Capítulo 3
Implementación del SERBAPA

3.1.- Modificación del código secuencial

Se comenzó modificando el programa BETIS del libro “Boundary Element Method” de Federico París y José Cañas (1997), para que aceptara nodos y elementos con numeraciones no consecutivas y con ello pudiera tener huecos el dominio. Se generó este programa para hacer pruebas y posteriormente implementarlo en el SERBA, segundo programa del libro anterior, pero, en lugar de calcular flujos y potenciales, calcula tensiones y desplazamientos en un dominio. Los archivos de entrada de nuestro programa están diseñados de forma que puedan siempre ser procesados por los programas originales de calculo (BETIS y SERBA).

Para poder procesar huecos en el dominio se creo una “matriz de conectividades” que almacena para cada elemento su nodo de entrada y de salida.

También se reprogramó para que reportara los valores de las

deformaciones en todo el dominio, ya que era necesario para posteriormente calcular la derivada topológica. El programa original tan solo daba los valores de tensión normal y tangencial en coordenadas locales en el contorno por lo que hubo que aplicarle una matriz de transformación para obtener coordenadas globales. Para ello se calculó en primer lugar la deformación de todos los elementos del contorno y así obtener ε_{22}. Con ello podemos calcular σ_{22} con las siguientes expresiones:

$$\varepsilon_{22}=(u_2-u_1)*n_x/L+(v_2-v_1)*n_y/L \qquad (1)$$

$$\sigma_{22}=\varepsilon_{22}*E+\sigma_{11}*pois \qquad (2)$$

donde n_x y n_y son las componentes del vector normal al elemento, **L** es la longitud de dicho elemento, **pois** es el valor del coeficiente poisson y **E** el coeficiente de elasticidad del material.

Con estos datos ya podiamos calular el resto de valores de las componentes del tensor de esfuerzos.

Además se le ha incorporado la posibilidad de ver dibujado el contorno y puntos internos antes y después de aplicar los esfuerzos para ver el dominio deformado, y los diagramas de esfuerzos utilizando un triangulador especial (Meshsuite) que tiene la capacidad de distinguir si hay agujeros en el dominio. Todo ello es visible en el programa Matlab, pero no en Octave.

Un esquema de trabajo del programa SERBA modificado podemos encontrarlo en la sección Anexo 7.2.

Probamos varios ejemplos como son el de un tubo sometido a presión interior, una placa sometida a tracción y el de una viga empotrada a tracción, flexión y torsión y sus resultados fueron comparados con las soluciones analíticas de la bibliografía.

Para comprobar la precisión de la aplicación se presenta a continuación

a modo de ejemplo el problema del tubo de pared gruesa donde se pueden comprobar los valores de tensión radial y circunferencial (tablas 3.1 y 3.2) con una presión interior de 100 Kg/cm2 y con unos radios interior y exterior de 5 y 10 cm respectivamente y una malla de 200 puntos de contorno y 356 puntos internos. La ecuación teórica de dichas tensiones es:

$$\sigma_{rr,\theta\theta}=p*R_i^2/(R_e^2-R_i^2)*(1\pm R_e^2/r^2) \qquad (3)$$

Teórico	SERBA
-100	-93,83118
-76,86	-77,57351
-59,26	-59,83365
-45,56	-46,08803
-34,69	-35,10874
-25,93	-26,25068
-18,75	-19,00249
-12,8	-12,99304
-7,82	-7,94279
-3,6	-3,77426
0	-0,96067

Tabla 3.1

Teórico	SERBA
200	200,49
210,8	211,02
223,46	223,3
238,41	238,27
256,25	256,05
277,78	277,48
304,08	303,62
336,69	336,02
377,78	376,44
430,58	430,38
500	499,88

Tabla 3.2

Como ejemplos de resultados gráficos de la aplicación tenemos el de la viga empotrada sometida a flexión simple representado en la figura 3.1

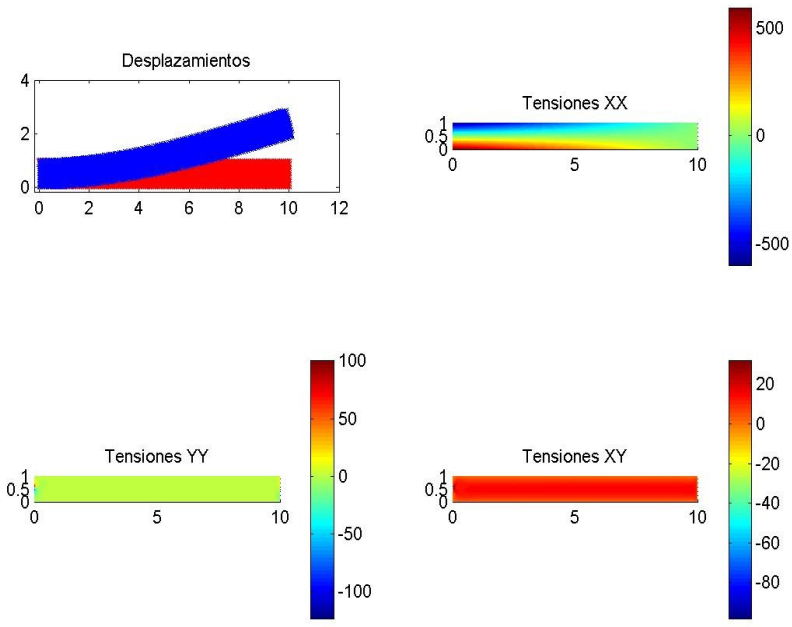

Figura 3.1

3.2.- Paralelización del SERBA

Posteriormente se llevo a cabo la paralelización de dicho programa al que se le dio el nombre de SERBAPA, con las librerías MPI. Esto era necesario para poder hacer cálculos con un elevado número de nodos, que era el caso que posteriormente se iba a tener con el uso de la derivada topológica.

Se utilizó el nodo0 (computadora desde el que se ejecuta el programa) como nodo principal, el cual abre el archivo de escritura, recibe los cálculos del resto de nodos, resuelve la matriz principal y escribe los resultados.

El número de nodos de contorno se divide entre las diferentes máquinas del cluster. Cada una de ellas lee el archivo de entrada, calcula las integrales de los puntos del contorno que le corresponden y manda los resultados al nodo0. No escriben ni resuelven la matriz. Una vez resuelta la matriz por el nodo0, este manda los resultados a todas las computadoras para que de nuevo, repartan el total de puntos, esta vez internos, en cada una de ellas y resuelvan el dominio con sus respectivas tensiones y deformaciones.

Para el paso de datos se utilizaron básicamente los comandos MPI_Send y MPI_Recive aunque podría mejorarse mucho utilizando la función MPI_Bcast así como otras funciones básicas del MPI. Otras rutinas utilizadas fueron MPI_Barrier para sincronizar los nodos y MPI_Wtime para calcular los tiempos de cálculo. Y por supuesto las necesarias para inicializar el entorno de trabajo MPI que son MPI_Init, MPI_Comm_Size, MPI_Comm_Rank y MPI_Comm_Finalize.

Para comprobar los resultados se analizó un dominio de dimensiones 10*10 met, con una E=210000 Mpa y una Poisson=0.3. Dicha placa se dividió en 20, 40, 60, 80 y 100 divisiones por cada lado y con un porcentaje del 5% de área hueca. Esto dio unos tiempos de procesamiento que se exponen a continuación en la tabla 3.3:

	Nodos cluster	Divisiones	Integ contorno	Resolvedor	Ptos internos	Total
SECUENCIAL	1	20	0,12	0,14	0,60	0,88
	1	40	1,18	6,20	7,65	15,12
	1	60	5,56	67,94	37,35	111,04
	1	80	12,27	227,29	100,62	340,49
	1	100	28,86	911,89	240,42	1181,64
PARALELIZADO	3	20	11,52	0,16	0,31	13,12
	4	20	22,54	0,16	0,29	24,12
	5	20	33,55	0,17	0,21	35,12
	6	20	44,63	0,16	0,17	46,07
	3	40	12,14	6,94	3,90	23,66
	4	40	23,12	6,84	3,63	34,25
	5	40	35,45	7,05	2,72	46,15
	6	40	44,99	7,03	2,18	55,40
	3	60	14,58	79,64	19,39	114,32
	4	60	27,02	79,48	17,32	124,93
	5	60	36,04	80,90	12,98	130,95
	6	60	47,13	78,95	10,41	137,77
	3	80	19,13	250,27	51,38	321,54
	4	80	28,87	253,99	47,17	331,22
	5	80	38,54	254,81	35,36	329,95
	6	80	48,93	267,67	28,33	346,37

Tabla 3.3: Comparación de resultados

Solo están reportados los tiempo del calculo de integrales del contorno, el tiempo del resolvedor y el de calculo de integrales de puntos internos ya que el resto de tiempos es despreciable frente a estos.

Como puede observarse en los datos de la tabla adjunta, hay un problema en el cálculo de las integrales de los puntos del contorno, ya que el tiempo se incrementa con el número de nodos, cuando debería, disminuir, o a lo sumo, quedarse igual.

Después de realizar una profunda investigación del programa se vio que era un problema de tiempo de paso de información de cada uno de los nodos “calculadores” al nodo “administrador”. Dicha información era la matriz "coef", que contiene los cálculos de todos los puntos del contorno. Estaba programado para que cada nodo calculara solo una parte de los nodos, pero pasara al nodo0 la matriz entera, con ceros en las posiciones no calculadas. Cada nodo le mandaba al *nodo0 (del cluster)* la lista de nodos que calculaba y la matriz *coef* entera. El *nodo0* era el que posteriormente rellenaba su matriz *coef* con los

trozos recogidos del resto. Viendo que esto era un problema, y que no se podía utilizar el comando "allocate" (por no estar definido para MPI) para reservar memoria y de esa forma solo pasar el trozo que interesaba de la matriz, se cambio el programa de forma que pasara la matriz *coef* linea a linea, y además, solo hasta la posición 2*(numero nodos del contorno), en lugar de 2*(máximo numero de nodos admisible) que es la dimensión real de la matriz. Con esto se redujo el tiempo de paso de información, dejando de ser el proceso embudo, y así dieron resultados mas coherentes. Para el problema del gancho se calcularon tiempos y el resultado fue el siguiente (tabla 3.4):

	Nodos cluster	Integ contorno	Resolvedor	Ptos internos	Total
NORMAL	1	8,39	127,77	91,36	227,83
	2	11,52	139,95	95,96	245,45
	3	22,54	140,83	47,99	195,15
	4	33,55	136,14	30,69	172,46
	5	44,63	142,04	23,59	171,23
	6	12,14	145,46	19,2	169,7
	7	23,12	143,88	30,8	181,31
PARALELIZADO	8	35,45	145,04	26,49	177,55
	9	44,99	147,03	22,02	176,88
	10	14,58	144,61	21,08	171,33

Tabla 3.4: Tiempos del problema "Gancho"

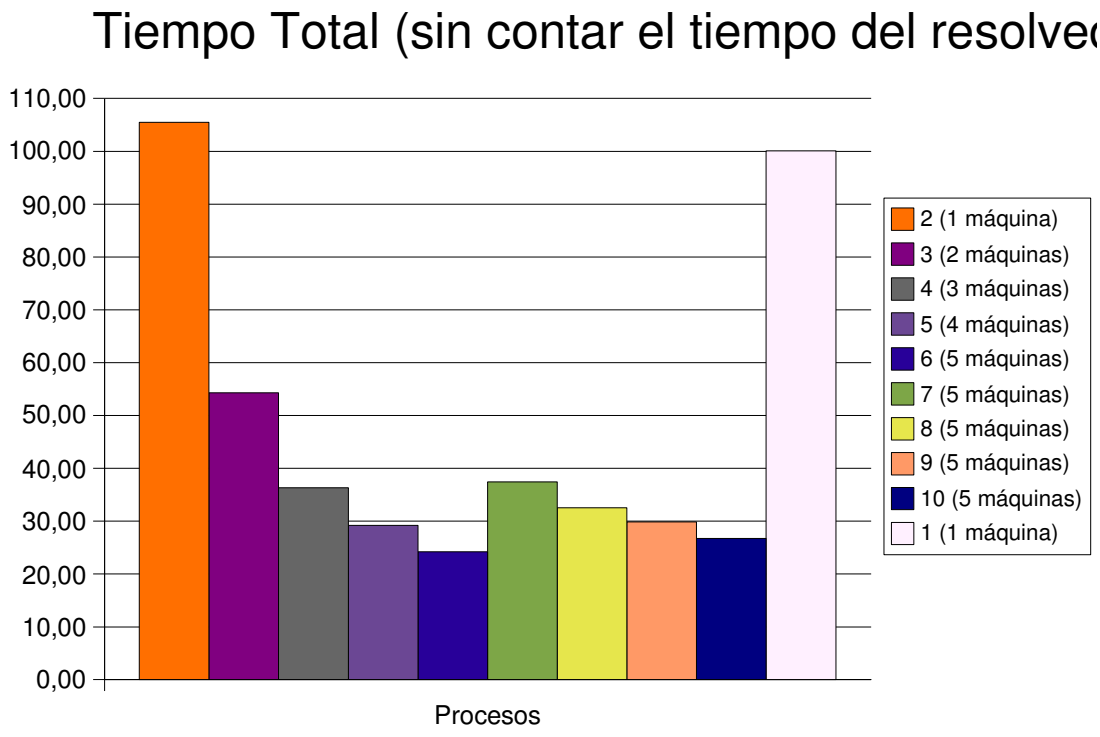

Figura 3.2

Para poder evaluar los resultados de la tabla 3.4 y de la figura 3.2 es necesario comprender el concepto de proceso en paralelo. Aunque tengamos un número N de máquinas, podemos ejecutar el programa en M procesos, y a través de un archivo llamado *machifile.dat* se dividen las máquinas el número de procesos, por lo que podríamos, a modo de ejemplo, incluso ejecutar en paralelo en una sola máquina. Evidentemente esto sería ineficiente pero es un recurso muy utilizado cuando el trabajo está en fase de construcción.

En la figura 3.2 puede observarse como tomando como referencia el programa ejecutado de forma secuencial y comparándolo con el paralelo ejecutado en 2 procesos, aumenta el tiempo de cómputo. Esto es debido al tiempo de paso de información del nodo que calcula al *nodo0*, o administrador. Téngase en cuenta que no se puede ejecutar la aplicación en paralelo en un sólo procesador (una sola máquina) ya que esta programado para que como mínimo uno calcule y otro administre.

Vemos como el resultado de tiempo mas bajo nos da 6 procesos, es decir, una máquina administra y calcula y las otras cuatro sólo calculan. Mas tareas hacen que una de las máquinas trabaje mas de una vez y por tanto sea mas ineficiente el programa.

Como conclusión de esta sección, proponemos que el número óptimo de procesos, es decir de tareas a repartir entre las máquinas es de 6, si bien creemos que con un cambio de configuración en el cluster podrían disminuir los tiempos. Esto es debido a que la computadora principal también se utiliza como nodo “trabajador” en cada ciclo de 5 tareas. En nuestra opinión habría que suprimirla del archivo *machifile.dat*, que contiene la lista de procesadores que trabajan.

En cuanto al resolvedor, no estaba en principio paralelizado, pues ya en una versión posterior, se implantaría el paquete de trabajo Scalapack, que es una aplicación basada en MPI que hará que no sea necesario el paso de los datos y que el resolvedor actué en paralelo desde cada nodo.

El formato de archivo de entrada para del programa SERBAPA y de la aplicación se encuentra en el Anexo 7.1. En el Anexo 7.2 encontramos un esquema de trabajo del SERBAPA. El código del programa se puede localizar en el CD adjunto al trabajo.

3.3.- Ejemplos de prueba

Se validaron los resultados del programa con una chapa cuadrada a la que se sometía a tracción horizontal y vertical. La chapa se dividía en 20, 40, 60, 80 y 100 divisiones por cada eje hasta llegar a un total de 10000 puntos internos en el último ejemplo. A continuación se representan las tensiones en el dominio. La primera de las gráficas de la figura 3.4 representa el desplazamiento a escala de la chapa de estudio. En cada uno de los ejemplos se le sustraía una cantidad de material del 5% en forma de agujeros de forma lo mas homogénea posible. Esta geometría reúne las características que se encuentran en el algoritmo de resolución de optimización topológica.

- Gráficos resultantes con 20 divisiones por cada eje (Figura 3.4):

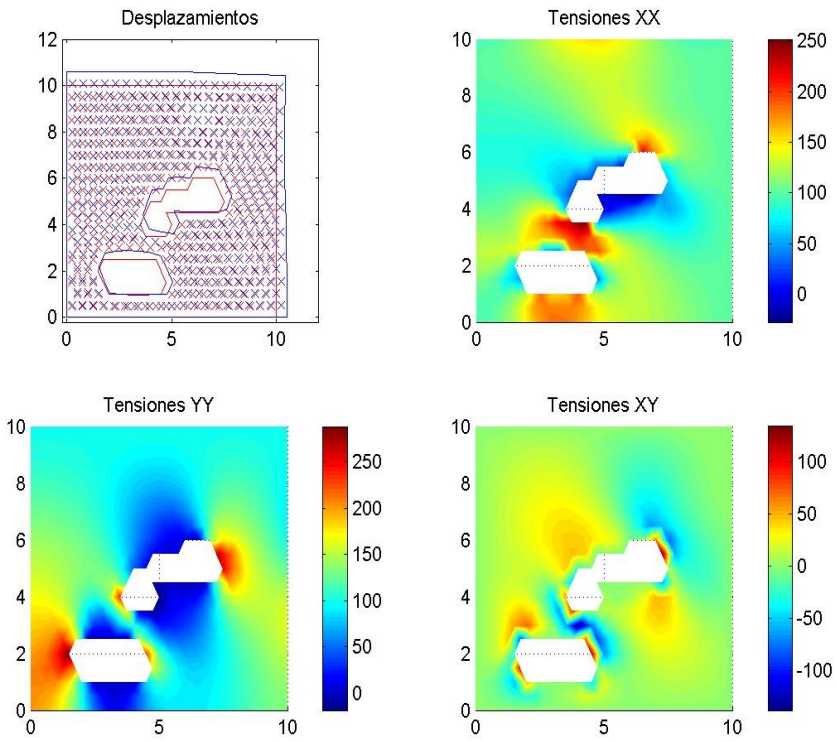

Figura 3.4: Desplazamientos y mapa de tensiones

- Gráficas resultantes con 40 divisiones por cada eje (Figura 3.5):

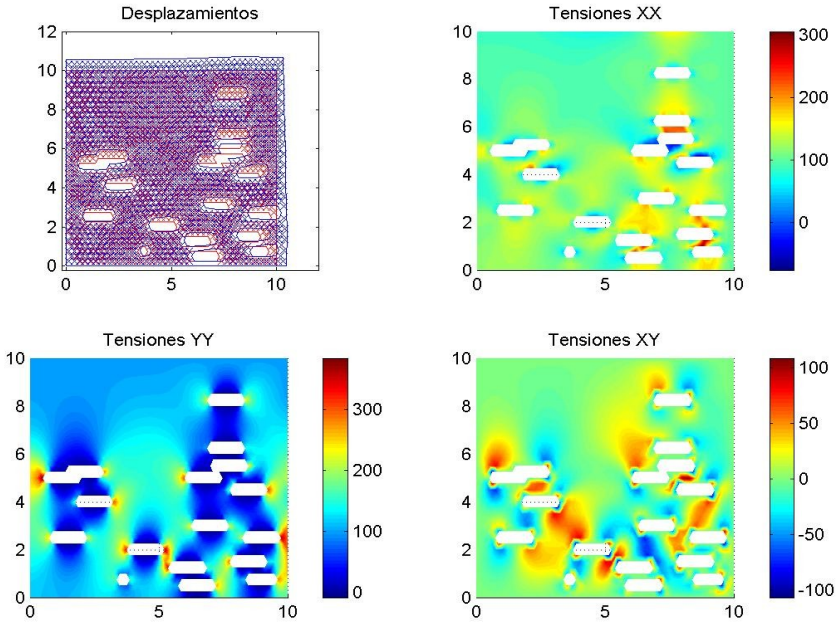

Figura 3.5: Desplazamientos y mapa de tensiones

- Gráficos resultantes con 60 divisiones por cada eje (Figura 3.6):

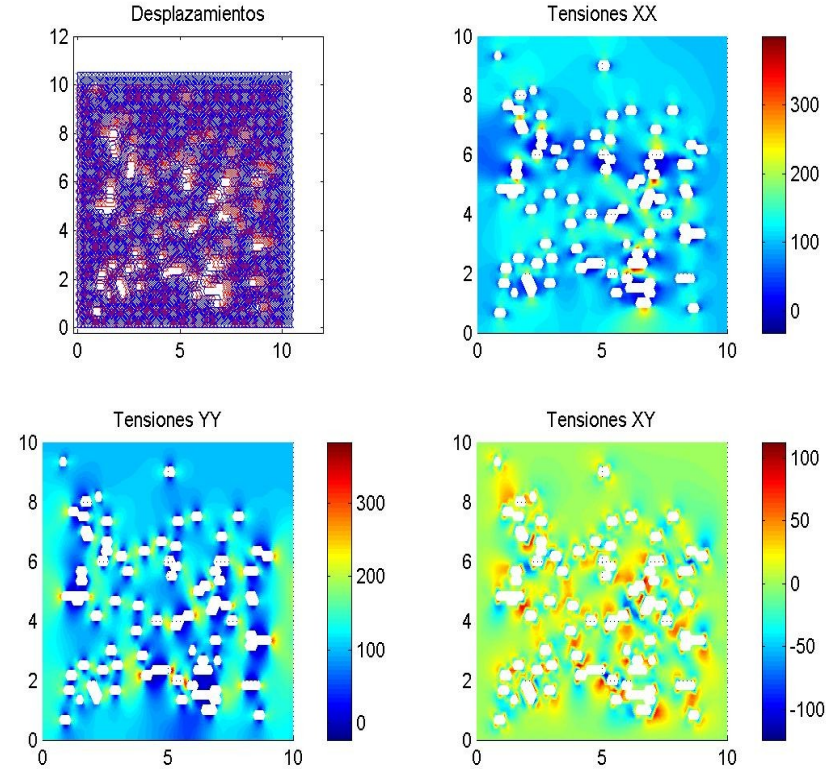

Figura 3.6: Desplazamientos y mapa de tensiones

- Gráficos resultantes con 80 divisiones por cada eje (Figura 3.7):

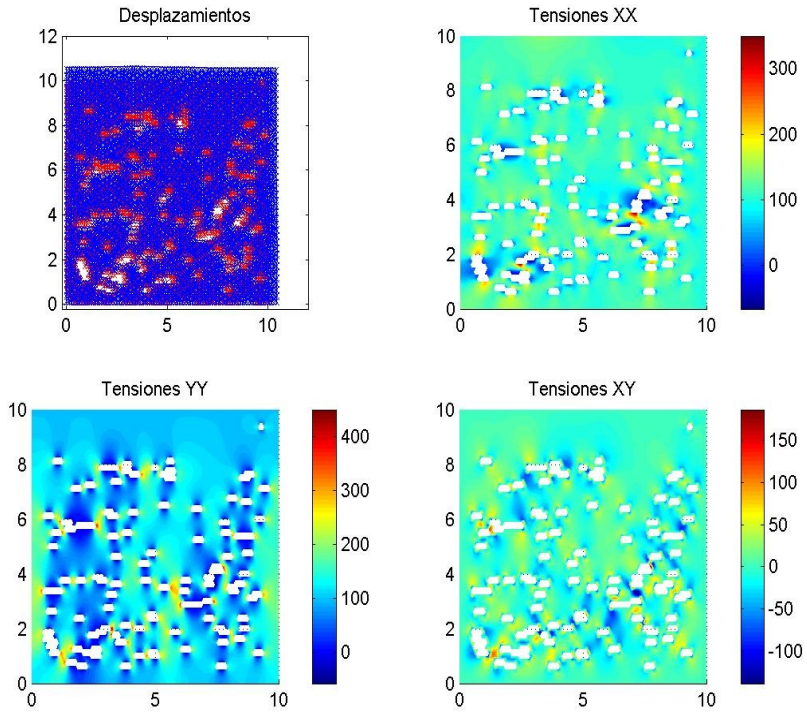

Figura 3.7: Desplazamientos y mapa de tensiones

- Gráficos resultantes con 80 divisiones por cada eje (Figura 3.8):

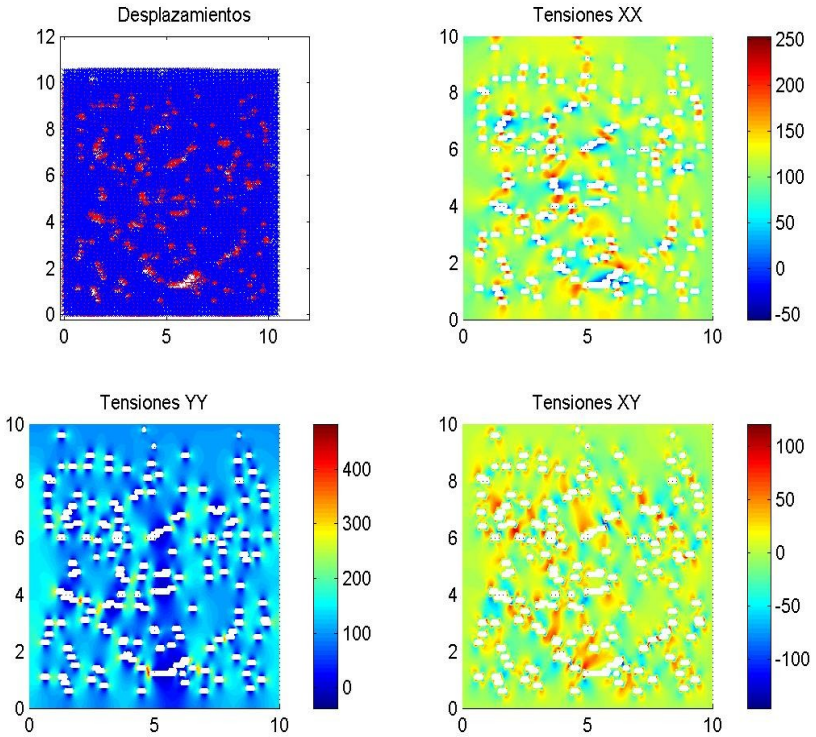

Figura 3.8: Desplazamientos y mapa de tensiones

7.1.- Formato de archivo de entrada

A continuación se presenta el formato que debe tener el archivo de entrada de la aplicación estudiada. En la siguiente página se encuentra la aclaración de cada una de las variables del formato.

Nombre_del_archivo.
Num_pts_contorno Num_pts_internos Num_pts_fijos
Tipo_de_probema E Poisson Lon_max
Coord_x_pto_interno Coord_y_pto_interno
....
....
Num_pto_contorno Coord_x_pto_contorno
Coord_y_pto_contorno
....
....
Num_pto_contorno Cod1 Val1 Cod2 Val2 Cod3 Val3
Cod4 Val4
....
....
Num_elemento Nodo_entrada Nodo_salida
....
....
Numero_punto_total Prescindibilidad
....
....

Aclaraciones del formato:

Nombre_del_archivo: Sera una cadena de caracteres sin espacios de máxima longitud 50.

Num_pts_contorno: Número de puntos que tendrá el contorno. Su valor máximo será de 2000 puntos.

Num_pts_internos: Número de puntos internos a evaluar en el problema. Su valor máximo será de 40.000.

Num_pts_fijos: Número de puntos fijos a evaluar en el problema. Su valor máximo será de 5.

Tipo_de_probema: Podrán evaluarse problemas en tensión o deformación plana. Los valores serán 0 o 1 respectivamente.

E: Modulo de elasticidad E del material

Poisson: Constante de Poisson del material.

Lon_maxima: Máxima longitud del problema. Solo de forma aproximada

Coord_x_pto_interno: Coordenada x de cada uno de los puntos internos

Coord_y_pto_interno: Coordenada y de cada uno de los puntos internos

Num_pto_contorno: Número de punto de contorno en el que estamos.

Coord_x_pto_contorno: Coordenada x de ese nodo del punto del contorno.

Coord_x_pto_contorno: Coordenada y de ese nodo del punto del contorno.

Num_pto_contorno: Número de punto de contorno en el que estamos.

Cod1: código del valor del elemento anterior al nodo en el que nos encontramos en la dirección normal a dicho elemento. Se adjunta tabla de códigos.

Val1: valor de dicho código en esa dirección (normal anterior)

Cod2: código del valor del elemento anterior al nodo en el que nos encontramos en la dirección tangencial a dicho elemento. Se adjunta tabla de códigos.

Val2: valor de dicho código en esa dirección (tangencial anterior)

Cod3: código del valor del elemento posterior al nodo en el que nos encontramos en la dirección normal a dicho elemento. Se adjunta tabla de códigos.

Val3: valor de dicho código en esa dirección (normal posterior)

Cod4: código del valor del elemento posterior al nodo en el que nos encontramos en la dirección tangencial a dicho elemento. Se adjunta tabla de códigos.

Val4: valor de dicho código en esa dirección (tangencial posterior)

Numero_elemento: Número del elemento en el que estamos. Deberán tener

sentido horario en el contorno externo y sentido antihorario en los huecos.

Nodo_entrada: Nodo de entrada del elemento.

Nodo_salida: Nodo salida del elemento.

Numero_punto_total: Numero que le corresponde según un listado de puntos del contorno mas puntos internos.

Prescindibilidad: Nos dice si un punto es prescindible o no y si tiene condiciones en la frontera. Tendrá un valor de 2 en caso de tener condición de contorno, un 1 si no se puede quitar y un 0 si es prescindible, es decir, que se puede quitar.