tiempo de solución reducido.

Ecuaciones	Tiempo [h]	Memoria [GB]	Particiones	Traslape	PCs
3,970,662	0.12	17	52	12	13
14,400,246	0.93	47	52	10	13
32,399,048	1.60	110	60	17	15
41,256,528	2.38	142	60	15	15
63,487,290	4.93	215	84	17	29
78,370,466	5.59	271	100	20	29
82,033,206	5.13	285	100	20	29

Tabla 6.2. Resultados para sistemas de ecuaciones de diferente tamaño.

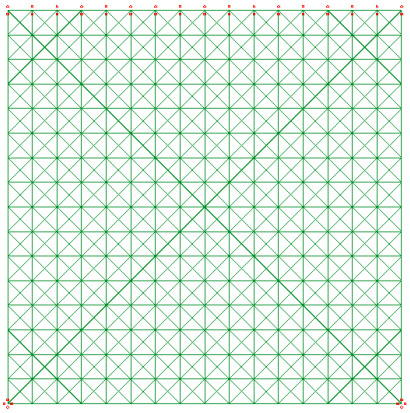
Los primeros cuatro resultados se obtuvieron utilizando el "Cluster 2". Para los siguientes tres casos se utilizaron en conjunto el "Cluster 1" y el "Cluster 2". La gráfica 6.8 permite comparar visualmente el tamaño del sistema de ecuaciones contra el tiempo de solución.



Gráfica 6.8. Comparativo de número de ecuaciones contra tiempo de solución del problema.

6.9. Un caso particular con malla estructurada

El programa que utilizamos para generar las mallas no nos permitió generar mallas más refinadas para el problema de la figura 6.1, por lo que creamos un problema más simple utilizando una malla estructurada regular, de esta forma sí pudimos generar una malla con más elementos, figura 6.6. Las condiciones de frontera para este caso son: las esquinas inferiores fijas y en toda la parte superior se impone un desplazamiento. El problema de 100’020,002 de ecuaciones se resolvió en 2 horas y 46 minutos (9964.9 segundos). Se utilizanron en conjunto el “Cluster 1” y el “Cluster 2”.



Elementos:	100’000,000
Nodos:	50’010,001
Particiones (4 por 29):	116
Traslape:	20
Tolerancia:	0.0001
Ecuaciones:	**100’020,002**
Memoria (maestro):	25.8 GB
Memoria (cada esclavo):	8.3 GB
Memoria (total):	260.0 GB
Ecuaciones por esclavo:	1’004,816

Figura 6.6. Malla simple para este ejemplo

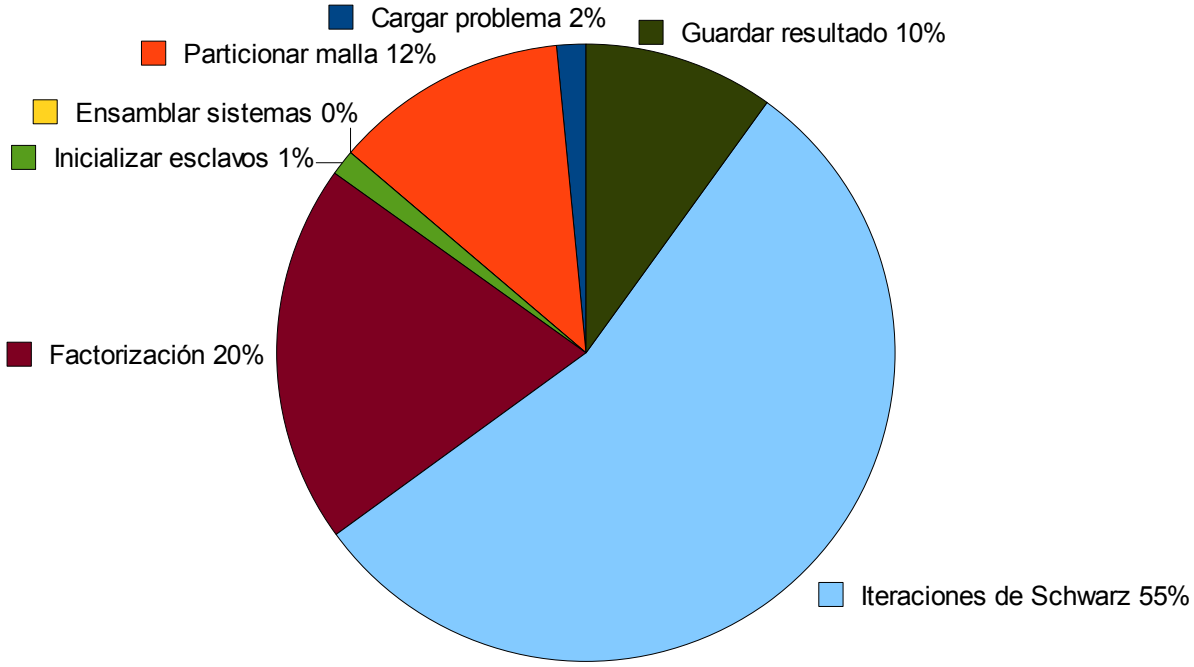
6.9.1. Distribución de tiempos del algoritmo

En la tabla 6.3 mostramos como se distribuyó el tiempo de solución del problema en cada una de las etapas del programa de cómputo.

Etapa	Tiempo [s]
Cargar problema	151.7
Particionar malla (116 particiones)	1220.3
Ensamblar sistemas	1.5
Inicializar esclavos	132.8
Factorización	1985.0
Iteraciones de Schwarz (3672 de 1.382s c/u)	5483.5
Guardar resultado	992.9
Total	**9964.9**

Tabla 6.3. Tiempos de ejecución por etapa.

Estos resultados se pueden comparar más facilmente en la gráfica 6.9.



Gráfica 6.9. Distribución de tiempos en la solución.

Visto de otro modo, podemos decir que en total se resolvieron 425,952 sistemas de 1’004,816 ecuaciones cada uno (en promedio), en un tiempo de 2 horas 46 min.

7. Conclusiones y trabajo futuro

El gradiente conjugado en paralelo con OpenMP presenta muy buenos resultados con hasta 8 procesadores, con más de 8 se degrada demasiado la eficiencia, esto es debido a que no es factible que todos los procesadores utilicen el bus de datos al mismo tiempo, lo que crea cuellos de botella, lo que da lugar tiempos de espera muy grandes.

El esquema de Schwarz funciona muy bien debido a que el tráfico de información entre los nodos es muy reducido. Se encontró además que se puede mejorar el tiempo de convergencia significativamente aumentando la cantidad de capas de traslape entre las particiones.

Es complicado evaluar la eficiencia de la descomposición de dominios para elementos finitos en problemas con mallas muy irregulares. Hemos encontrado que cuando las particiones queda con fronteras pequeñas se mejora la velocidad de convergencia. Pero, si una partición tiene más condiciones de frontera de Dirichlet, entonces converge más rápido, esto crea un desbalance de cargas, lo que alenta el tiempo de convergencia a una solución global. Para evitar este problema habría que implementar un mallador que sea más “inteligente”, para que éste pueda decidir donde sería mejor colocar las fronteras y mejorar el balance de cargas.

Encontramos que el utilizar el esquema de descomposición de dominios con un solver directo (factorización Cholesky simbólica) aunque costoso al momento de factorizar es muy eficiente al momento de resolver los sistemas en cada iteración. La gran desventaja de este método es que se utiliza mucha memoria. La cantidad de memoria está relacionada a la cantidad de conectividades que tenga la malla generada al discretizar con elemento finito.

Ésta es una alternativa para resolver problemas de gran escala cuando se dispone de la memoria suficiente para poder factorizar las matrices.

El programa desarrollado sólo permite hacer mallados con triángulos o tetraedros, estos son los tipos de elementos que aunque permiten mallar geometrías complejas, generan mallas con el mayor número de interconexiones. Creemos que, para geometrías sencillas, se lograría una mejor eficiencia en el programa si se utilizaran cuadriláteros o hexaedros con el fin de obtener matrices más ralas y por lo tanto más rápidas de resolver y con un menor consumo de memoria.

La parte más necesaria a mejorar es el particionador, actualmente se utilizan las rutinas de METIS, pero este tiene la desventaja de crear las particiones tratando de igualar la cantidad de elementos en cada una. Lo que se necesita es crear un particionador que divida las particiones considerando el número de conectividades de cada una. Al tener todas las particiones un número semejante de conectividades se generarán sistemas de ecuaciones ralos con un número semejante de elementos distintos de cero, lo que en este tipo de problemas en paralelo se traducirá en un mejor balance de carga.

Se está trabajando para extender el uso de estas rutinas a otros problemas de elemento finito, por ejemplo, para resolver las ecuaciones de Navier-Stokes con fluidos multi-fásicos en medios porosos.

Bibliografía

[Bote03] S. Botello, H. Esqueda, F. Gómez, M. Moreles, E. Oñate. Módulo de Aplicaciones del Método de los Elementos Finitos, MEFI 1.0. Guanajuato, México. 2003.

[Chap08] B. Chapman, G. Jost, R. van der Pas. Using OpenMP: Portable Shared Memory Parallel Programming. The MIT Press, 2008.

[DAze93] E. F. D'Azevedo, V. L. Eijkhout, C. H. Romine. Conjugate Gradient Algorithms with Reduced Synchronization Overhead on Distributed Memory Multiprocessors. Lapack Working Note 56. 1993.

[Dohr03] Dohrmann C. R. A Preconditioner for Substructuring Based on Constrained Energy Minimization. SIAM Journal of Scientific Computing. Volume 25-1, pp. 246-258, 2003.

[Drep07] U. Drepper. What Every Programmer Should Know About Memory. Red Hat, Inc. 2007.

[Esch97] H. Eschenauer, N. Olhoff, W. Schnell. Applied Structural Mechanics. Springer, 1997.

[Farh01] C. Farhat, M. Lesoinne, P. Le Tallec, K. Pierson, D. Rixen. FETI-DP: A dual-primal unified FETI method - part I: A faster alternative to the two-level FETI method. International Journal of Numerical Methods Engeneering, Vol 50. pp1523–1544, 2001.

[Flyn72] M. Flynn, Some Computer Organizations and Their Effectiveness, IEEE Transactions on Computers., Vol. C-21, pp. 948, 1972.

[Gall90] K. A. Gallivan, M. T. Heath, E. Ng, J. M. Ortega, B. W. Peyton, R. J. Plemmons, C. H. Romine, A. H. Sameh, R. G. Voigt, Parallel Algorithms for Matrix Computations, SIAM, 1990.

[Geor81] A. George, J. W. H. Liu. Computer solution of large sparse positive definite systems. Prentice-Hall, 1981.

[Geor89] A. George, J. W. H. Liu. The evolution of the minimum degree ordering algorithm. SIAM Review Vol 31-1, pp 1-19, 1989.

[Golu96] G. H. Golub, C. F. Van Loan. Matrix Computations. Third edidion. The Johns Hopkins University Press, 1996.

[Heat91] M T. Heath, E. Ng, B. W. Peyton. Parallel Algorithms for Sparse Linear Systems. SIAM Review, Vol. 33, No. 3, pp. 420-460, 1991.

[Kary99] G. Karypis, V. Kumar. A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM Journal on Scientific Computing, Vol. 20-1, pp. 359-392, 1999.

[Kimn06a] J. H. Kimn, M. Sarkis. OBDD: Overlapping balancing domain decomposition methods and generalizations to the Helmholtz equation. Proceedings of the 16th International Conference on Domain Decomposition Methods, 2005.

[Kimn06b] J. H. Kimn, B. Bourdin. Numerical Implementation of Overlapping Balancing Domain Decomposition Methods on Unstructured Meshes. Proceedings of the 16th International Conference on Domain Decomposition Methods, 2005.

[Lipt77] R. J. Lipton, D. J. Rose, R. E. Tarjan. Generalized Nested Dissection, Computer Science Department, Stanford University, 1997.

[Lipt79] R. J. Lipton, R. E. Tarjan. A Separator Theorem for Planar Graphs. SIAM Journal on Appliend Mathematics, Vol. 36-2, 1979.

[Li05] J. Li, O. Widlund. FETI-DP, BDDC, and block Cholesky methods. International Journal for Numerical Methods in Engineering. Vol. 66-2, pp. 250-271, 2006.

[Mand93] J. Mandel. Balancing Domain Decomposition. Communications on Numerical Methods in Engineering. Vol. 9 pp 233-241, 1993.

[Mand02] J. Mandel, C. R. Dohrmann. Convergence of a balancing domain decomposition by constraints and energy minimization. Journal of Numerical Linear Algebra with Applications, Vol. 10-7, pp. 639 - 659.

[Mand05] J. Mandel, C. R. Dohrmann, R. Tezaur. An algebraic theory for primal and dual substructuring methods by constraints. Elsevier Science Publishers. Journal of Applied Numerical Mathematics, Vol. 54-2, 2005.

[MPIF08] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, Version 2.1. University of Tennessee, 2008.

[Noce06] J. Nocedal, S. J. Wright. Numerical Optimization, Springer, 2006.

[Piss84] S. Pissanetzky. Sparse Matrix Technology. Academic Press, 1984.

[Quar00] A. Quarteroni, R. Sacco, F. Saleri. Numerical Mathematics. Springer, 2000.

[Saad03] Y. Saad. Iterative Methods for Sparse Linear Systems. SIAM, 2003.

[Smit96] B. F. Smith, P. E. Bjorstad, W. D. Gropp. Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge Univeristy Press, 1996.

[Ster95] T. Sterling, D. J. Becker, D. Savarese, J. E. Dorband, U. A. Ranawake, C. V. Packer. BEOWULF: A Parallel Workstation For Scientific Computation. Proceedings of the 24th International Conference on Parallel Processing, 1995.

[Tose05] A. Toselli, O. Widlund. Domain Decomposition Methods - Algorithms and Theory. Springer, 2005.

[**Widl08]** O. B. Widlund. The Development of Coarse Spaces for Domain Decomposition Algorithms. Proceedings of the 18th International Conference on Domain Decomposition, 2008.

[Wulf 95] W. A. Wulf , S. A. Mckee. Hitting the Memory Wall: Implications of the Obvious. Computer Architecture News, 23(1):20-24, March 1995.

[Yann81] M. Yannakakis. Computing the minimum fill-in is NP-complete. SIAM Journal on Algebraic Discrete Methods, Volume 2, Issue 1, pp 77-79, March, 1981.

[Zien05] O.C. Zienkiewicz, R.L. Taylor, J.Z. Zhu, The Finite Element Method: Its Basis and Fundamentals. Sixth edition, 2005.

Apéndice A. Guía para hacer pruebas

Para probar el proyecto es necesario utilizar el programa de pre y postproceso GiD, desarrollado por CIMNE (International Center for Numerical Methods in Engineering), para generar el mallado y al final para visualizar los resultados.

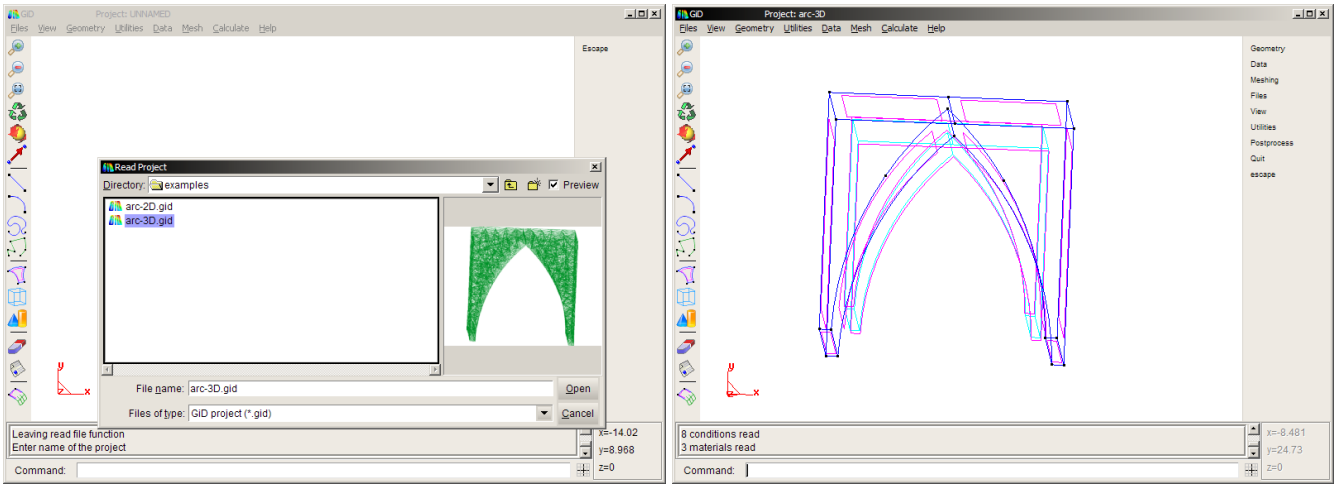
Los resultados se pueden ver de forma combinada, es decir, los resultados de todas las particiones mezclados o con los resultados de las particiones de forma independiente.

Las instrucciones funcionan haciendo pruebas tanto en Windows como en Linux.

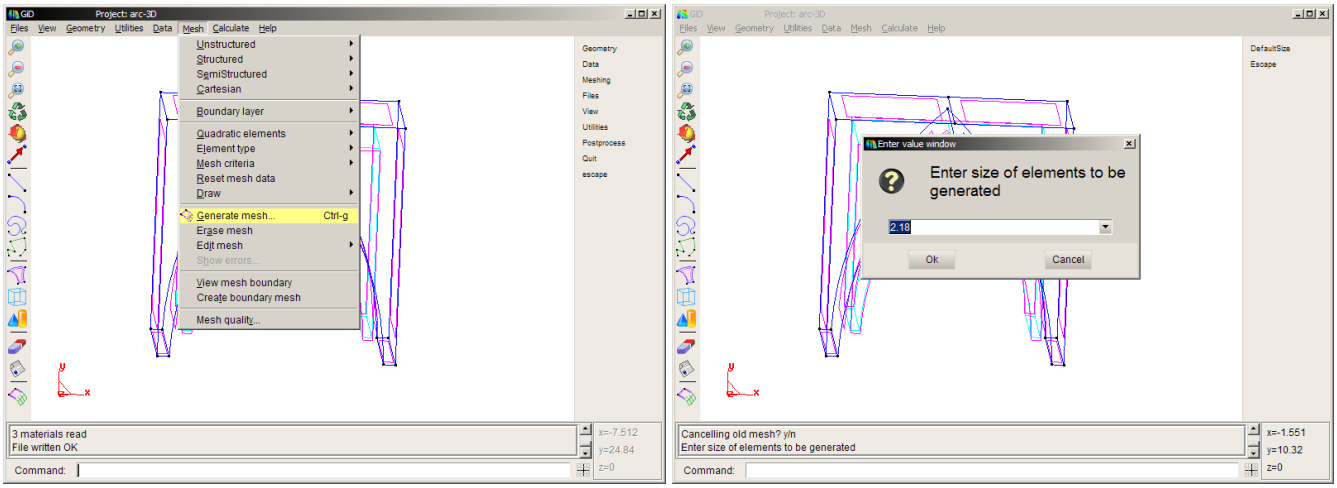
Forma combinada

1. Abrir el GiD.

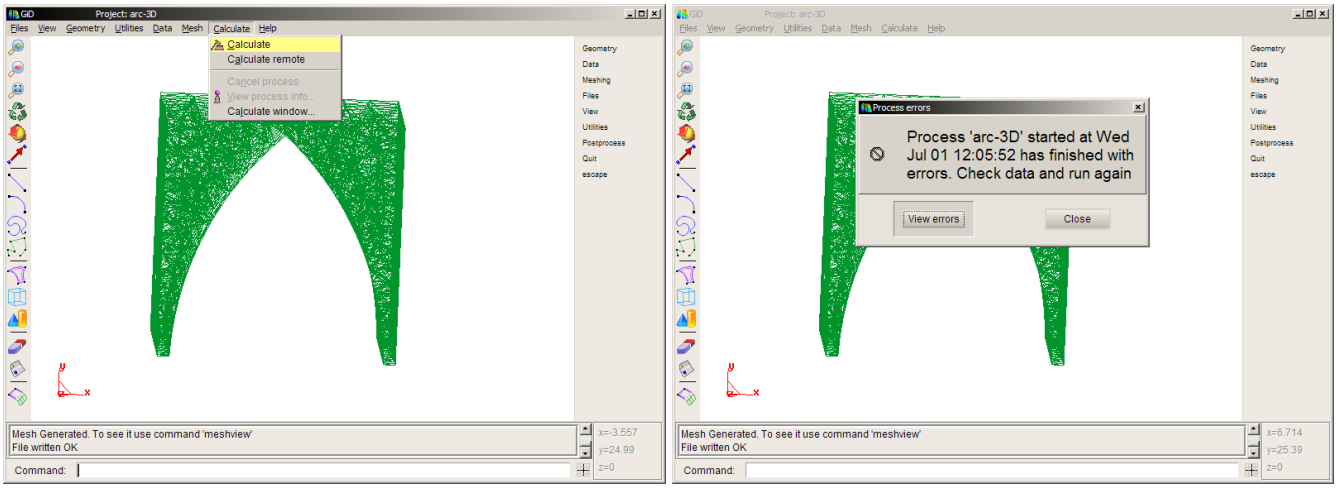
2. Desde el menú [Files/Open] seleccionar alguno de los directorios contenidos en "Mecanic4/examples", por ejemplo el directorio "Mecanic4/examples/arc-3D.gid"



3. Generar una malla con el menú [Mesh/Generate mesh]. Se puede cambiar la densidad del mallado ajustando el parámetro del tamaño del elemento.



4. Ejecutar el menú [Calculate/Calculate], esto generará un archivo "Mecanic4/examples/arc-3D.gid/arc-3D.dat" que contendrá los datos de entrada del programa. El sistema generará un mensaje de error, esto es porque para usar MPI es necesario ejecutar el programa manualmente.

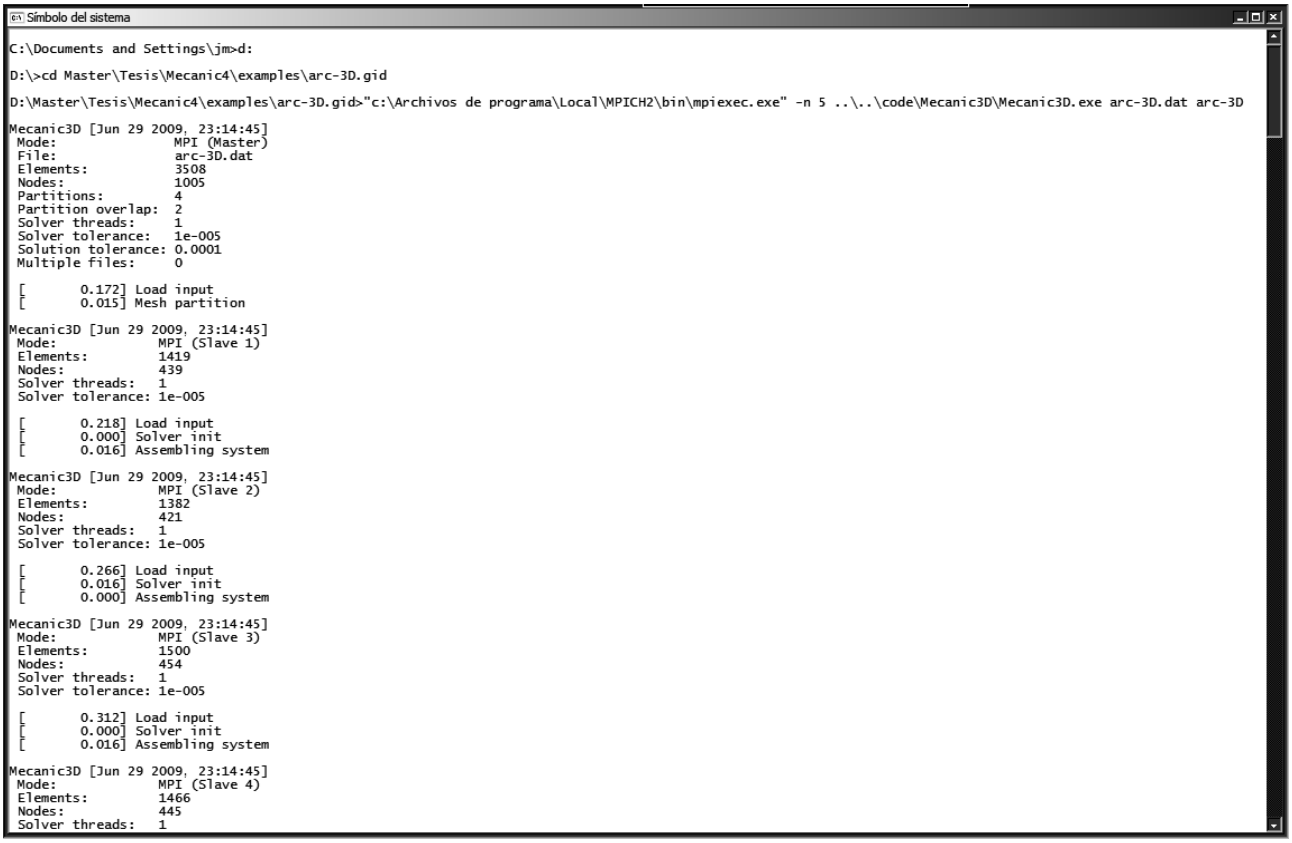


5. Abrir una consola de comandos.

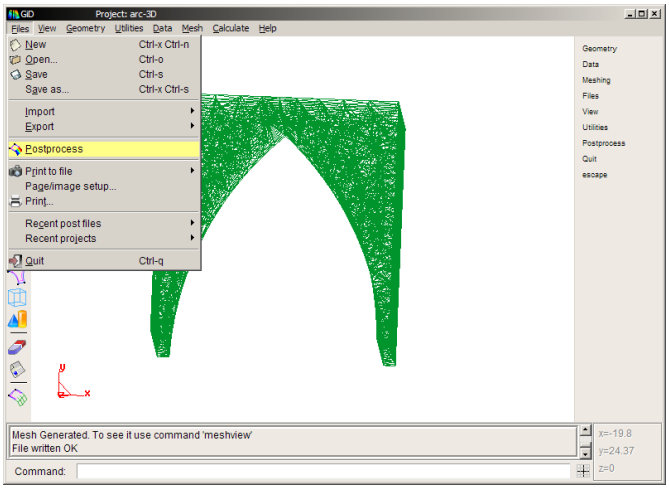
5.1. Cambiarse al directorio "Mecanic4/examples/arc-3D.gid"

5.2. Ejecutar "mpirun -n 5 ../../code/Mecanic3D arc-3D.dat arc-3D" (esto dividirá el problema en 4 particiones)

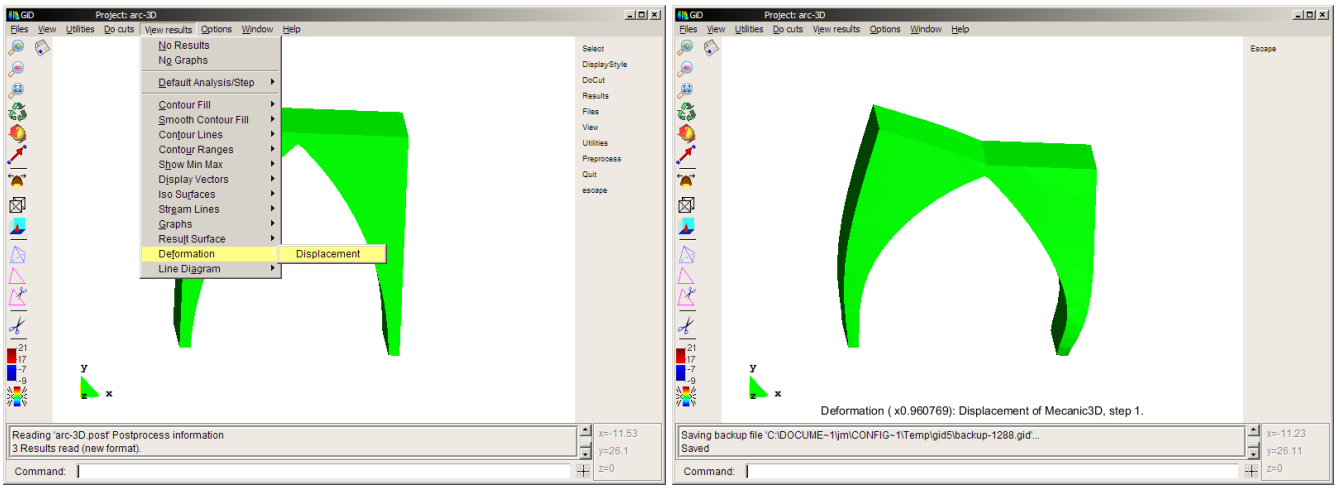
5.3. Al terminar se generará un archivo "Mecanic4/examples/arc-3D.gid/arc-3D.post.res" que contendrá los resultados de desplazamientos, deformaciones y tensiones. Los valores que reporta el programa entre corchetes son el tiempo que tardó en ejecutar cada tarea en segundos.



6. Regresar al GiD y seleccionar el menú [Files/Postprocess]



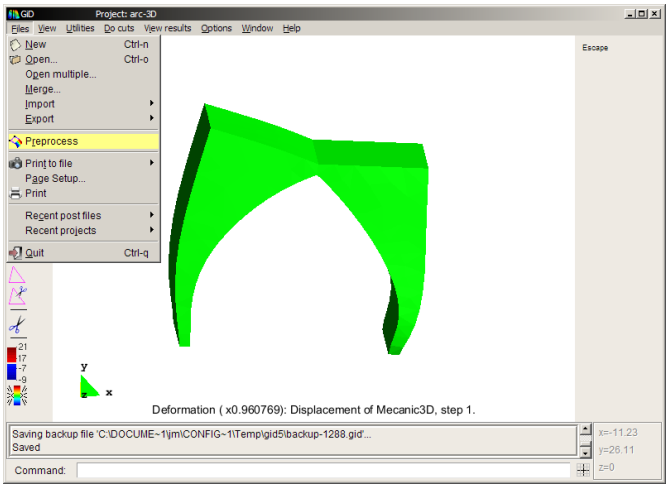
7. Para ver la deformación resultante, en el menú [View results/Deformation/Displacements]



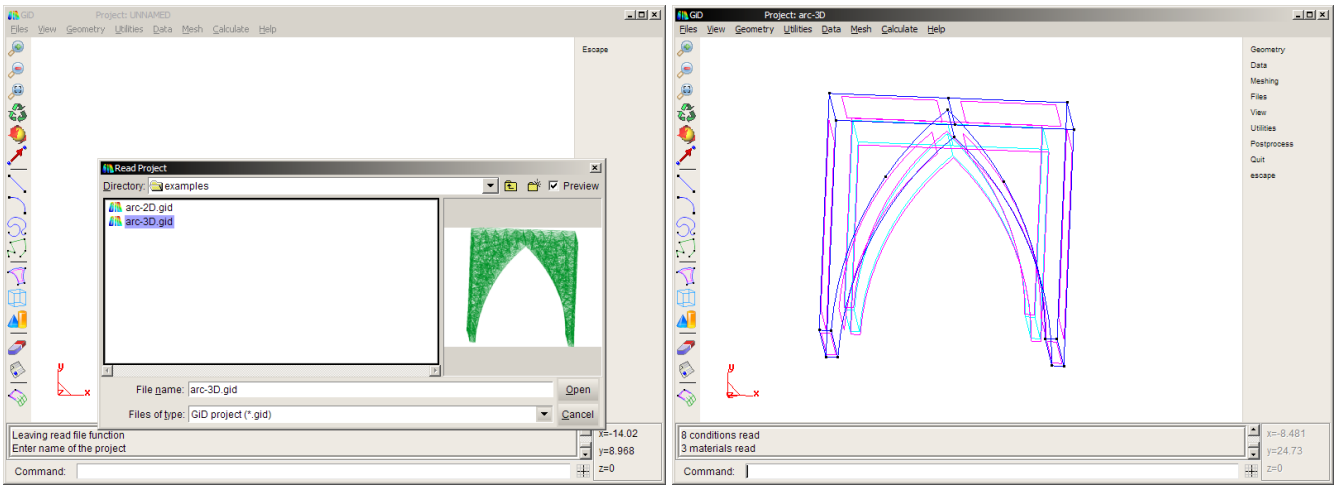
Particiones independientes

Para ver los resultados de cada partición de forma independiente:

1. En caso de que ya esté abierto GiD, cambiar a modo de preproceso, abriendo el menú [Files/Preprocess].



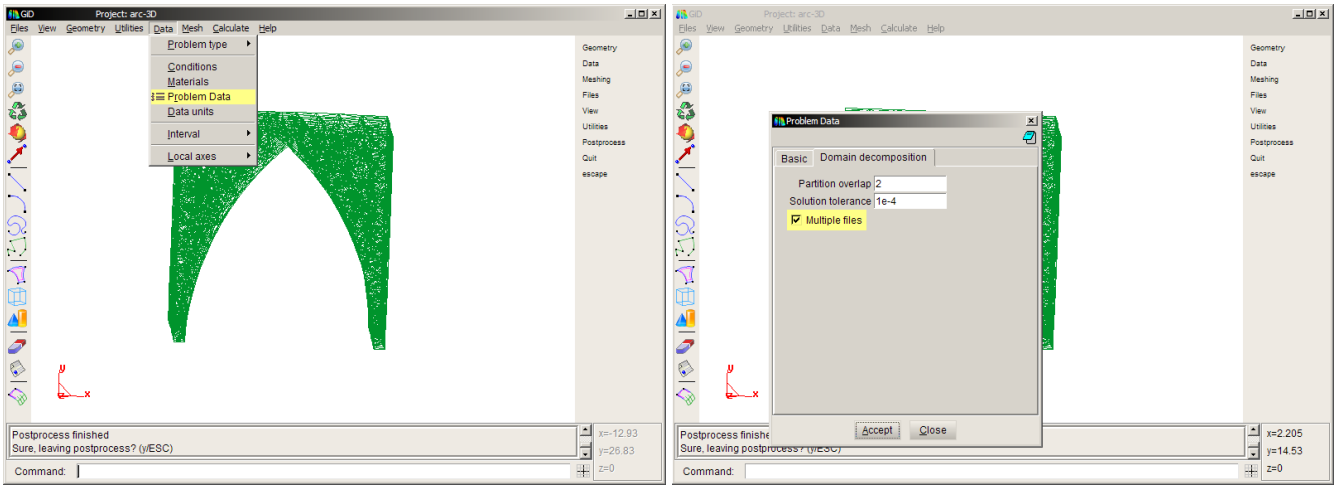
2. Desde el menú [Files/Open] seleccionar alguno de los directorios contenidos en "Mecanic4/examples", por ejemplo el directorio "Mecanic4/examples/arc-3D.gid"



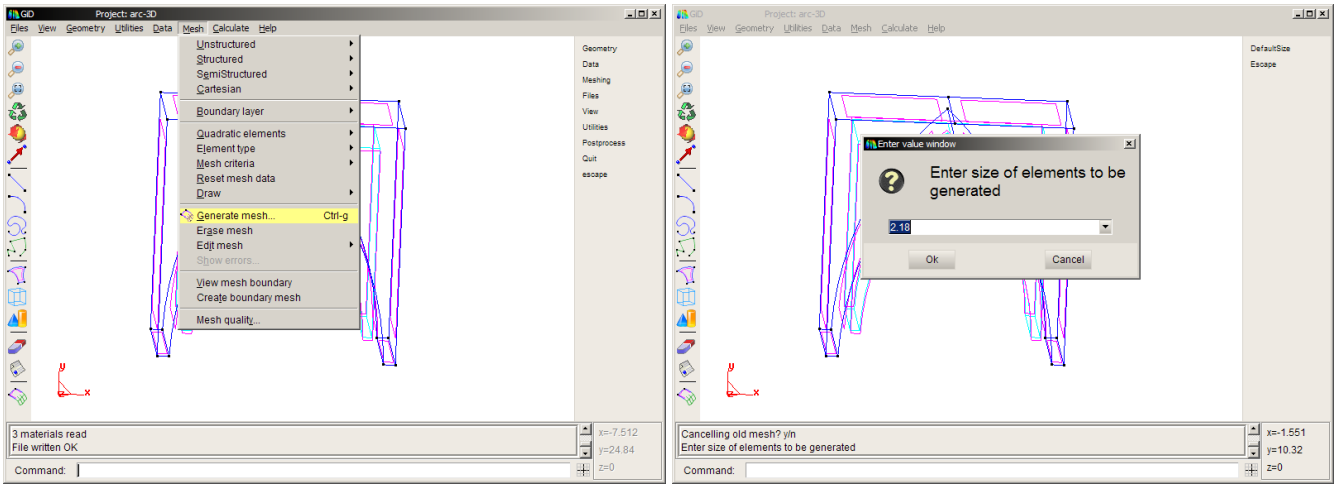
3. Abrir el menú [Data/Problem data], aquí se pueden modificar los parámetros de ejecución del programa Mecanic3D.

3.1. Seleccionar [Domain decomposition]

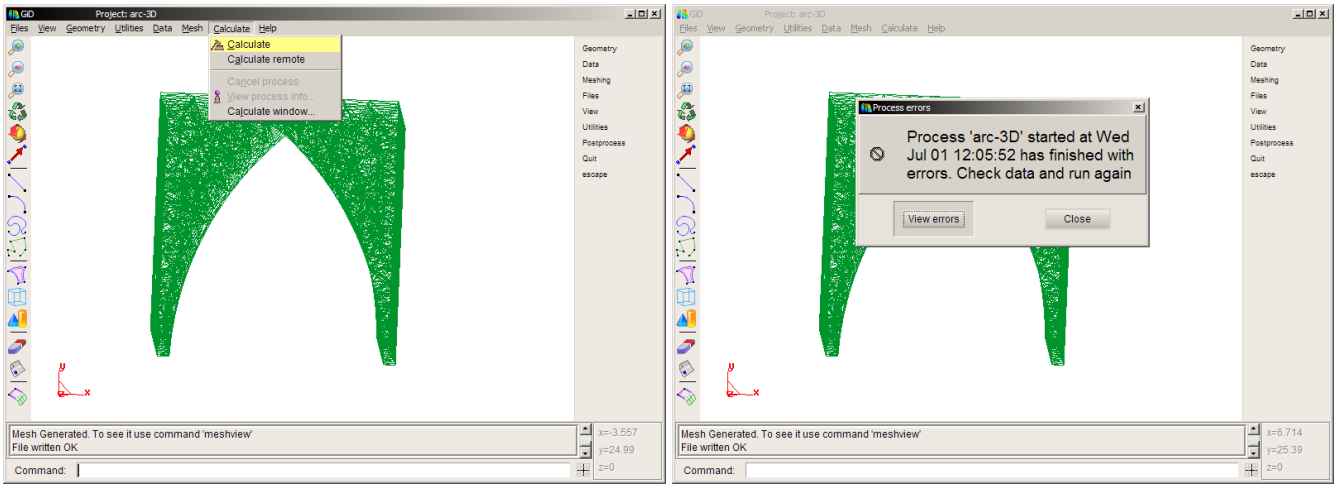
3.2. Activar la opción [Multiple files] y presionar [Accept]



4. Generar una malla con el menú [Mesh/Generate mesh]. Se puede cambiar la densidad del mallado ajustando el parámetro del tamaño del elemento.



5. Ejecutar el menú [Calculate/Calculate], esto generará un archivo "Mecanic4/examples/arc-3D.gid/arc-3D.dat" que contendrá los datos de entrada del programa. El sistema generará un mensaje de error, esto es porque para usar MPI es necesario ejecutar el programa manualmente.

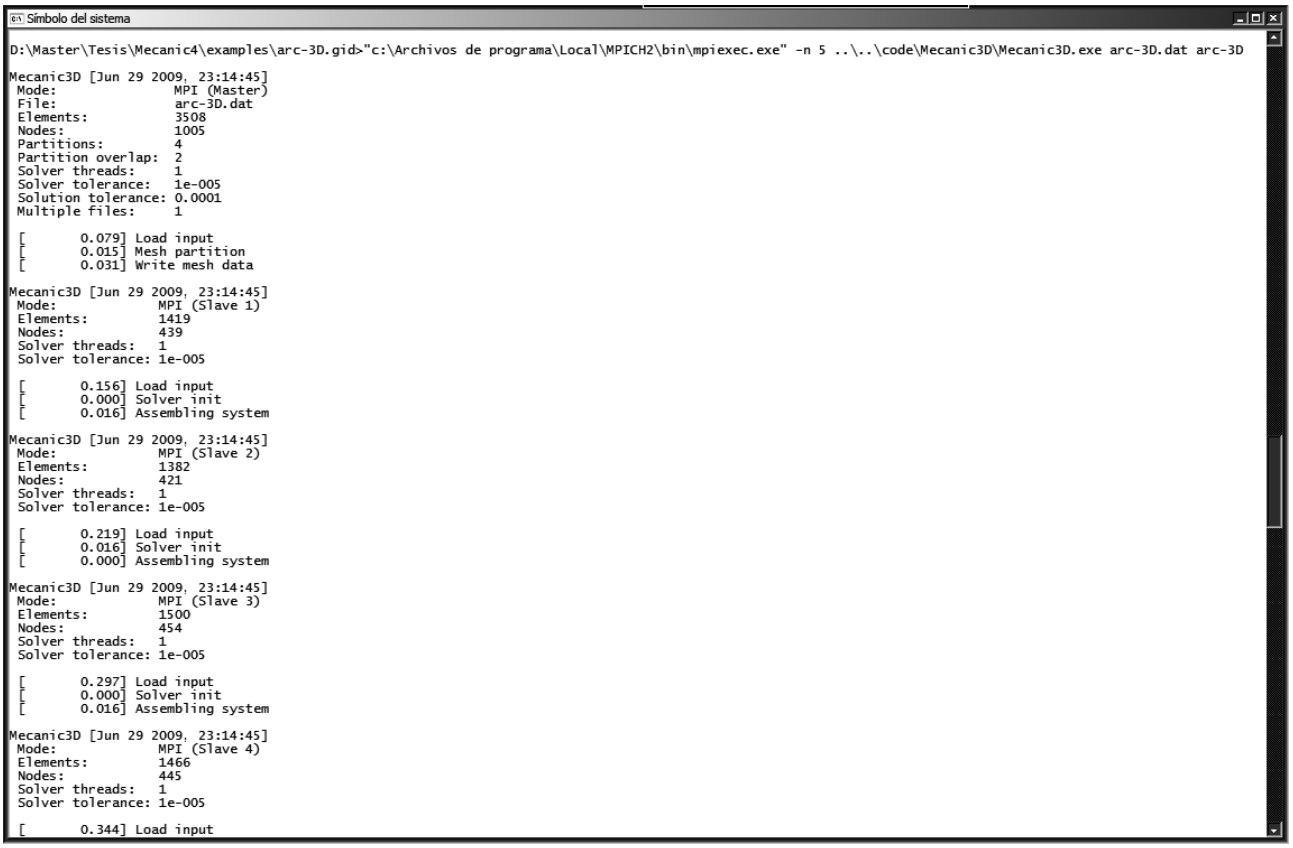


6. Abrir una consola de comandos.

6.1 Cambiarse al directorio "Mecanic4/examples/arc-3D.gid"

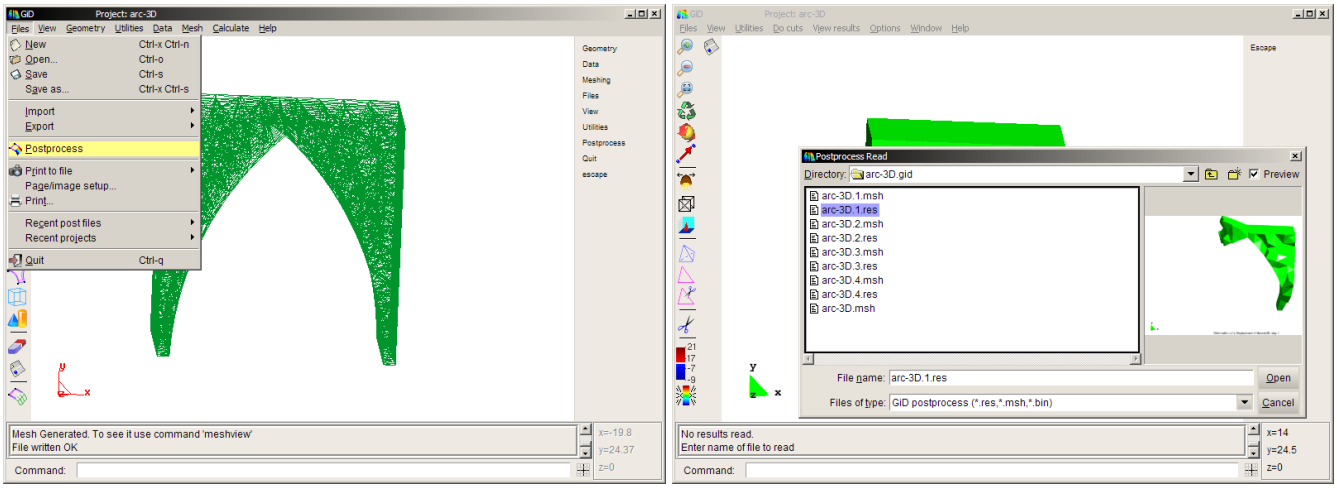
6.2 Ejecutar "mpirun -n 5 ../../code/Mecanic3D arc-3D.dat arc-3D" (esto dividirá el problema en 4 particiones)

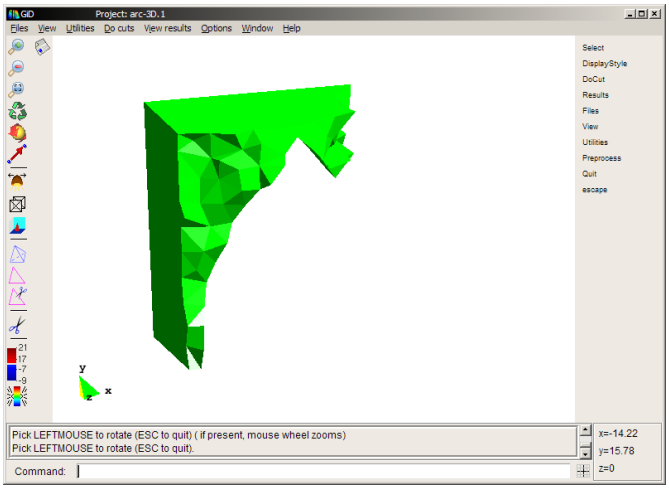
6.3 Al terminar se generarán varios archivos "Mecanic4/examples/arc-3D.gid/arc-3D.*.res" que contendrán los resultados de desplazamientos, deformaciones y tensiones para cada partición. Los valores que reporta el programa entre corchetes son el tiempo que tardó en ejecutar cada tarea en segundos.



7. Regresar al GiD y seleccionar el menú [Files/Postprocess]

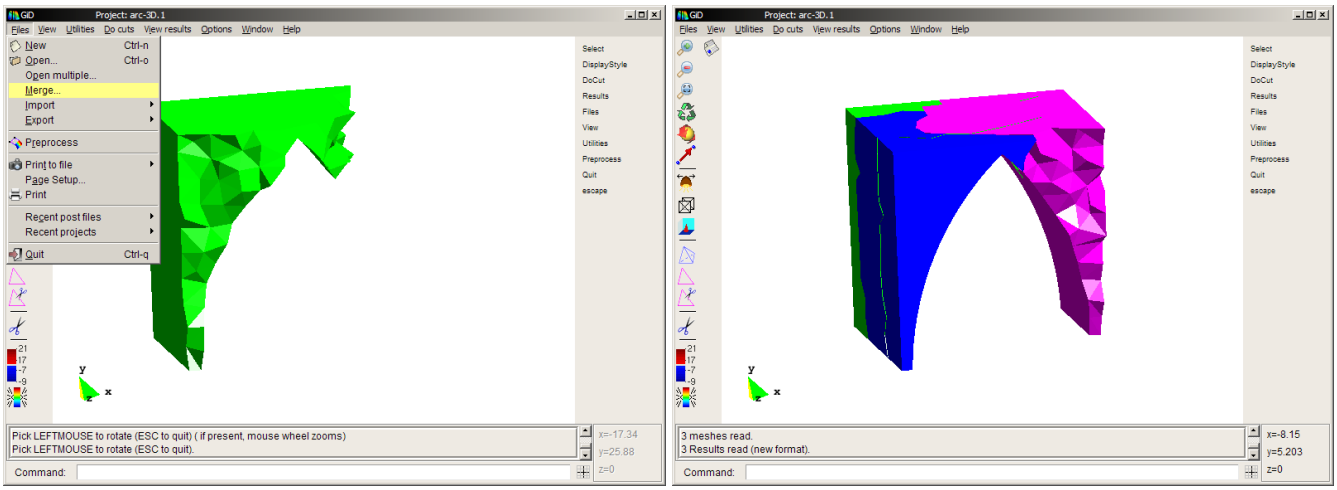
7.1. Abrir con el menú [Files/Open] el archivo "Mecanic4/examples/arc-3D.gid/arc-3D.1.res"



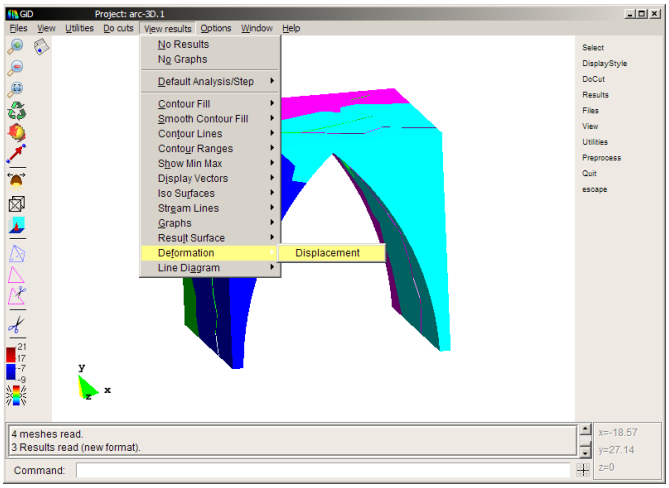


8. Para ver las otras particiones, regresar al punto 7.1 y elegir otro archivo "Mecanic4/examples/arc-3D.gid/arc-3D.*.res"

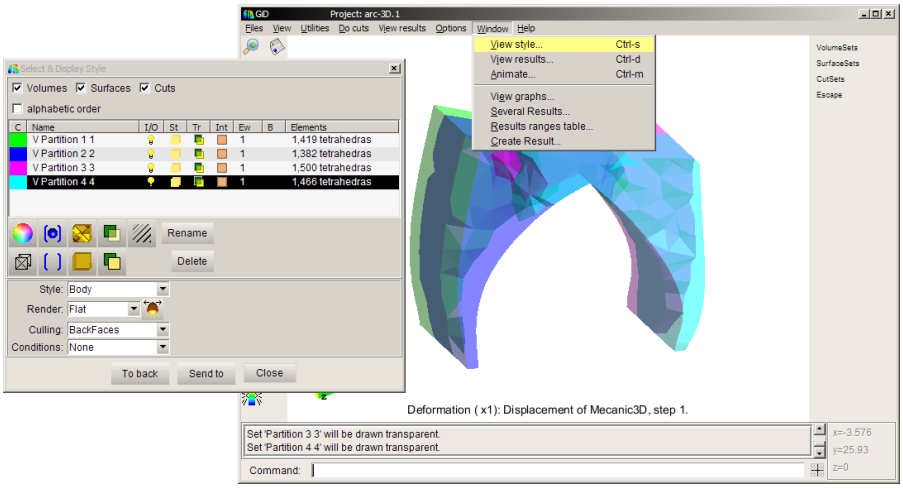
9. Es posible mezclar las particiones, para esto abrir [Files/Merge] y seleccionar otro de los archivos "Mecanic4/examples/arc-3D.gid/arc-3D.*.res"



10. Para ver la deformación resultante, en el menú [View results/Deformation/Displacements]



11. Se puede elegir cambiar el modo de visualización abriendo el menú [Window/View style].



Apéndice B. Ejemplo de ejecución en un cluster

A continuación describimos un ejemplo de los comandos necesarios para, una vez teniendo un archivo de datos, ejecutar el programa en un cluster.

Ejemplo del comando para ejecutar el programa en el cluster es

```
mpirun -n 57 -hostfile hostfile.txt -rf rankfile.txt --mca mpi_yield_when_idle 1 Mecanic3D arc-3D.dat arc-3D
```

Ejemplo del *hostsfile*

```
el-insurgente.local slots=1 max_slots=1
compute-0-17.local slots=4 max_slots=4
compute-0-18.local slots=4 max_slots=4
compute-0-19.local slots=4 max_slots=4
compute-0-20.local slots=4 max_slots=4
compute-0-21.local slots=4 max_slots=4
compute-0-22.local slots=4 max_slots=4
compute-0-23.local slots=4 max_slots=4
compute-0-24.local slots=4 max_slots=4
compute-0-25.local slots=4 max_slots=4
compute-0-26.local slots=4 max_slots=4
compute-0-27.local slots=4 max_slots=4
compute-0-28.local slots=4 max_slots=4
compute-0-31.local slots=4 max_slots=4
compute-0-32.local slots=4 max_slots=4
```

Ejemplo del *rankfile*

```
rank 0=el-insurgente.local slot=0
rank 1=compute-0-17.local slot=0
rank 2=compute-0-18.local slot=0
rank 3=compute-0-19.local slot=0
rank 4=compute-0-20.local slot=0
rank 5=compute-0-21.local slot=0
rank 6=compute-0-22.local slot=0
rank 7=compute-0-23.local slot=0
rank 8=compute-0-24.local slot=0
rank 9=compute-0-25.local slot=0
rank 10=compute-0-26.local slot=0
rank 11=compute-0-27.local slot=0
rank 12=compute-0-28.local slot=0
rank 13=compute-0-31.local slot=0
rank 14=compute-0-32.local slot=0
rank 15=compute-0-17.local slot=1
rank 16=compute-0-18.local slot=1
...
rank 56=compute-0-32.local slot=3
```

Apéndice C. Generación de mallas grandes

No es factible crear mallas de decenas de millones de elementos usando el GiD en modo interactivo, para estos casos se tiene que ejecutar en una consola de comandos en modo batch. El siguiente script llama a GiD en este modo y ejecuta el archivo batch (bch) que se muestra más abajo.

```
#!/bin/sh
GIDDEFAULT=/Applications/GiD-9.2.2b.app/Contents/MacOS
GIDDEFAULTTCL=$GIDDEFAULT/scripts
TCL_LIBRARY=$GIDDEFAULT/scripts
TK_LIBRARY=$GIDDEFAULT/scripts
DYLIB_LIBRARY_PATH=$GIDDEFAULT/lib
DYLD_LIBRARY_PATH=$GIDDEFAULT/lib

export GIDDEFAULT
export GIDDEFAULTTCL
export TCL_LIBRARY
export TK_LIBRARY
export DYLIB_LIBRARY_PATH
export DYLD_LIBRARY_PATH

/Applications/GiD-9.2.2b.app/Contents/MacOS/gid.exe -n -b  mesh.arc-3D.sstruct.bch
"../../examples/arc-3D.gid"
```

Script para llamer a GiD

```
*****OUTPUTFILENAME "arc-3D.log"
Meshing Reset y escape escape escape
Meshing SemiStructured Volumes 400 InvertSelection escape escape escape
Meshing ElemType Tetrahedra InvertSelection escape escape escape
Meshing Generate 0.220 escape escape escape
Files WriteCalcFile "arc-3D.dat" escape escape escape
```

Archivo bch con los comandos de GiD para la generación de mallas

El archivo batch anterior tiene dos parámetros, uno es la cantidad de divisiones y otro el tamaño del elemento. Variando estos dos valores se obtienen los siguientes resultados:

Divisiones	Size	Elements	Nodes
400	0.200	13077600	2334622
400	0.175	16644000	2951360
400	0.150	22416000	3944637
400	0.125	31996800	5584727