IMPLEMENTACIÓN DE UN RESOLUTOR EFICIENTE DE LA ECUACIÓN DE POISSON POR ELEMENTOS FINITOS, EN ARQUITECTURAS TIPO SECUENCIAL Y SIMD

Enzo A. Dari
Marcelo J. Vénere
División Mecánica Computacional
Centro Atómico Bariloche - CNEA
Bariloche - Argentina

RESUMEN

Se describe la implementación de un resolutor eficiente por Elementos Finitos de la Ecuación de Poisson en dos y tres dimensiones. Se utilizaron símplices en ambos casos, con redes no estructuradas. Se comparó la perfomance de distintas técnicas para resolver el sistema de ecuaciones resultante, con distintas formas de almacenamiento de la matriz de rigidez. Los resultados numéricos aconsejan el uso de técnicas de almacenamiento ralo con resolución directa del sistema de ecuaciones.

Se analizó además la implementación del resolutor que emplea un esquema de Gradientes Conjugados Elemento a Elemento en una máquina con arquitectura SIMD de pocos procesadores.

ABSTRACT

The implementation of an efficient Finite Element Poisson solver in two and tree dimensions is described. Simplex are used in both cases, with unstructured grids. The perfomance of different solution techniques and different stiffness matrix storing methods are tested. Numerical results has shown the advantages of using sparse matrix techniques and direct solution.

In adition, the implementation of the solver using a Conjugate Gradient - Element by Element scheme in a few processor SIMD machine was analized.

INTRODUCCIÓN:

La resolución de un problema por Elementos Finitos comprende los siguientes pasos:
-Cálculo de las matrices elementales
-Ensamblaje de la matriz del sistema de ecuaciones
-Resolución del sistema

Cuando se trabaja con elementos simples, la etapa que más esfuerzo computacional requiere es la de resolución del sistema de ecuaciones, ya que se realiza una gran cantidad de operaciones de punto flotante sobre los elementos de la matriz del sistema. Dos factores importantes a tener en cuenta son: el modo de almacenamiento de la matriz del sistema y el método de resolución.

En una primera parte analizaremos la perfomance de distintas técnicas de almacenamiento de la matriz de rigidez y de resolución del sistema de ecuaciones, para la resolución por Elementos Finitos del problema de Poisson bi y tri-dimensional, a coeficientes constantes por elemento. Se utilizaron para discretizar el dominio triángulos de tres nodos (resp. tetraedros de cuatro nodos), y como esquema de resolución el método de Galerkin con funciones base lineales [1].

Finalmente, analizaremos la programación del "Núcleo de Poisson" en una máquina con arquitectura tipo SIMD.

EL NÚCLEO DE POISSON:
ANÁLISIS DE PERFOMANCE DE DISTINTAS TÉCNICAS

Técnicas de almacenamiento de la matriz de rigidez:

Frecuentemente la matriz del sistema de ecuaciones que surge de la discretización por Elementos Finitos de un problema es "rala", es decir, tiene una gran cantidad de elementos nulos. El almacenamiento de la matriz completa requeriría demasiada memoria, además de aumentar innecesariamente el número de operaciones necesario para su resolución (se realizan muchas operaciones sobre elementos nulos). Por otra parte, la matriz que surge de discretizar con el método descripto la Ecuación de Poisson por Elementos Finitos es simétrica y definida positiva, propiedades éstas de relevancia en la elección de los algoritmos de resolución.

Almacenamiento Banda:

Esta técnica consiste en almacenar todos los elementos ij de la matriz tal que $0 < j - i < \beta(A)$, donde:

$$\beta(A) = \max_{i,j} \{ |j-i| \ / \ a_{ij} \neq 0 \}$$

es el mínimo número de supradiagonales que es necesario almacenar para tener todos los elementos no nulos de la matriz A en memoria. Por ser la matriz simétrica sólo se almacena la semi-banda superior (o inferior).

Almacenamiento Skyline:

Este tipo de almacenamiento está especialmente pensado para cuando se pretende resolver el sistema de ecuaciones resultante por eliminación de Gauss (más precisamente, para matrices simétricas: factorización de Cholesky). El mismo consiste en un almacenamiento "banda variable", es decir, se guardan los elementos a_{ij} / $0 < j-i < \beta_j$ (A), donde:

$$\beta_j\ (A) = \max_i \{\ |j-i|\ /\ a_{ij} \neq 0\ ,\ j > i\ \}$$

Es fácil ver que, cuando se calcula el factor triangular de Cholesky de la matriz A, el mismo puede ser almacenado con la misma estructura que la matriz A, esto es, los elementos que dejan de ser nulos durante la factorización, tienen espacio reservado en la matriz original.

Para minimizar la cantidad de memoria a utilizar se suele renumerar la red, de modo de que el semiancho de banda (β (A) + 1) sea lo menor posible. Si bien no existe un algoritmo que garantice este resultado, uno de los más empleados, y que da en general buenos resultados es el debido a Cuthill y McKee [2]. Para el método Skyline se utiliza una variante, que consiste en invertir el ordenamiento del método de Cuthill-McKee [3]; se ha probado [4] que esta modificación nunca empeora el ordenamiento original (desde el punto de vista de memoria utilizada y número de operaciones necesarias para factorizar la matriz), mejorándolo frecuentemente, sin que cambie el semiancho de banda de la matriz.

Almacenamiento Ralo:

Esta técnica permite sacar el máximo provecho posible de la "ralitud" de la matriz, dado que sólo almacena los coeficientes no-nulos [5]. Cuando se resuelve el sistema por factorización de Cholesky, aparecen elementos no-nulos en el factor triangular, que eran nulos en la matriz original; a este fenómeno se lo denomina "llenado" del factor triangular. Para minimizar el llenado es necesario un adecuado ordenamiento de los nodos de la red, para lo cual existen varios algoritmos (ver por ejemplo: [5,6])

Resolución del sistema de ecuaciones resultante:

Se analizarán dos métodos para la resolución del sistema de ecuaciones: uno directo (Factorización de Cholesky) y otro iterativo (Gradientes Conjugados).

Las operaciones necesarias para resolver por gradientes conjugados un sistema de ecuaciones son productos escalares, productos matriz-vector y sumas escaleadas de vectores. Al no presentarse el problema del llenado de los factores triangulares, la técnica de almacenamiento ralo es, para el caso más general de redes no-estructuradas, la más adecuada. (Realiza operaciones sólo con los elementos no nulos. No es necesaria una renumeración de las incógnitas).

Resultados obtenidos:

Se implementaron programas para la resolución de la ecuación de Poisson estacionaria, con las técnicas mencionadas en las secciones anteriores, con el fin de analizar su perfomance (tiempo empleado y memoria utilizada).

Los resultados obtenidos se muestran en las Figs 1 y 2. Las redes utilizadas en todos los casos fueron no-estructuradas, renumerados sus nodos según el método de Cuthill-McKee inverso para el caso de almacenamiento Banda y Skyline, y según el ordenamiento de grado mínimo [7] para el caso de almacenamiento Ralo.

Los tiempos analizados corresponden a la suma de tiempo de ensamblaje de la matriz y resolución del sistema, teniendo en cuenta el caso en que, en general, será utilizado el núcleo de Poisson: centro de un lazo iterativo, cambiando las condiciones de contorno y/o los coeficientes, pero no la topología de la red. En este caso, el tiempo necesario para el resto de las etapas del cálculo se hace rápidamente despreciable.

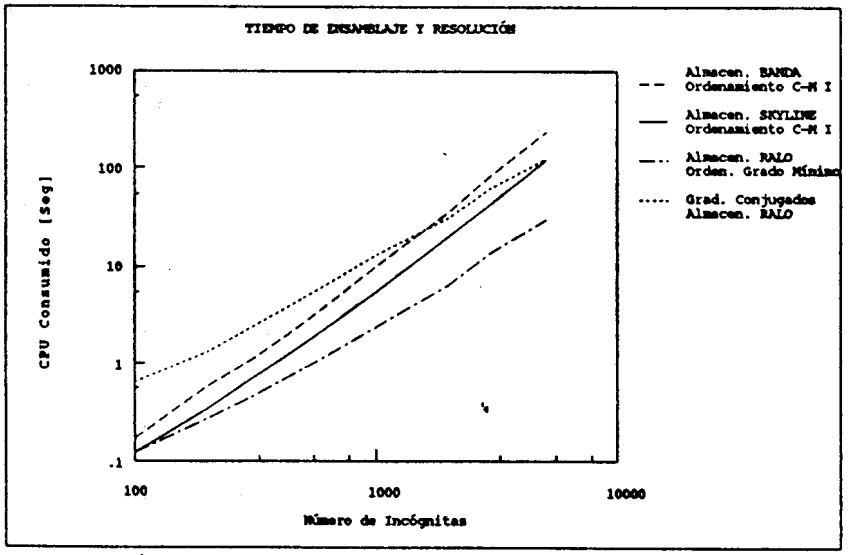

Figura 1 : Perfomance de distintas técnicas de resolución (Tiempo de CPU)

En la Fig 1 se muestra el tiempo de CPU consumido, en función del número de incógnitas. Realizando un ajuste de las curvas obtenidas experimentalmente se observa que las técnicas Banda y Skyline emplean $O(N^{1.8})$ para las etapas mencionadas, mientras que la técnica de almacenamiento Ralo tarda $O(N^{1.4})$. El "punto de corte" (número de incógnitas para el cual se hace ventajosa la técnica de menor orden) está en las 100 incógnitas. También se muestra el tiempo consumido por el programa que utiliza almacenamiento ralo y resolución por gradientes conjugados; el mismo tiene el mis-

mo orden que la resolución directa con almacenamiento ralo, aunque con una constante mayor. Cabe mencionar que no se realizó ningún tipo de precondicionamiento, con lo cual, al menos en teoría, debería disminuir el orden de este algoritmo.

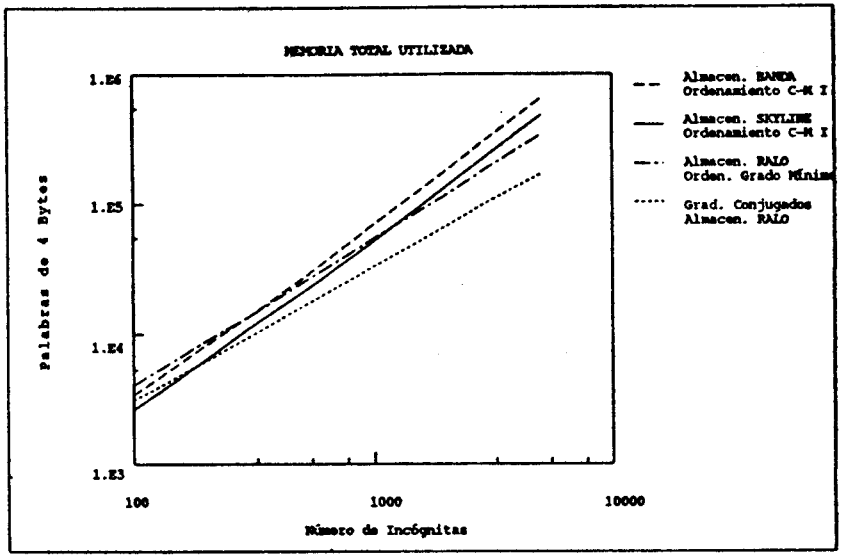

Figura 2 : Perfomance de distintas técnicas de resolución (Memoria utilizada)

En cuanto a la memoria utilizada, los resultados se pueden observar en la Fig 2. Análogamente se obtuvo que tanto en Banda como en Skyline se necesitan O ($N^{1.4}$) lugares de memoria para la resolución del problema por el método directo, mientras que la técnica rala utiliza sólo O ($N^{1.1}$). El "punto de corte" en este caso es de 1000 incógnitas. Al resolver el problema por el método iterativo de gradientes conjugados, la memoria requerida es O(N), ya que no se presenta el problema del llenado del factor triangular.

En base a la experiencia obtenida con la programación del núcleo de Poisson bidimensional, se implementó el caso tridimensional, utilizándose tetraedros de cuatro nodos, técnica de almacenamiento ralo y resolución directa del sistema de ecuaciones.

PROGRAMACIÓN DEL NÚCLEO DE POISSON EN UNA ARQUITECTURA SIMD

Descripción de la arquitectura:

Una matriz sincrónica de procesadores en paralelo consiste en un conjunto de procesadores, bajo la supervisión de una única unidad de control. A este sistema se lo conoce con el nombre de procesador matricial (Array Processor) y puede manejar una única instrucción operando sobre un conjunto de datos simultáneos (Single Instruction-Multiple Data o SIMD).

Nos ocuparemos de un caso particular de esta arquitectura, que es el que corresponde a la supercomputadora dedicada del proyecto ARPAS de CNEA [8], basada en un proyecto similar del Istituto Nazionale di Fisica Nucleare de Italia [9,10].

La arquitectura de estas supercomputadoras se muestra en la Fig 3. Consta de un controlador central, que envía instrucciones a los demás elementos: un banco de 16 Unidades de Punto Flotante (FPU), un banco de 16 conjuntos de memorias, una red de intercomunicación y un sequencer. Todo el conjunto actúa como coprocesador de un "host", con el cual se comunica a través del controlador central.

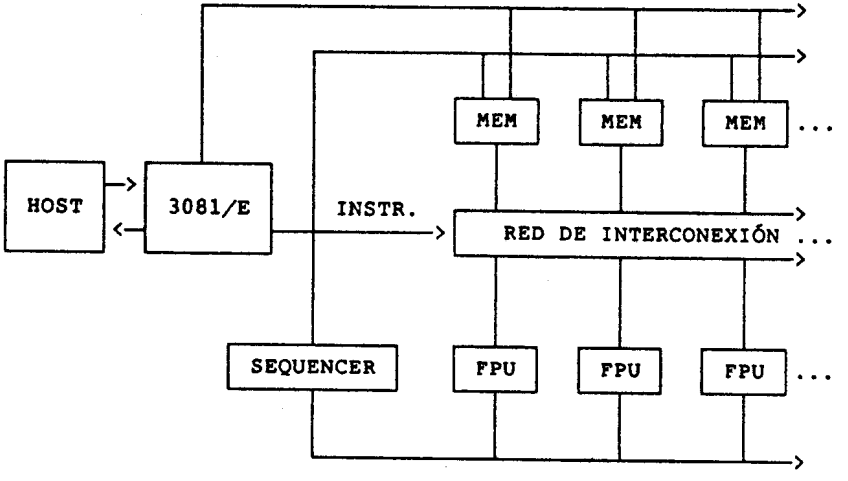

Figura 3 : Arquitectura de las supercomputadoras dedicadas de los proyectos APE / ARPAS

Esta arquitectura fue diseñada originalmente para la resolución de ecuaciones de cromodinámica cuántica (QCD) en su formulación por celdas. La discretización de estas ecuaciones en el espacio-tiempo tetradimensional, para alcanzar un nivel de precisión aceptable lleva al cálculo de una integral en 5×10^6 variables para un retículo de 20^4 sitios (para lo cual se utilizan métodos Montecarlo), y el cálculo de algunas columnas de una matriz rala de $10^6 \times 10^6$. Estas estimaciones fueron hechas con la experiencia de unas 10^3 horas de cálculo en Cray-1.

La naturaleza del problema, y los métodos iterativos empleados para resolverlo hacen especialmente adecuado este tipo de arquitectura, ya que:

- Se trata de un problema local, lo cual minimiza la complejidad de la red de intercomunicación entre FPU's-memorias.
- Se pueden usar algoritmos paralelos en forma natural.
- Los cálculos son en su mayoría multiplicaciones y sumas de punto flotante.

La red de comunicación es tal que si se asigna una memoria a cada FPU, la misma puede acceder a su propia memoria, o a la de sus vecinas (estructura en anillo).

Las FPU son unidades aritméticas de punto flotante, que operan con números complejos (resp. reales en el proyecto ARPAS) de 32 bits (simple precisión), realizando en forma optimizada la operación A * B + C. Tanto los sumadores como los multiplicadores trabajan en modo "pipeline"[11, Cap. 3], de modo que se puede obtener un resultado por ciclo de reloj.

Programación del núcleo de Poisson:

Para la resolución por Elementos Finitos de la ecuación de Poisson en esta arquitectura, se utilizó un esquema elemento-a-elemento (EAE) [12], con el fin de obtener un alto grado de concurrencia en las operaciones. Como método de resolución se eligió Gradientes Conjugados.

En el esquema utilizado, cada elemento es procesado por una FPU, lográndose un buen balance de carga cuando el número de elementos es mucho mayor que 16. Las matrices elementales (y los Vectores Término Independiente elementales, VTI) son calculados completamente en paralelo, y son almacenados en las memorias asociadas a cada FPU.

La matriz global del sistema de ecuaciones resultante nunca se ensambla (Se elimina la segunda de las etapas mencionadas en la introducción).

Las condiciones de contorno se colocan a nivel de las matrices elementales, colocando un 1 en la posición diagonal correspondiente al nodo con condición de Dirichlet, ceros en el resto de la fila, y el valor de la condición en el VTI elemental. Para conservar la simetría de las matrices elementales, la columna correspondiente a cada nodo con condición de contorno se elimina, pasando estas contribuciones al VTI elemental.

En las condiciones de contorno tipo Neumann (de derivada) está involucrado sólo un elemento, la contribución de las mismas se suma en el VTI elemental.

La resolución por Gradientes Conjugados implica en cada iteración el cálculo de un producto matriz-vector, que es el que mayor tiempo de CPU consume. Para calcular este producto en paralelo, no se ensamblan las matrices elementales, sino que se descompone en:

$$p = A \cdot d = \sum_{e=1}^{NEL} A_e \cdot d = \sum_{e=1}^{NEL} p_e$$

donde p es el vector producto, A es la matriz global y A_e son las matrices elementales, NEL es el número de elementos de la red, los cuales son procesados por distintas FPUs.

El almacenamiento en memoria de las matrices elementales está pensado para el mejor aprovechamiento del pipeline de cada FPU [13]. Se almacena el elemento A_{11} para todos los elementos asignados a la FPU, luego los elementos A_{22}, los elementos A_{33}, los A_{12}, A_{23}, y finalmente los A_{13}. El vector d se redistribuye en memoria, de modo que los primeros NEL/16 lugares estén ocupados por sus componentes correspondientes a la numeración del 1er nodo de la descripción de cada elemento asignado a la FPU; en los siguientes NEL/16 lugares los elementos de d correspondientes a la numeración del 2do nodo de cada elemento, y los restantes NEL/16 contienen los elementos de d correspondientes a la numeración del último nodo de la descripción de cada elemento asignado a esta FPU.

Con esta disposición de los datos en memoria, la multiplicación matriz-vector puede hacerse en 5 lazos:

- Uno de longitud 3*NEL/16 que calcula:

$$p_1 = A_{11} * d_1$$
$$p_2 = A_{22} * d_2$$
$$p_3 = A_{33} * d_3$$

- Otro de longitud 2*NEL/16, calculando:

$$p_1 = p_1 + A_{12} * d_2$$
$$p_2 = p_2 + A_{23} * d_3$$

- Otro de longitud NEL/16:

$$p_3 = p_3 + A_{13} * d_1$$

- Otro de longitud NEL/16:

$$p_1 = p_1 + A_{13} * d_3$$

- Y otro de longitud 2*NEL/16, que calcula:

$$p_2 = p_2 + A_{12} * d_1$$
$$p_3 = p_3 + A_{23} * d_2$$

Duplicando en memoria los elementos A_{13}, los d_1 y los d_3 se pueden reducir los últimos 4 lazos a 2 de longitud 3*NEL/16 cada uno. Para nuestro caso, en problemas grandes no se justificaría, ya que el número de etapas de los pipelines es pequeño, y se logra una buena perfomance con longitudes de lazos no demasiado grandes.

El punto crítico de este algoritmo es el ensamblaje de los vectores p elementales en un vector global, de dimensión NOD (Nro de nodos de la red). Es necesario un ensamblaje dependiente de los elementos asignados a cada FPU, y luego un ensamblaje de los vectores de cada FPU en un único vector global. Dado que en esta arquitectura el direccionamiento es idéntico para todas las FPU, este ensamblaje debe hacerse en forma secuencial.

El resto de las operaciones interiores al loop de iteraciones de Gradientes Conjugados son: 2 productos escalares de longitud NOD, sobre vectores globales (no dependientes de cada FPU). Una suma escaleada de vectores de longitud 3*NEL/16 y 2 sumas escaleadas de vectores de longitud NOD. Todas estas operaciones pueden ser realizadas en pipeline,

obteniéndose un buen aprovechamiento de la estructura de cada FPU, si las longitudes de los vectores son suficientemente largas (se obtiene un resultado A * B + C por cada ciclo de reloj).

Este esquema fue programado en el lenguaje que utilizará la máquina del proyecto ARPAS, cuyo compilador ya está implementado en su primera fase.

Si bien la supercomputadora del proyecto ARPAS no está construída a la fecha, y por lo tanto no se puede evaluar la perfomance del algoritmo descripto, se pueden hacer algunas predicciones en base al número de operaciones necesario y el grado de paralelismo en cada etapa del cálculo.

En la etapa del cálculo de las matrices elementales es donde este algoritmo tiene su principal ventaja. Las operaciones pueden efectuarse completamente en paralelo y es de esperar un factor de aceleración con respecto al cálculo secuencial aproximándose a 16 para un número suficientemente grande de elementos. No es de extrañar que cuando se busca la arquitectura óptima para una "Máquina de Elementos Finitos" se elija una arquitectura SIMD para la etapa del cálculo de matrices elementales [14].

En este esquema no se realiza el ensamblaje de la matriz global, por lo tanto es de esperar que el número de operaciones cuando se trabaje con la matriz del sistema de ecuaciones sea mayor que el mínimo.

El punto de mayor consumo de tiempo de CPU es el loop de gradientes conjugados, y de este loop, la multiplicación matriz-vector; veamos el número de operaciones necesarias: por cada matriz elemental de 3 * 3 son 9 productos y 6 sumas, divididas en las 16 FPUs, ya que se pueden efectuar en paralelo. El ensamblaje del vector producto elemental se debe hacer en forma secuencial, e insume: 3*NEL sumas. Con este ensamblaje se obtiene un vector de longitud NOD en la memoria asociada a cada FPU; para obtener el vector global son necesarias $\log_2 16 = 4$ sumas por cada nodo. En total, para el cálculo del vector producto fueron necesarias:

$$(15/16 + 3) * NEL + 4 * NOD$$

operaciones de punto flotante. Utilizando la aproximación: NEL ≈ 2 * NOD, válida para redes grandes (con poco borde):

$$\# \text{ de operaciones} = 11.875 * NOD$$

Comparemos con el número de operaciones necesario en caso de utilizar un procesador, ensamblando la matriz global en formato ralo: para inicializar el vector p se puede utilizar el producto de los elementos diagonales de A y los elementos de d; son NOD productos. Para terminar de calcular el producto son necesarias 2 * NAR multiplicaciones y otras tantas sumas (NAR: Nro de aristas de la red = Nro de elementos extradiagonales de la matriz), el número total de operaciones es:

$$NOD + 4 * NAR$$

que, utilizando la aproximación para redes con poco borde: NAR = 3 * NOD se reduce a:

de operaciones = 13 * NOD

Como puede verse, no se obtiene con el esquema propuesto un buen factor de aceleración del cálculo, por usar más de 1 procesador. Sin embargo no se tuvo en cuenta el aprovechamiento del pipeline de las FPU. De las 11.8 operaciones del 1er algoritmo, 5.8 son completamente vectorizables, y el resto en forma parcial. De las 13 operaciones por nodo del 2do caso, 7 son vectorizables y 6 no lo son.

CONCLUSIONES

Se presenta un análisis de perfomance de varias implementaciones del Método de Elementos Finitos para la resolución de la ecuación de Poisson bidimensional.

Entre aquellas que utilizan la resolución directa del sistema de ecuaciones, la de menor orden (almacenamiento ralo de la matriz de rigidez) resulta ser conveniente desde un número de incógnitas suficientemente bajo como para que se justifique su empleo en todo el rango de las aplicaciones de interés.

En cuanto a la resolución iterativa estudiada, si bien el orden es igual (o menor) que cualquier técnica de resolución directa, el tiempo de CPU es mayor en todo el rango verificado.

Además se estudia la implementación de un resolutor eficiente de la ecuación de Poisson en una máquina con arquitectura SIMD. El esquema utilizado (Gradientes Conjugados Elemento a Elemento) es paralelizable en gran parte. Sin embargo, las partes no-paralelizables del mismo hacen que el factor de aceleración sea cercano a 1.

Para lograr aumentar este factor se pueden trabajar tanto sobre el hardware como sobre el software. En el primer caso, si fuera posible el direccionamiento indirecto, una buena parte del ensamblaje sería paralelizable. En el segundo caso, si se empleara alguna subdivisión adecuada del dominio se podrían disminuir sensiblemente las operaciones necesarias para el ensamblaje de los vectores.

REFERENCIAS

[1] F. BASOMBRÍO - "Resolución por Elementos Finitos de la ecuacion Cuasiarmónica bidimensional" CNEA-NT 31/78.

[2] E. CUTHILL and J. MCKEE - "Reducing the bandwidth of sparse symmetric matrices" Proc. 24th Nat. Conf. Assoc. Comput. Mach., ACM Publ. pp 157-172 (1969)

[3] A. GEORGE - "Computer Implementation of the Finite Element Method" Tech. Rept. STAN-CS-208, Standford University (1971)

[4] J. W. LIU and A. H. SHERMAN - "Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices" SIAM Journal of Numerical Analysis Vol. 13 pp 198-213 (1975)

[5] S. PISSANETZKY - "Sparse matrix technology" Academic Press (1984)

[6] A. GEORGE and J. W. LIU - "Computer solution of Large Sparse Positive Definite Systems" Prentice-Hall series in computational mathematics. Prentice-Hall (1981)

[7] W. F. TINNEY - "Comments on using sparsity techniques for power system problems" in Sparse Matrix Proceedings, IBM Research Rept. RAI 3-12-69 pp 25-34 (1969)

[8] C. J. BOGNI et al. - "Proyecto Supercomputadora Dedicada" Informe interno CNEA

[9] G. SALINA - "Progetto APE: un super-computer dedicato per teorie di gauge sul reticolo" Tesina di laurea. Universidad de Roma (1985)

[10] P. BACILIERI et al. - "The APE project" Nota Interna IFU, Universita di Roma 1, n. 839 (1984)

[11] C. J. BOGNI y L. A. MARRONE - "Computadoras: Introducción a las Arquitecturas Paralelo" Campinas: Editora da UNICAMP (1986)

[12] E. BARRAGY and G. F. CAREY - "A Parallel Element-by-Element Solution Scheme" Int. J. for Numer. Methods in Eng. Vol. 26 pp 2367-2382 (1988)

[13] L. J. HAYES and P. DEVLOO - "A Vectorized Version of a Sparse Matrix-Vector Multiply" Int. J. for Numer. Methods in Eng. Vol. 23 pp 1043-1056 (1986)

[14] D. H. NORRIE and C. I. W. NORRIE - "Program structure and Architecture for a Finite Element Machine" Int. J. for Numer. Methods in Eng. Vol. 22 pp 241-248 (1986)