UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS

LA COMPUTACIÓN PARALELA EN LA SOLUCIÓN NUMÉRICA DE LA ECUACIÓN NO-LINEAL DE ADVECCIÓN-DISPERSION

TESIS

que para obtener
el Título de

Físico

presenta

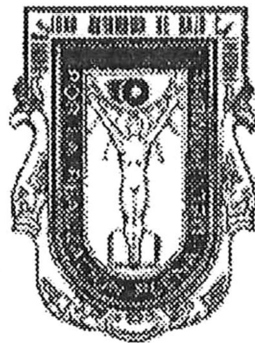
Juan Alberto Aranda Alvarez

Ensenada, B.C., Junio de 1997

UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS

LA COMPUTACIÓN PARALELA EN LA SOLUCIÓN NUMÉRICA DE LA ECUACIÓN NO-LINEAL DE ADVECCIÓN-DISPERSION



TESIS

Que para obtener el título de

Físico

presenta:

Juan Alberto Aranda Alvarez

Ensenada, Baja California, Junio de 1997

Dedicatoria

A mis Padres que me otorgaron todo sin habermelo ganado...

A Tania por su amor y paciencia...

A Bebe Aranda-Nassar por haber llegado

UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS

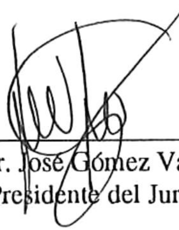
La Computación Paralela en la Solución Numérica de la Ecuación No-Lineal de Advección-Dispersion

T E S I S   P R O F E S I O N A L

QUE PRESENTA

JUAN ALBERTO ARANDA ALVAREZ

APROBADO POR



Dr. José Gómez Valdés
Presidente del Jurado

Fis. Francisco Juárez
SECRETARIO

L.C.C Leopoldo Morán
1er. VOCAL

Agradecimientos

Al Dr. José Gómez Valdés por su infinita paciencia.

A mis sinodales Leopoldo Moran y Francisco Juárez.

A la familia Madrigal Ojeda por toda su ayuda.

A la M.C. Gloria Rubí que siempre fue y es un apoyo en mi carrera.

Al CONACYT por el apoyo económico.

Al CICESE por facilitarme sus instalaciones.

A la UABC por permitirme formarme como profesionista

Al Castillo Inc. por todas las atenciones prestadas

y a mis compañeros de la Facultad de Ciencias de la UABC (Alfonso, Dan-el, Vero, David, Desiderio, Miguel, Berenice, La Nana, Rodrigo, Omar, Raul, Daniel "Lauro", Mary, Mayra, Irma, Harvy, Charlie, El Baisa, El beto Rock, Julio, Billy, Mike, Eri, ...).

RESUMEN de la Tesis de Juan Alberto Aranda Alvarez presentada como requisito parcial para la obtención de la Licenciatura en Física. Ensenada, Baja California, México. Junio de 1997.

La Computación Paralela en la Solución Numérica de la Ecuación No-Lineal de Advección-Dispersion

Resumen aprobado:

Dr. José Gómez Valdés

Con el objetivo general de analizar las ventajas de la computación en paralelo en la investigación de los procesos físicos, se parte de un programa FORTRAN de Wang y Anderson (1982) de la solución numérica de las ecuaciones de movimiento para el problema de advección-dispersión de un contaminante en un acuífero, usando el método de elemento finito y se desarrolla un programa en paralelo usando PVM con algunos elementos de C++ que resuelve el mismo problema. Este se implanta en una red local heterogénea de estaciones de trabajo del departamento de Oceanografia Física de CICESE. Como el método de elemento finito necesita de la solución de un sistema de ecuaciones, se utilizan tres métodos de solución: Jacobi, Gauss-Seidel y Gradiente Conjugado. Se proponen estrategias de paralelización para cada uno, se muestran resultados para el de Jacobi. Después se calcularon tiempos de ejecución y rendimiento para 1, 2, hasta 7 procesadores en la máquina virtual utilizando mallas de 1x10, 1x50, 1x100 y 10x10 elementos. Se concluye que el método de elemento finito es ameno a la paralelización, y que mejora su desempeño al refinar la malla. Al aumentar el número de elementos se obtienen mejoras cuando se usan de 3 a 5 procesadores. PVM resultó un medio viable, altamente recomendable en el proceso de aprendizaje de la computación paralela.

Indice

a) Contenido

I Introducción ____ *1*

II Antecedentes ____ *3*

2.1 Advección-Dispersión ____ 3

2.2 Solución Analítica ____ 6

2.3 Solución Numérica ____ 6

2.3.1 Elemento finito ____ 6

2.3.2 Condiciones a la frontera ____ 11

2.3.3 Solución de la ecuación diferencial matricial ____ 12

2.4 PVM ____ 15

2.4.1 Rendimiento ____ 16

III Metodología ____ *18*

3.1 Proceso de paralelización. ____ 21

3.1.1 Paralelización del método iterativo de Jacobi. ____ 21

3.1.2 Paralelización del método iterativo de Gauss-Seidel. ____ 22

IV Resultados ____ *24*

4.1 Tiempos de ejecución. ____ 24

4.2 Rendimiento: ____ 26

V Discusión ____ *28*

VI Conclusiones ____ *33*

VII Literatura Citada ____ *34*

VIII Apendices o Anexos ____ *36*

1. Código objelem2.hxx ____ 36
2. Código objdom2.hxx ____ 39
3. Código esclav2.cxx ____ 51
4. Código master2.hxx ____ 51
5. Código master2.cxx ____ 63

b) Figuras

Fig. 1 Dominio del problema ______ *5*
Fig. 2 Diagrama esquemático mostrando la relación de las matrices de los elementos y la matriz global. El diagrama muestra los elementos bajo consideración. Tomada de Cheng (1978.) ______ *10*
Fig 3. Comparación de la solución analítica (linea sólida) y la solución numérica (linea punteada), para los tiempos 10, 50, 100, 150, 200, 250, 300, 350, 450, 450 y 500 dias. ______ *29*
Fig 4. Distribución de los datos en la matriz para (a) malla de 10 x 1, (b) malla de 10 x 2 y (c) malla de 10 x 3 ______ *32*

c) Tablas

Experimento 1. El problema original de Wang y Anderson (10 elementos). Utilizando Jacobi con 1, 2, hasta 7 procesadores en la máquina virtual. En microsegundos ______ **24**
Experimento 2. Refinación de la malla de Wang y Anderson de 10 elementos a 50 y 100 elementos. ______ **25**
Experimento 3. Refinacion de la malla de 1 x 10 a 10x10. ______ **25**
Experimento 0. Los tiempos de ejecución de los programas secuenciales. ______ **26**
Experimento 1. (Rendimiento). ______ **26**
Experimento 2.(Rendimiento). ______ **26**
Experimento 3. (Rendimiento). ______ **27**

I Introducción

La calidad y cantidad de las reservas acuíferas se pueden estimar objetivamente mediante modelos del flujo de aguas subterraneas y del transporte de solutos. Las actividades humanas amenazan a estos recursos, impactando negativamente a la cantidad y calidad del agua que se extrae. Dos claros ejemplos son el completo agotamiento de la reserva debido a el excesivo uso de estaciones de bombeo y la contaminación por la existencia de basureros (Istok, 1989).

Los modelos matemáticos que representan la fisica de los acuiferos son en general complejos, de forma tal que encontrar las soluciones de las ecuaciones diferenciales correspondientes es una tarea difícil, si no imposible. Con el advenimiento de computadoras de alto rendimiento, la solución numérica de estas ecuaciones se ha facilitado, y ahora se pueden hacer estudios más cercanos a la realidad.

Se han desarrollado varios métodos numéricos para el estudio de los acuíferos, de los cuales destaca el método de elemento finito. El concepto fundamental del método de elemento finito es que cualquier cantidad continua, e.g., temperatura, presión o desplazamiento, puede aproximarse por un conjunto de funciones seccionalmente continuas a lo largo de un número finito de elementos. Las funciones son definidas usando los valores de la cantidad continua en un número finito de puntos o nodos en su dominio. El método tuvo su origen en la industria aeroespacial en los principios de los cincuentas y fue presentado en Turner *et al.* (1956).

Este método requiere computación de alto rendimiento (Duller, 1983). Tal que para problemas en donde se considere la tridimensionalidad espacial, por ejemplo, es necesario contar con supercomputadoras para alcanzar la eficiencia requerida.

La computación paralela es un paradigma novedoso de cómputo en el cual un cierto número de procesadores y unidades de memoria, nodos, trabajan sincronizada o asincronizadamente, comunicandose entre ellos para evitar conflictos e intercambiar valores intermedios (Landau, 1993). Hay varias maneras de relacionar la memoria y los procesadores. Una de las más populares es aquella en la cual cada procesador tiene su memoria privada y, por ello, se necesita pasar los datos mediante mensajes. Así, cuando la información que esta almacenada en un nodo es requerida por otro nodo, el nodo otorgante debe mandar la información y el nodo receptor debe recibirla. En la IBM SP y Cray T3D estos mensajes pueden comunicar información entre nodos y sincronizar actividades.

PVM ("Parallel Virtual Machine") es un sistema que permite que una colección heterogénea de computadoras en red sean usadas como un recurso computacional distribuido en el cual se puede hacer programación en paralelo (Geist *et al.*, 1994).

Esta tesis parte del trabajo de Wang y Anderson (1982), que resolvieron numéricamente por medio de elemento finito la ecuación de movimiento para el problema de advección-dispersión de un contaminante en un acuífero dentro de un campo de flujo uniforme.

Se desarrolla un programa con algunos elementos de C++. Después se escribe un programa paralelo que resuelve el mismo problema de Wang y Anderson y se implanta el programa en la red local heterogénea de estaciones de trabajo del Departamento de Oceanografía Física de CICESE. Más que buscar la más alta eficiencia de los métodos, se busca el buen funcionamiento del programa paralelo.

II Antecedentes

La literatura es vasta en cuanto al problema de advección y dispersión en acuíferos, y sobre el método de elemento finito. Si bien PVM es una tecnología reciente, ya hay varios trabajos en matemáticas aplicadas aunque relativamente pocos en aplicaciones científicas.

2.1 Advección-Dispersión

Si C es la concentración del soluto en unidades de masa por unidad de volumen de agua, entonces el flujo dispersivo de masa será

$$\left.\begin{aligned} f_x* &= nCv_x* \\ f_y* &= nCv_y* \end{aligned}\right\}, \tag{1}$$

donde f_x* y f_y* son las componentes cartesianas del flujo local dispersivo, v_x* y v_y* son las componentes de la velocidad lineal y n es la porosidad del acuífero.

Si se asume que el flujo dispersivo es proporcional al gradiente de concentración en la dirección del flujo por la porosidad, entonces

$$Cv_x* = -D_L \frac{\partial C}{\partial x}, \tag{2}$$

donde D_L es la componente longitudinal del coeficiente de dispersión.

En el caso de la componente transversal se tiene

$$Cv_y* = -D_T \frac{\partial C}{\partial y}, \tag{3}$$

donde D_T es la componente transversal del coeficiente de dispersión.

El flujo advectivo del soluto C esta definido como el transporte del soluto a una velocidad $\bar{v}_x$ conservando masa, en la forma

$$\bar{f}_x = nC\bar{v}_x. \tag{4}$$

El flujo total es la suma de los flujos advectivos y dispersivos, como estamos considerando un flujo uniforme en la dirección x, se puede expresar como

$$f_x = n(C\bar{v}_x + Cv_x^*) \; , \tag{5}$$

donde f_x es la componente en la dirección x del flujo total y $\bar{v}_x$ es la velocidad promedio en la dirección x en el acuífero. De la misma manera en la dirección y se tiene

$$f_y = nCv_y^* \; , \tag{6}$$

donde f_y es la componente en la dirección y del flujo total.

Conservación de masa implica que la divergencia del flujo sea igual a la razón de decrecimiento de la concentración de soluto por unidad de volumen del acuífero, esto es

$$\nabla \cdot f = -n\frac{\partial C}{\partial t} \; , \tag{7}$$

donde ∇ es el operador nabla bidimensional en coordenadas cartesianas, f es el vector de flujo total y t el tiempo.

Usando las Ecs. (2)-(6), además de considerar el coeficiente de dispersión y de porosidad constantes, se obtiene

$$D_L\frac{\partial^2 C}{\partial x^2} + D_T\frac{\partial^2 C}{\partial y^2} - \bar{v}_x\frac{\partial C}{\partial x} = \frac{\partial C}{\partial t}, \tag{8}$$

la cual es una versión modificada de la expresión general no-lineal de advección-dispersión.

Una vez planteada la ecuación de advección-dispersión, lo siguiente es obtener su solución, que se puede obtener por dos métodos: analíticos y numéricos. Ogata y Banks (1961) resolvieron analíticamente la ecuación (8).

Consideramos un campo de flujo en la dirección de x, donde v_x es la velocidad lineal promedio en la dirección de x. Para un tiempo menor o igual a cero, la concentración en el acuífero es cero. Para tiempos mayores que cero, la concentración en la frontera izquierda del acuífero se vuelve una constante C_0. La Ec. (8) es la ecuación de movimiento para este problema, las condiciones iniciales y de frontera para el problema son:

Condiciones iniciales: $C(x,0) = 0$ para toda x

Condiciones de frontera: $C(0,t) = C_0$ para $t > 0$

$C(\infty, t) = 0$ para $t < 0$

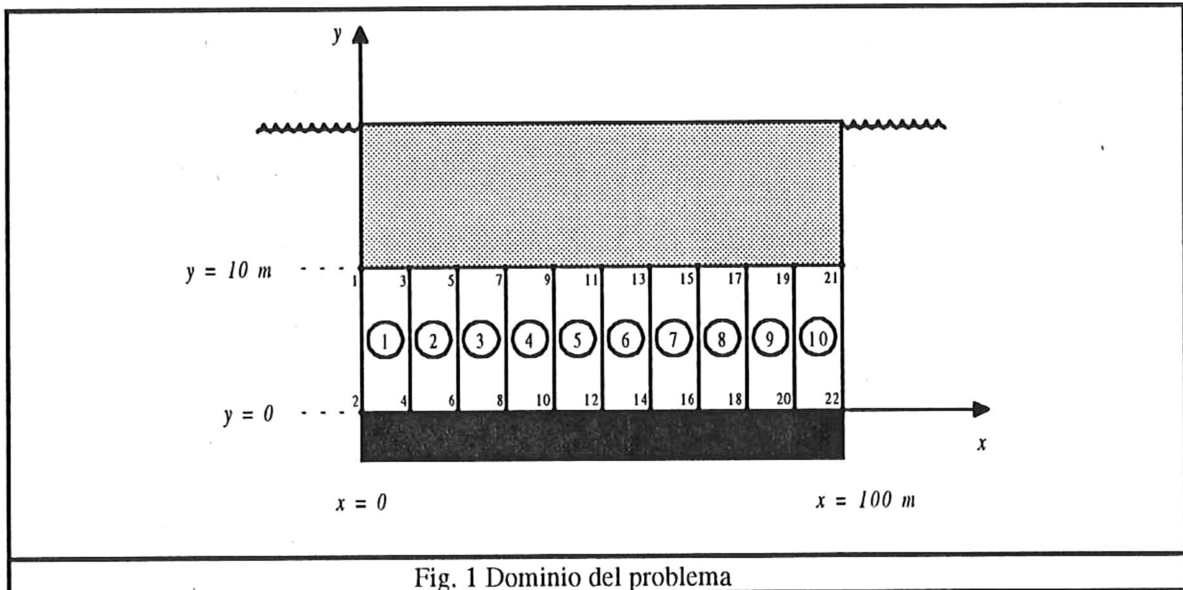


Fig. 1 Dominio del problema

2.2 Solución Analítica

Con las condiciones dadas en el párrafo anterior, Ogata y Banks (1961) derivaron la solución analítica en la forma

$$C(x,t) = \frac{C_0}{2}\left\{\exp\left(\frac{\bar{v}_x x}{D_L}\right) erfc\left(\frac{x + \bar{v}_x t}{2\sqrt{D_L t}}\right) + erfc\left(\frac{x - \bar{v}_x t}{2\sqrt{D_L t}}\right)\right\}, \tag{9}$$

donde $\bar{v}_x$ es la velocidad lineal promedio, D_L es la componente longitudinal del coeficiente de dispersión y

$$erfc(z) = \frac{2}{\sqrt{\pi}} \int_z^{\infty} e^{-u^2} du, \tag{10}$$

es la función error complementaria. En este caso $D_T = 0$.

2.3 Solución Numérica

2.3.1 Elemento finito

El primero en aplicar el método de elemento finito en el estudio de acuíferos fue Zienkiewickz y su equipo en 1955 (Kazda, 1990). Este método es apropiado en el análisis de problemas en los cuales las fronteras son irregulares o el medio es heterogéneo o anisotrópico (Hantush y Mariño, 1995). También es útil en problemas acoplados o en problemas de fronteras en movimiento (Wang, 1982).

a) Método de Galerkin

El método de Galerkin se basa en el principio de residuos ponderados, esto es, una cantidad se minimiza a lo largo del dominio del problema. El principio se expresa directamente en terminos de la ecuación diferencial. El residuo es una medida del grado de ajuste con la cantidad que no cumple con la ecuación en cada punto del dominio. La meta es minimizar el error.

La Ec. (8) se escribirá como

$$\iint_D \left(D_L \frac{\partial^2 \hat{C}}{\partial x^2} + D_T \frac{\partial^2 \hat{C}}{\partial y^2} - \bar{v}_x \frac{\partial \hat{C}}{\partial x} - \frac{\partial \hat{C}}{\partial t} \right) N_L(x,y)dxdy = 0, \tag{11}$$

donde $\hat{C}$ es la función a minimizar, $N_L(x,y)$ es la función de peso asociada al nodo L y D es el dominio del problema.

Integrando la Ec. (11) por partes obtenemos

$$\iint_D \left(D_L \frac{\partial \hat{C}}{\partial x}\frac{\partial N_L}{\partial x} + D_T \frac{\partial \hat{C}}{\partial y}\frac{\partial N_L}{\partial y} + \bar{v}_x \frac{\partial \hat{C}}{\partial x} N_L + \frac{\partial \hat{C}}{\partial t} N_L \right) dxdy = \int_\Gamma \left(D_L \frac{\partial \hat{C}}{\partial x} n_x + D_T \frac{\partial \hat{C}}{\partial y} n_y \right) N_L d\sigma, \tag{12}$$

donde el lado derecho de la ecuación es una integral de línea a lo largo de la frontera del acuífero, D_L y D_T son las componentes longitudinal y transversal del coeficiente de dispersión, Γ es la frontera del dominio, n_x y n_y son las componentes de un vector unitario normal a la frontera, y σ es una variable de integración representando la distancia a lo largo de la frontera en dirección contraria a las manecillas del reloj.

b) Elemento Rectangular

La forma de elemento más sencilla es la de un rectángulo de lados paralelos a los ejes x y y.

En este caso, el elemento rectangular tiene una longitud 2a y una altura 2b. La ecuación de interpolación se puede escribir en términos de las coordenada locales,

$$C = a_1 + a_2 x + a_3 y + a_4 xy , \tag{13}$$

donde a_i son las funciones de peso para elementos rectangulares, y se calculan como sigue: el elemento rectangular e esta especificado por sus cuatro nodos i, j, m y n, y son numerados en orden contrario a las manecillas del reloj empezando de la esquina inferior izquierda. Estos elementos están orientados paralelos a los ejes

coordenados y están centrados en el origen. Las funciones de peso $N^e_L(x, y)$ definen la solución de prueba $\hat{C}^e(x,y,t)$ sobre el elemento, es decir

$$\hat{C}^e(x,y,t) = N_i^e(x,y)C_i(t) + N_j^e(x,y)C_j(t) + N_m^e(x,y)C_m(t) + N_n^e(x,y)C_n(t), \quad (14)$$

donde

$$\left.\begin{aligned} N_i^e(x,y) &= \frac{1}{4}\left(1-\frac{x}{a}\right)\left(1-\frac{y}{b}\right) \\ N_j^e(x,y) &= \frac{1}{4}\left(1+\frac{x}{a}\right)\left(1-\frac{y}{b}\right) \\ N_m^e(x,y) &= \frac{1}{4}\left(1+\frac{x}{a}\right)\left(1+\frac{y}{b}\right) \\ N_n^e(x,y) &= \frac{1}{4}\left(1-\frac{x}{a}\right)\left(1+\frac{y}{b}\right) \end{aligned}\right\}, \quad (15)$$

y además

$$\left.\begin{aligned} 2a &= x_j - x_i = x_m - x_n, \\ 2b &= y_n - y_i = y_m - y_j. \end{aligned}\right\}. \quad (16)$$

Cada función de interpolación tiene la propiedad de que $N_L^e(x,y)$ es 1 en el nodo L y 0 en los otros tres. Sustituyendo la Ec. (13) en la Ec. (11) e integrando en cada elemento resulta un sistema de ecuaciones, el cual se puede escribir en forma matricial como

$$[G]\{C\} + [U]\{C\} + [P]\left\{\frac{\partial C}{\partial t}\right\} = \{f\}, \quad (17)$$

donde $\{C\}$ es la matriz columna de las concentraciones nodales, $\{\partial C/\partial t\}$ es la matriz columna de la derivada temporal de las concentraciones nodales, las matrices $[G]$, $[U]$, y $[P]$ corresponden a los términos individuales en la integral del lado izquierdo de la Ec. (11). La matriz columna $\{f\}$ corresponde a la integral de frontera en el lado derecho.

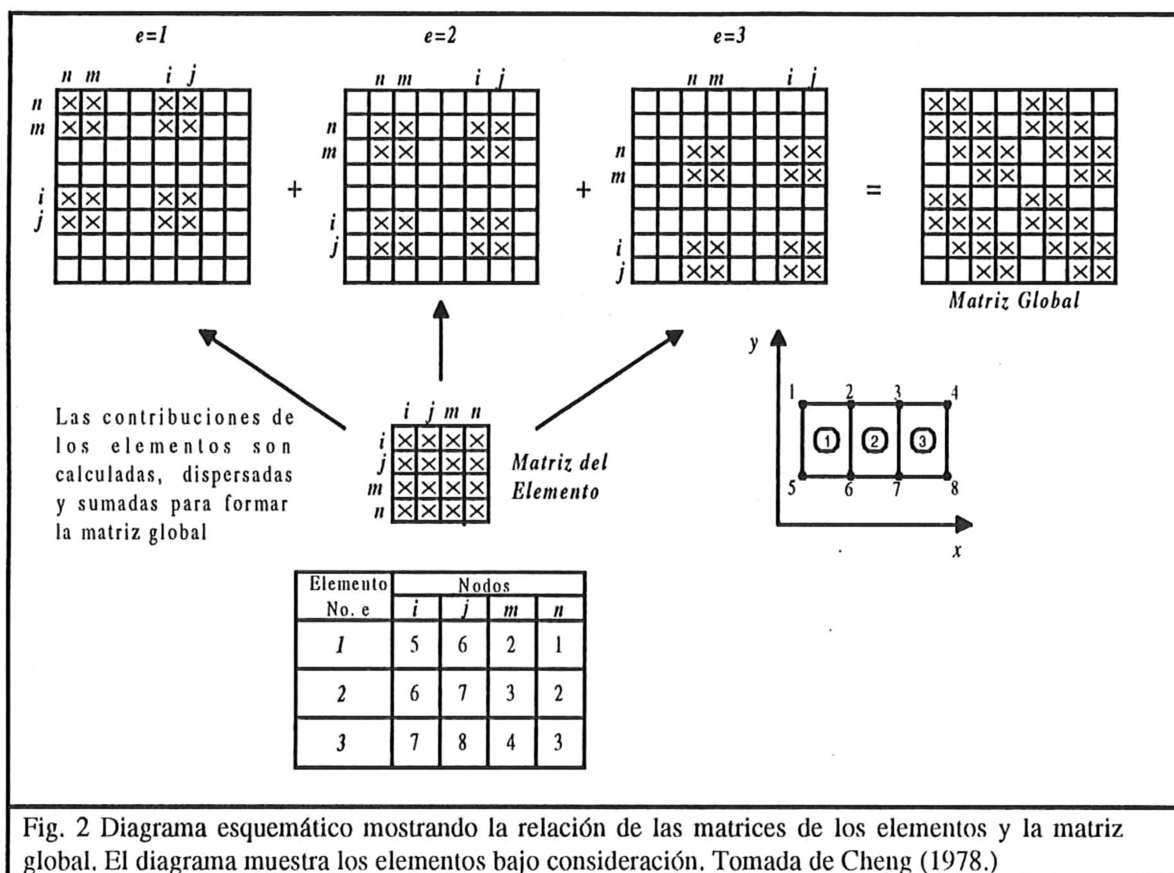
Las matrices [G], [P] y [U] son llamadas matrices globales, y se forman de

$$\left.\begin{aligned}
G^e_{L,i} &= \int_{-a}^{a}\int_{-b}^{b}\left(D_L \frac{\partial N^e_i}{\partial x}\frac{\partial N^e_L}{\partial x} + D_T \frac{\partial N^e_i}{\partial y}\frac{\partial N^e_L}{\partial y}\right)dxdy \\
G^e_{L,j} &= \int_{-a}^{a}\int_{-b}^{b}\left(D_L \frac{\partial N^e_j}{\partial x}\frac{\partial N^e_L}{\partial x} + D_T \frac{\partial N^e_j}{\partial y}\frac{\partial N^e_L}{\partial y}\right)dxdy \\
G^e_{L,m} &= \int_{-a}^{a}\int_{-b}^{b}\left(D_L \frac{\partial N^e_m}{\partial x}\frac{\partial N^e_L}{\partial x} + D_T \frac{\partial N^e_m}{\partial y}\frac{\partial N^e_L}{\partial y}\right)dxdy \\
G^e_{L,n} &= \int_{-a}^{a}\int_{-b}^{b}\left(D_L \frac{\partial N^e_n}{\partial x}\frac{\partial N^e_L}{\partial x} + D_T \frac{\partial N^e_n}{\partial y}\frac{\partial N^e_L}{\partial y}\right)dxdy
\end{aligned}\right\}, \tag{18}$$

$$\left.\begin{aligned}
P^e_{L,i} &= \int_{-a}^{a}\int_{-b}^{b} N^e_i N^e_L dxdy \\
P^e_{L,j} &= \int_{-a}^{a}\int_{-b}^{b} N^e_j N^e_L dxdy \\
P^e_{L,m} &= \int_{-a}^{a}\int_{-b}^{b} N^e_m N^e_L dxdy \\
P^e_{L,n} &= \int_{-a}^{a}\int_{-b}^{b} N^e_n N^e_L dxdy
\end{aligned}\right\}, \tag{19}$$

$$\left.\begin{aligned}
U^e_{L,i} &= \bar{v}_x \int_{-a}^{a}\int_{-b}^{b} \frac{\partial N^e_i}{\partial x} N^e_L dxdy \\
U^e_{L,j} &= \bar{v}_x \int_{-a}^{a}\int_{-b}^{b} \frac{\partial N^e_j}{\partial x} N^e_L dxdy \\
U^e_{L,m} &= \bar{v}_x \int_{-a}^{a}\int_{-b}^{b} \frac{\partial N^e_m}{\partial x} N^e_L dxdy \\
U^e_{L,n} &= \bar{v}_x \int_{-a}^{a}\int_{-b}^{b} \frac{\partial N^e_n}{\partial x} N^e_L dxdy
\end{aligned}\right\}. \tag{20}$$

La generación de las matrices globales [G], [P] y [U] se hace según el elemento que se calcule y el resultado se inyecta a la matriz global organizándose en la matriz según los subíndices de los nodos como se muestra en la Figura 2.



Elemento No. e	Nodos i	j	m	n
1	5	6	2	1
2	6	7	3	2
3	7	8	4	3

Fig. 2 Diagrama esquemático mostrando la relación de las matrices de los elementos y la matriz global. El diagrama muestra los elementos bajo consideración. Tomada de Cheng (1978.)

C) Cuadratura Gaussiana

Las integrales en las Ecs. (18)-(20) se pueden hacer analíticamente. Los integrandos son polinomios en x y y, y las soluciones son fáciles de obtener. La integración numérica mediante la cuadratura Gaussiana es fácil de codificar en un programa y nos lleva, en este caso, a una solución exacta.

Consideremos primero la cuadratura gaussiana en una dimensión. Sea $g(\xi)$ una función definida en $-1 \leq \xi \leq 1$. entonces se aproxima a la integral por medio de una suma ponderada a lo largo de un número finito de puntos

$$\int_{-1}^{1} g(\xi) d\xi = \sum_{i=1}^{n} W_i g(\xi), \tag{21}$$

donde W_i es el factor de peso requerido en el punto de Gauss ξ_i, $g(\xi)$ es un polinomio con términos a la ξ^2, la ecuación es una igualdad exacta tomando $n = 2$, $\xi_1 = -1/\sqrt{3} = -0.57735$, $\xi_2 = 1/\sqrt{3} = 0.57735$, y $W_1 = W_2 = 1$.

La solución de la integral doble es de la forma siguiente

$$\int_{-1}^{1}\int_{-1}^{1} g(\xi,\eta)d\xi\, d\eta = g(\xi_1,\eta_1) + g(\xi_2,\eta_1) + g(\xi_1,\eta_2) + g(\xi_2,\eta_2) \quad (22)$$

Las integrales de la Ec. (17) se pueden poner en la forma de la ecuación anterior a través del cambio de variables siguiente,

$$\xi = \frac{x}{a} \text{ y } \eta = \frac{y}{b} \quad (23)$$

Entonces, $dx = ad\xi$ y $dy = bd\eta$. Los límites de la integración se transforman en -1 a 1 y las funciones de interpolación en las coordenadas (ξ,η) son

$$\left.\begin{aligned} \tilde{N}_i^e(\xi,\eta) &= \tfrac{1}{4}(1-\xi)(1-\eta) \\ \tilde{N}_j^e(\xi,\eta) &= \tfrac{1}{4}(1+\xi)(1-\eta) \\ \tilde{N}_m^e(\xi,\eta) &= \tfrac{1}{4}(1+\xi)(1+\eta) \\ \tilde{N}_n^e(\xi,\eta) &= \tfrac{1}{4}(1-\xi)(1+\eta) \end{aligned}\right\} \quad (24)$$

Resumiendo, las integrales de las Ecs. (17) se transforman de acuerdo a

$$\int_{-a}^{a}\int_{-b}^{b} g(x,y)dxdy = ab\int_{-1}^{1}\int_{-1}^{1} \tilde{g}(\xi,\eta)d\xi d\eta \quad (25)$$

donde $\tilde{g}(\xi,\eta)$ es $g(x, y)$ expresado en las coordenadas (ξ,η).

2.3.2 Condiciones a la frontera

De la Ec.(17) se observa que $\{f\}$ es igual a la integral de línea de la Ec. (12). Se toman las siguientes condiciones a la frontera:

1) $f_L=0$ para todos los puntos interiores en la malla,

2) f_L=0 para nodos con fronteras donde no hay flujo, y

3)f_L es irrelevante para nodos que tengan concentraciones de soluto constantes en el tiempo si se elimina la L-ésima ecuación.

2.3.3 Solución de la ecuación diferencial matricial

La Ec. (17) es una ecuación diferencial matricial de primer orden. Para resolverla se toma una aproximación por medio de diferencias finitas, en notación matricial

$$\left\{\frac{\partial C}{\partial t}\right\} = \frac{1}{\Delta t}\left(\{C\}^{t+\Delta t} - \{C\}^{t}\right), \tag{26}$$

donde el índice representa la concentración en los tiempos t y $t+\Delta t$, Δt es el paso en el tiempo. La Ec. (26) es una columna donde sus elementos son los cambios de la concentración con respecto al tiempo, $C_L=C_L(t)$ es la concentración al tiempo t en el nodo L. La derivada en el tiempo en un nodo en particular es

$$\frac{\partial C_L}{\partial t} = \frac{C_L^{t+\Delta t} - C_L^{t}}{\Delta t}, \tag{27}$$

Si C es la aproximación de la concentración al tiempo $t + \Delta t$, entonces la Ec. (17) esta dada por

$$\left([G]+[U]+\frac{1}{\Delta t}[P]\right)\{C\}^{t+\Delta t} = \frac{1}{\Delta t}[P]\{C\}^{t} + \{f\}. \tag{28}$$

Si

$$\{B\} = \frac{1}{\Delta t}[P]\{C\}^{t} \tag{29}$$

la L-ésima columna de $\{B\}$ puede calcularse explícitamente porque conocemos las concentraciones al tiempo t, utilizando la Ec. (28) tenemos

$$\left([G]+[U]+\frac{1}{\Delta t}[P]\right)\{C\}^{t+\Delta t}=\{B\}+\{f\}, \tag{30}$$

y utilizando las condiciones de frontera llegamos a

$$(G_{i,L}+U_{i,L}+\tfrac{1}{\Delta t}P_{i,L})C_L^{t+\Delta t}=B_L+f_L. \tag{31}$$

Como se puede observar la Ec. (31), es un sistema de ecuaciones de la forma $Ax=b$; el cual se puede resolver por métodos iterativos o directos.

a) Métodos Iterativos

1. Jacobi

El método consiste en resolver la i-ésima ecuación de $Ax = b$ para x_i , siempre y cuando $a_{ii} \neq 0$ en la forma,

$$C_i=\frac{-\sum_{L=1}^{L=i-1}(G_{i,L}+U_{i,L}+\frac{1}{\Delta t}P_{i,L})C_L^t-\sum_{L=i+1}^{n}(G_{i,L}+U_{i,L}+\frac{1}{\Delta t}P_{i,L})C_L^t+B_L+f_L}{G_{i,i}+U_{i,i}+\frac{1}{\Delta t}P_{i,i}} \tag{32}$$

El método se puede escribir en la forma $C^{t+\Delta t} = T\,C^t + d$. Sea A ($= G_{i,L} + U_{i,L} + P_{i,L}/\Delta t$) dividida en su parte diagonal (D), y su parte fuera de la diagonal. Ahora la parte fuera de la diagonal se divide en la parte triangular inferior de A ($-L$) y su parte triangular superior de A ($-U$). Con esta notación

$$C = D^{-1}(L+U)C + D^{-1}b \tag{33}$$

que en su forma iterativa es

$$C^{t+\Delta t} = D^{-1}(L+U)C^t + D^{-1}b \tag{34}$$

donde $b = \{B\} + \{f\}$.

2. Gauss-Seidel

Un análisis de la Ec. (32) sugiere una mejora. Para calcular $C_i^{t+\Delta t}$, se usan las componentes C^t. Como, para $i > 1$, $C_1^{t+\Delta t}, \dots, C_{i-1}^{t+\Delta t}$ ya han sido calculadas y supuestamente son mejores aproximaciones a la solución real $C_1, \dots, C_{i-1}$ que $C_1^t, \dots, C_{i-1}^t$,

parece razonable calcular $C_i^{t+\Delta t}$ usando los valores calculados más recientemente; es decir,

$$C_i = \frac{-\sum_{L=1}^{L=i-1}\left(G_{i,L}+U_{i,L}+\frac{1}{\Delta t}P_{i,L}\right)C_L^{t+\Delta t} - \sum_{L=i+1}^{n}\left(G_{i,L}+U_{i,L}+\frac{1}{\Delta t}P_{i,L}\right)C_L^{t} + B_L + f_L}{G_{i,i}+U_{i,i}+\frac{1}{\Delta t}P_{i,i}}, \quad (35)$$

donde B_L es $B_L = \frac{1}{\Delta t}\sum_{jj=1}^{n} P_{L,jj}C_{jj}^{t}$. Este es el método que Wang y Anderson (1982) utilizaron para resolver el sistema de ecuaciones. En forma matricial

$$C^{t+\Delta t} = (D-L)^{-1}UC^{t} + (D-L)^{-1}b. \quad (36)$$

3. Gradiente Conjugado

Este es un método de minimización. Para una matriz simétrica positiva A, la x que minimiza la función cuadrática $q(x)=(1/2)x^{T}Ax - x^{T}b$, solución del sistema $Ax=b$. La razón es que el gradiente de $q(x)$ es $Ax=b$, el cual es cero cuando $q(x)$ es mínimo.

El método de Gradiente Conjugado consta de una iteración de la forma

$$C(t+\Delta t) = C(t) + \alpha(t)d(t), \quad (37)$$

donde el nuevo valor de {C} es función del valor anterior de {C}, un escalar α, y un vector de dirección $\boldsymbol{d}$.

Antes de la primera iteración se deben indicar los valores de $x(0)$, $d(0)$ y $q(0)$. El algoritmo según Shewchuk (1994) es el siguiente:

dado A, b, un valor inicial de x y una tolerancia del error de $\varepsilon<1$, se tiene

$$i \Leftarrow 0$$
$$r \Leftarrow b - Ax$$
$$\delta_{new} \Leftarrow r^{T}r$$
$$\delta_0 \Leftarrow \delta_{new}$$
Mientras $i < i_{max}$ y $\delta_{new} > \varepsilon^2\delta_0$ haz
$$q \Leftarrow Ad$$
$$\alpha \Leftarrow \frac{\delta_{new}}{d^{T}q}$$

$$
\begin{aligned}
&x \Leftarrow x + \alpha d \\
&\text{Si } i \text{ es divisible por 50} \\
&\quad r \Leftarrow b - Ax \\
&\text{de otra forma} \\
&\quad r \Leftarrow r - \alpha q \\
&\delta_{\text{old}} \Leftarrow \delta_{new} \\
&\delta_{\text{new}} \Leftarrow r^T r \\
&\beta \Leftarrow \frac{\delta_{new}}{\delta_{old}} \\
&d \Leftarrow r + \beta d \\
&i \Leftarrow i + 1
\end{aligned} \tag{38}
$$

donde i es el número de iteración, r es el residuo, d es un vector , α es un escalar que indica el tamaño del paso, δ es el gradiente al cuadrado, $i_{\max}$ es el número de iteraciones máxima, ε es el criterio de convergencia, q y β variables temporales.

El problema contiene condiciones de Dirichlet, al aplicarse al método de gradiente conjugado se deben excluir los renglones correspondientes a los puntos en donde dichas condiciones se cumplen. Y en los renglones restantes los coeficientes correspondientes a estos puntos deben pasar al lado derecho de la Ec. (31). Para así formar un sistema con los puntos que no tiene condiciones de Dirichlet.

2.4 PVM

El proyecto PVM inicio en el verano de 1989 en los laboratorios del Oak Ridge National Laboratory en el estado de Tennessee, E.U. (Geist *et al.*, 1994), desde entonces sus diseñadores han recibido la ayuda de la National Science Fundation y de varias universidades estadounidenses. Ewing *et al.* (1993) mostraron la utilidad de PVM en la solución numérica y la visualización de la propagación de ondas sísmicas en la corteza terrestre.

PVM es una biblioteca de intercambio de mensajes que permite a una computadora UNIX en red trabajar como un sistema paralelo de memoria distribuída.

Automáticamente se puede ejecutar una tarea en la computadora más apropiada, también el usuario puede indicar que computadora es la que realizará una cierta tarea en particular, en el caso de una red con diferentes tipos de máquinas. En todas estas modalidades, PVM se encarga de las conversiones de datos necesarias entre computadoras, así como de la comunicación entre ellas.

PVM es muy flexible, soporta la forma más general de computación paralela, el paradigma MIMD (Multiple Instructions, Multiple Data), lo cual permite el uso de cualquier otro paradigma paralelo. Las estructuras de control se pueden implantar con las funciones debidas de PVM. Los programas del usuario pueden tener acceso a PVM por medio de las bibliotecas de rutinas FORTRAN ó C de PVM. Este trabajo utiliza la red de computadoras SUN-4 del departamento de Oceanografía Física del CICESE, empleando las bibliotecas de PVM en lenguaje C++.

2.4.1 Rendimiento

Los dos terminos más comunes para evaluar el rendimiento de un programa paralelo se llaman incremento de la rapidez de procesamiento o "*speed up*" y eficiencia.

El incremento de la rapidez de procesamiento describe la calidad de la ejecución en paralelo y se define como

$$S_p = \frac{T_1}{T_p}, \tag{39}$$

donde S_p es el incremento de la rapidez de procesamiento, T_1 es el tiempo requerido de ejecución del algoritmo secuencial, y T_p es el tiempo requerido por el algoritmo paralelo corriendo en *p* procesadores. El incremento lineal es definido como *P*, esto

es, el tiempo requerido por un procesador sera *p* veces el tiempo requerido para correrlo en *p* procesadores.

La eficiencia se define como.

$$E_p = \frac{S_p}{P}. \tag{40}$$

el "speedup" de un algoritmo paralelo corriendo en p procesadores entre el numero de procesadores.

III Metodología

C++ es una version mejorada de C, la cual permite el desarrollo de programas orientados a objetos. El alto nivel de organizacion que ofrece C++ fue la razon de haberlo preferido en lugar de FORTRAN. El trabajo de Wang y Anderson (1982) (escrito en FORTRAN) consta de tres bloques, el primero genera los valores de la malla, las coordenadas cartesianas, los valores de la concentración y las condiciones de flujo en cada punto. Finalmente indica que puntos nodales le tocan a cada elemento. El segundo bloque contiene el método de elemento finito propiamente dicho, el cual genera los valores de las matrices globales de coeficientes (G), (U) y (P) (dadas por las Ecs. (18)-(20) utilizando el método de cuadratura gaussiana) para la Ec. (17), en la cual hay una dependencia en el tiempo, después de una segunda aproximación por medio de un esquema implícito se llega al sistema de la Ec. (31). Por último se resuelve este sistema de ecuaciones (bloque 3) para cada paso del tiempo utilizando las matrices de coeficientes generadas en el bloque 2.

Al utilizar el algunos elementos de programación orientada a objetos lo que se pretende es poder manejar las partes del programa como unidades autónomas que se relacionan entre sí. En este caso se pueden identificar dos, el dominio convenido del problema y el elemento finito. La forma en la que se define un objeto es por medio de las clases.

`Programa Principal`. El programa principal es similar al de Wang y Anderson (1982). Contiene 3 bloques. El primero genera las condiciones iniciales y de frontera, el segundo la matriz global de coeficientes, y por ultimo se resuelve el sistema de ecuaciones por medio del metodo iterativo de Jacobi, Gauss-Seidel ó Gradiente Conjugado.

Se han definido dos objetos, el objeto *Dominio* y el objeto *Elemento.* El objeto *Dominio* es el encargado de crear la matriz global y de asignar los valores a los elementos, número en la malla de los nodos del elemento y posición xy de los nodos del elemento. El objeto *elemento* genera la matriz local de conductancia del elemento, efectúa los pasos del método de elemento finito. La definición de éstos dos objetos se encuentran en los archivos de encabezado *"dom_ad.hpp"* y *"ele_ad.hpp"*, el programa *Adjectiv.cxx* es en donde se generan las condiciones iniciales y se soluciona el sistema de ecuaciones por el método de Gauss-Seidel.

`Estructura Punto.` Aquí se genera el tipo de dato llamado *Punto* el cual almacena el número del nodo, su posición (x,y) y la condición de flujo.

`Objeto Elemento.` La clase Elemento, es la encargada de formar la matriz local de coeficientes a partir de las Ecs. (18)-(20). Se le asignan sus cuatro puntos nodales y este forma la matriz. Para trabajar con elementos rectangulares lineales se necesitan definir cuatro puntos i, j, m y n, empezando desde la esquina inferior izquierda moviendose en dirección contraria a las manecillas del reloj. Se utiliza la estructura *Punto* para trabajar con ellos. El objeto *Dominio* es el que asigna puntos al elemento, esto lo hace utilizando la función ***Asignar()***, la cual recibe los cuatro nodos que le corresponden. La función ***Matriz()*** genera las matrices de coeficientes de cada elemento, dentro de esta función es donde se realiza el trabajo de interpolación y la integración Gaussiana, lo cual genera 3 matrices G, U y P de dimension 4x4.

`Objeto Dominio.` La clase Dominio es la encargada de generar la malla, condiciones a la frontera y asignar los nodos a los elementos finitos. Después que el objeto elemento genera su matriz local, el objeto Dominio acomoda los valores en la matriz global. Por último se resuelve el sistema de ecuaciones que indica la Ec. (31)

por medio del método Jacobi, Gauss-Seidel ó Gradiente Conjugado. El objeto dominio conoce de cuantos elementos consta el dominio y también como están organizados los puntos en la malla. Es el que contiene las matrices globales de coeficientes y genera los resultados de los diferentes elementos. La función ***Asig()*** toma las coordenadas de los puntos del dominio así como la matriz NODE que contiene la distribución de los nodos en los diferentes elementos para formar los puntos i, j , m y n de cada elemento. Al asignarle a cada elemento sus nodos, se generan la matriz local correspondiente de coeficientes. *Dominio* se encarga de almacenar los valores en la matriz global. Las funciones Jacobi(), Gauss_Seidel() y GradienteConjugado() resuelven el sistema de ecuaciones. El método de Gradiente Conjugado se programó según Shewchuk(1994). El usuario elije la función (método) a su conveniencia.

Después de haber cotejado los resultados del programa secuencial C++ con la solución analítica, se procede a paralelizar dicho programa.

Después de que se generó la malla con sus condiciones a la frontera y la asignación de los nodos a los elementos, el proceso de paralización reparte el número de elementos entre el número de máquinas disponibles. Una máquina lleva a cabo las funciones de repartir los datos y recibir los resultados de las otras, a esto se le conoce como el esquema **amo-esclavo** ("master-slave''). El **amo** reparte un cierto número de elementos entre cada **esclavo**, creando así un subdominio del dominio convenido. Después cada esclavo se dedica a generar la matriz local de coeficientes para cada elemento. Una vez que han sido resueltas las Ecs. (18)-(20), los resultados se devuelven al amo el cual se encarga de acomodarlos en la matriz global (Fig 2). Enseguida se resuelve la Ec. (31), para cada paso en el tiempo. El amo servirá de

cronometrador, diciéndole a cada esclavo en que paso del tiempo se está trabajando para que estos se detengan al llegar al final.

3.1 Proceso de paralelización.

Al paralelizar el programa, utilizamos la misma programacion orientada a objetos que en el programa secuencial escrito en C++. La paralelizacion se hace en la modalidad ***Amo-Esclavo***, por lo tanto hay dos programas para la solución del problema. El objeto *Dominio* está en el programa ***Amo*** y fue necesario crear el objeto *subdominio* , el cual trabaja con la parte asignada por al ***Amo*** que está en el programa *Esclavo*. En el objeto *Dominio* incluiremos las rutinas para la paralelización y el *subdominio* incluye además de dichas rutinas las funciones de los objetos *Dominio* y *elemento* del programa en serie.

El programa ***Amo*** contiene el constructor del objeto(dominio()) el cual tiene dos opciones: crear la malla ó utilizar una ya definada en un archivo externo. es CrearMalla() y CargarMalla() son las funciones que ejecutan lo anterior. pvm_hostinfo() obtiene el número de máquinas y sus arquitecturas respectivas. Inicializar() reparte el número de elementos a los esclavos. Cada esclavo calcula las matrices locales de conductancia, y estos se la envían al maestro. La función *RecibirdeNodos()* forma la matriz global. Así se construye Ec. 27. La solución se obtendrá utilizando los metodos iterativos de Jacobi y Gradiente Conjugado.

3.1.1 Paralelización del método iterativo de Jacobi.

Tenemos de la Ec.(26) que (A)$\{C\}^{t+\Delta t} = \{B\}$, donde

$$\left.\begin{aligned} [A] &= [G] + [U] + [P] / \Delta t \text{ y} \\ \{B\} &= [P]\{C\}^{t} / \Delta t. \end{aligned}\right\} \qquad (38)$$

Cada procesador es responsable de un cierto número de renglones de A, así como de los elementos correspondientes de B. Con esta descomposición de datos, la interacción entre los procesadores ocurre en dos lugares, durante cada iteración del ciclo **"do...while"**. Primero, cada procesador deberá tener una copia del vector {C} calculado en la iteración anterior, cada procesador deberá transmitir los elementos de {C} que halla calculado a todos los demás procesos; para esto hacemos primero que cada esclavo se los transmita al maestro y este después de haber recibido de todos, le transmita el vector {C} completo a cada uno de ellos. Segundo, cada procesador calculará un valor del error basado en el cambio de valor en sus propios elementos de {C}. Se necesita encontrar el error máximo, el esclavo manda al maestro el resultado del error que obtuvo, el maestro encuentra el error máximo y se los regresa a cada uno de ellos, cada procesador tendrá el mismo valor para el error, lo cual permitirá a todos salir del ciclo **"do...while"** al mismo tiempo.

3.1.2 Paralelización del método iterativo de Gauss-Seidel.

Como mencionamos en la sección 1.4.3, Gauss-Seidel resuelve Ax=b tomando en cuenta los valores ya calculados para los demas puntos. Si la matriz es muy densa hay una componente secuencial significativa, por que el valor de $x_i(t+\Delta t)$ debe determinarse antes de determinar el calculo de $x_{i+1}(t+\Delta t)$ para todas las $i > 1$.

Si A es poco densa (el cual es nuestro caso) es más predispuesta a la paralelización. Recordemos que en la matriz de coeficientes solo los puntos de la malla que estan alrededor del nodo que estamos calculando tienen valores diferentes de cero. Esto nos permite utilizar solo los valores de los nodos que necesitamos y al calcularse estos podemos pasar a la siguiente iteración y no esperar a que se calcule el resto.

3.1.3 Paralelización del método iterativo de Gradiente Conjugado

El método de gradiente tiene dos complicaciones al trabajar con el en paralelo, el uso de un producto interno y un producto matriz-vector. El algoritmo hace repetido uso de estas funciones para calcular el residuo, el paso en el tiempo y el vector de dirección. El programa llamará a unas funciónes que realizan estas operaciones. El maestro guardará los resultados, mientras que los esclavos se dedicarán a realizar las operaciones producto interno y producto matriz-vector. Es en estas funciones donde se realiza la paralelización.

La primera, **ProductoInterno()**, recibe por referencia a dos vectores y hace el producto interno entre ellas repartiendo el número de elementos entre el número de esclavos. Los esclavos hacen el producto y suma. El maestro recibe el valor de cada esclavo, y la función regresa un escalar al método.

En la segunda, **MatrizVector()**, recibe por referencia a una matriz a, un vector v, y un vector u. La matriz a es la que se opera con el vector v y el resultado es asignado al vector u. Esta función tambien trabaja en paralelo repartiendo los renglones de la matriz entre el número de esclavos, y una copia del vector v. El maestro en la función recibe los elementos del vector u y lo arma para obtener el resultado necesario para el método de gradiente congujado.

IV Resultados

Se compara tiempo de ejecución del método utilizando Jacobi y un número variable de nodos en la máquina virtual. El número máximo de nodos en la maquina virtual fue de 7. Para estos casos mediremos el "*speedup*" y la eficiencia de los métodos. En este primer experimento se utilizó la malla de 10 elementos utilizada por Wang y Anderson. Después aumentaremos el número de elementos refinando la malla de Wang y Anderson. El tercer experimento utilizará una malla más refinada en la horizontal y en la vertical. Aunque este último experimento no nos brindará soluciones reales, debido a que la formulación numérica de Wang y Anderson es para una dimensión, nos ayudará a tener una idea de la carga computacional del método, esperando así observar ventajas en la paralelización.

4.1 Tiempos de ejecución.

Experimento 1. El problema original de Wang y Anderson (10 elementos). Utilizando Jacobi con 1, 2, hasta 7 procesadores en la máquina virtual. En microsegundos

	2	3	4	5	6	7
Bloque 1	1818	11,401	3,430	3,947	3,912	4,464
Bloque 2	877,938	787,531	1,308,397	938,890	1,200,664	1,050,367
Bloque 3	81,442,688	63,679,300	73,549,048	70,936,256	104,233,936	87,445,448
Total	82,322,444	64,478,232	74,860,875	71,879,093	105,438,512	88,500,279

Experimento 2. Refinación de la malla de Wang y Anderson de 10 elementos a 50 y 100 elementos.

- 50 elementos.

	2	3	4	5	6	7
Bloque 1	2,899	3,490	11,156	4,265	10,688	6,266
Bloque 2	4,374,355	2,458,013	1,617,687	2,040,103	1,880,776	2,283,459
Bloque 3	17,585,430	11,838,605	12,277,107	13,418,307	18,234,108	23,182,340
Total	21,962,684	14,300,108	13,905,950	15,462,675	20,125,572	25,472,065

- 100 elementos

	2	3	4	5	6	7
Bloque 1	5,278	14,612	7,233	48,230	19,498	41,105
Bloque 2	5,623,501	3,450,336	6,029,222	4,151,679	3,009,005	3,814,448
Bloque 3	12,530,396	9,608,625	12,702,231	10,566,815	17,477,928	17,884,570
Total	18,159,175	13,073,573	18,738,686	14,766,724	20,506,431	21,740,123

Experimento 3. Refinacion de la malla de 1 x 10 a 10x10.

- 100 elementos

	2	3	4	5	6	7
Bloque 1	10,002	55,561	36,080	178,676	41,986	29,406
Bloque 2	6,129,777	5,931,905	5,634,677	4,088,916	3,816,381	3,030,669
Bloque 3	11,884,986	12,896,052	12,180,971	13,028,936	13,798,777	14,325,534
Total	18,024,765	18,883,518	17,851,728	17,296,528	17,657,144	17,385,609

4.2 Rendimiento:

Experimento 0. Los tiempos de ejecución de los programas secuenciales.

	Experimento 1	Experimento 2		Experimento 3
tiempo	572,825	6,806453	28,277,522	27,991,964

Experimento 1. (Rendimiento).

No. Procs.	Speedup
2	0.006958
3	0.008884
4	0.007652
5	0.007969
6	0.005433
7	0.006473

Experimento 2.(Rendimiento).

- 50 elementos

No. Procs.	Speedup
2	0.309910
3	0.475972
4	0.489463
5	0.440186
6	0.338199
7	0.267212

- 100 elementos

No. Procs.	Speedup
2	1.557203
3	2.162953
4	1.509225
5	1.915727
6	1.378959
7	1.300707

Experimento 3. (Rendimiento).

No. Procs.	Speedup
2	1.552972
3	1.482349
4	1.568025
5	1.618357
6	1.585305
7	1.610065

V Discusión

En este trabajo se discute el rendimiento de PVM con el metodo de Jacobi, la utilización de elementos de programacion orientada a objetos y la programación paralela de la ecuación de advección-dispersión del transporte de solutos en un acuífero homogeneo utilizando PVM. Wang y Anderson (1982) utilizaron el método de Galerkin con elementos rectangulares para resolver este problema; en la parte temporal se usa un esquema implícito, obteniendo los mismos resultados.

Las ventajas de aplicar elementos de programación orientada a objetos en la solución de este problema radican en que de esta manera los elementos de la malla son más fáciles de manejar que utilizando un esquema no orientado a objetos. El manejar el objeto malla nos da una relación directa con el concepto de malla en elementos finitos, y por lo tanto, con el fenómeno físico en si. También se pueden utilizar elementos más generales, tales como cudriláteros, sin tener que modificar todo el código. De igual forma las modificaciones locales del programa son más fáciles, por ejemplo, resolver el sistema de ecuaciones por métodos diferentes consiste en sólo cambiar una línea de código después de haber agregado la función en la lista de los atributos públicos de la clase dominio, la cual contiene al objeto malla. Sin embargo, en este trabajo no se aplican los conceptos de herencia y polimorfismo de la programación orientada a objetos lo cual le da un alcance limitado, debido que en caso de una actualización, necesitamos modificar el código. Pero en la solución en paralelo de este diseño fue de gran ayuda debido a que en paralelo repartimos elementos en lugar de nodos.

Con la finalidad de analizar el rendimiento de PVM en la modelación de fenómenos físicos, en el problema secuencial trabajamos con los métodos Jacobi,

Gauss-Seidel y Gradiente Conjugado. La complejidad de programación secuencial aumenta en Gradiente Conjugado, pero su rendimiento es mucho mejor que los anteriores, del orden de (1:2). Siendo las soluciones obtenidas exactamente iguales para los 3 métodos.

En la Fig. 3 se muestra la solución analítica y la solución numérica, el error es mínimo, de tal forma que la solución numérica es aceptable

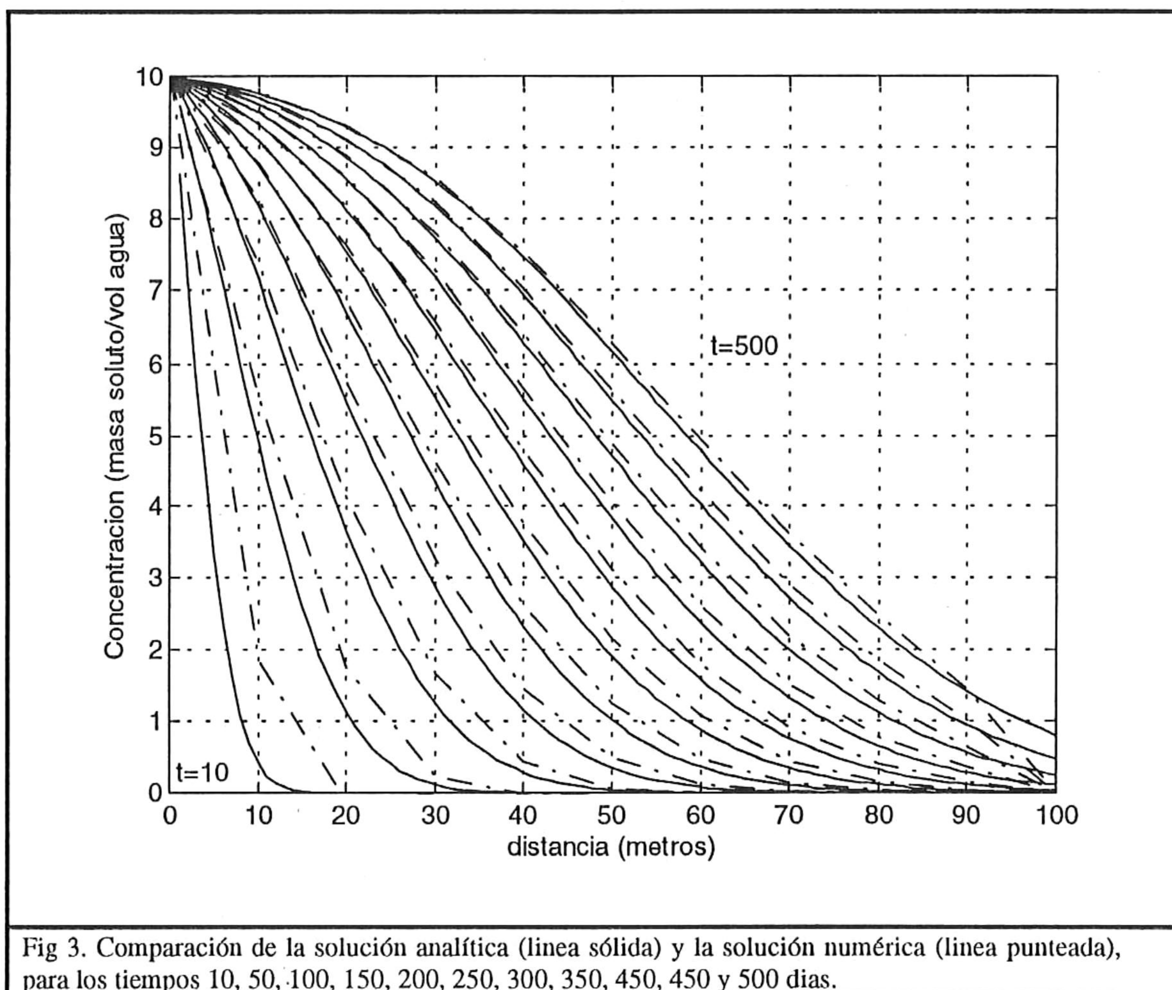


Fig 3. Comparación de la solución analítica (linea sólida) y la solución numérica (linea punteada), para los tiempos 10, 50, 100, 150, 200, 250, 300, 350, 450, 450 y 500 dias.

Al momento de paralelizar nos dedicamos a estudiar el comportamiento de los resultados al aumentar el número de procesadores en la máquina virtual y también los cambios que se presentaban cuando refinabamos la malla.

Empezamos trabajando con el dominio que utilizaron Wang y Anderson. Los tiempos de ejecución son muy altos debido a que la máquina virtual utiliza mucho tiempo en el paso de mensajes comparado con el tiempo necesario para resolver el problema en si. Al aumentar el número de elementos a 50, hay indicios de que la computación paralela está haciendo su trabajo en 3, 4 y 5 procesadores.

Al utilizar 100 elementos los resultados muestran que para 2 procesadores ya podemos hablar de un dominio en el cual la programación en paralelo es viable debido a que los resultados son mejores que al tiempo secuencial.

Aunque los resultados en el último experimento carecen de validez (debido a que nuestra aproximación algebraica de las ecuaciones es para movimiento en la dirección x) estos son utilizados para mostrar el comportamiento de la máquina virtual a una diferente distribución de los elementos en la malla de coeficientes. Trabajando con 100 elementos los resultados del bloque 2 muestran como la distribución de los elementos en los procesadores es mas amena a la paralelización cuando refinamos tambien la malla en y. Un ejemplo de la distribución de los datos en la matriz para una malla de 10 x 1 (Wang), de 10 x 2 y de 10 x 3 se muestran en la figura 4. Esta diagonalización, en la cual se forma un 'ancho de banda' es la que permite que el método de elementos finitos sea tan bueno para paralelizar.

En lo que se refiere a los resultados de solución del sistema de ecuaciones solo se pudo paralelizar el método de Jacobi, donde se puede observar que la máquina virtual es buena con 2 a 3 ó inclusive 4 procesadores. Una de las dificultades que encontré fue el método de Gauss-Seidel, el cual requiere que en vez de paralelizar la malla en si, paralelizemos el numero de iteraciones. Esto se conoce como paralelismo de control. En el código se utiliza dinamicamente la memoria, el implementar esta

tecnica costo trabajo debido que el numero de procesadores en la maquina virtual es variable, y por lo tanto no se sabe con exactitud el tamaño de los arreglos que se que son transmitidos a los esclavos. Con respecto al método de gradiente conjugado la utilización de algebra vectorial , proporcionó errores de asignación de memoria que hacia que la maquina virtual se detuviera por conflictos con el operador **new**.

La computación paralela es diferente en que tienes que pensar como aislar el proceso el cual se pueda repetir en varias máquinas con el mínimo intercambio de mensajes entre ellas. Al encontrarlo, ya solo depende de cómo se intercambiará la solución parcial entre ellas. Esto es de suma importancia cuando tenemos mallas de miles de elementos, las cuales sólo habían sido resueltas en grandes máquinas ubicadas en centros de supercómputo. PVM es una herramienta que permite aprender esta nueva forma de pensar, donde algoritmos relativamente fáciles se vuelven complicados y viceversa.

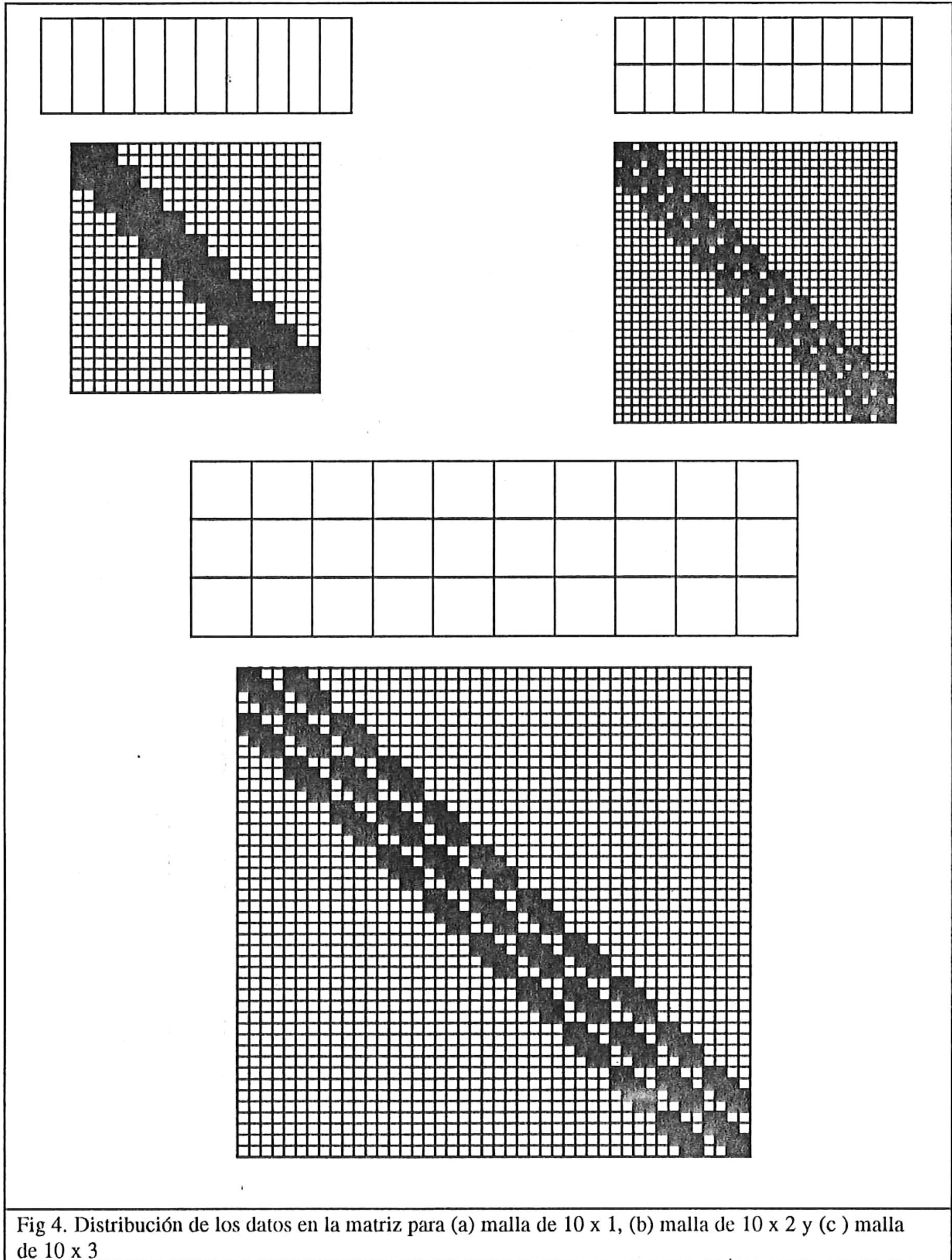


Fig 4. Distribución de los datos en la matriz para (a) malla de 10 x 1, (b) malla de 10 x 2 y (c) malla de 10 x 3

VI Conclusiones

Son pocas las aplicaciones fisicas de la programación paralela en México. Hay paises con más de 10 años de ventaja en el estudio de esta disciplina. El método de elemento finito es fácil de paralelizar y así como los resultados mejoran al refinar la malla, el tiempo de ejecución también. Despues, el tiempo de ejecución depende de que tipo de método se utilice para la solución del sistema de ecuaciónes.

Los modelos matemáticos de la fisica proveen la abstracción necesaria para desarrollar la programación orientada a objetos. El buen diseño (desde el punto de vista de la programación orientada a objetos) permitirá que la forma en que estan estructurados los programas sea similar a la forma en que estan organizados los diferrentes modelos fisicos, debido a que muchos son casos especiales de otros mas generales (herencia) ó utilizan diferentes técnicas numericas de solución (polimorfismo). Además de que proporciona un orden, el cual es necesario ya que ahora los programas que se utilizan para estudiar fenomenos más complicados de la naturaleza son proyectos muy grandes, y necesitan de mantemiento constantemente y de extensibilidad.

PVM es un medio viable, si no para la programación masiva, sí para el aprendizaje de la computación paralela.

VII Literatura Citada

Duller, A.W.G. y D.J. Paddon. 1983. Processor Arrays and the Finite Element Method, en Parallel Computing 83, M. Feilmeier, J. Joubert y U. Shendel (eds.) Elsevier Science Publishers. Holanda. p. 131-136.

Ewing, R.E., R.C. Sharpley, D. Mitchum, P. O'Leara y J.S. Sochacki. Distributed Computation of Wave Propagation Models Using PVM. Supercomputing '93 Proceedings, Portland Or.

Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek y V. Sunderam. 1994. PVM 3 User's guide and reference manual. United States Department of Energy. Oak Ridge. 161p.

Hantush, M.M. y M.A. Mariño. 1995. Continous-time stochastic analysis of groundwater flow in heterogenous aquifiers. Water Resourc. Res.

Kazda, I. 1990. Finite Element Techniques in Groundwater Flow Studies. Elsevier. Amsterdam. 313 p.

Landau, R.H., y P.J. Fink Jr. 1993. A scientist's and engineer's guide to Workstations an Supercomputers: Coping with Unix, RISC, vectors, and programming. Wiley Interscience. New York. 390 p.

Ortega, J.M. y R.G. Voigt. 1985. Solution of partial differential equations on vector and parallel computers. SIAM. Philadelphia. 99p.

Ragsdale, S. 1990. Parallel programming primer. Intel Corporation. Oregon. 122 p.

Scarbnick, C. 1994. What's the best parallel programing language ? Gather/Scater, SDSC. Vol 10 No.1.

Sim, M. y C.V. Chrysikopoulos. Analitical models for one-dimensional virus transport in saturates porous media. Water Resourc. Res. 1994.

Turner, M.J., R.W. Clough, H.C. Martin y L.C. Topp. 1956. Stiffness and Deflection Analysis of Complex Structures. J. Aeronautic Sci. Vol 23 No. 9.

Wang, H.F. y M.P. Anderson. 1982. Introduction to groundwater modelling: finite diference and finite elements methods. W.H. Freeman and Co., New York. 237 pp.

VIII Apendices o Anexos

1. Código objelem2.hxx

```
#ifndef NNODOELEM
#define NNODOELEM 4
#endif

/* OBJELEM2.HXX : Archivo de encabezado para OBJDOM2.HXX en el cual se
                  encuentran las operaciones de elemento finito, funcion de
                  interpolacion, derivadas, etc.*/

struct Punto {
 int flag;
 float x;
 float y;
};

class Elemento {
 Punto i, j, m, n;
public:
 Punto EvaluarPunto(float, float, int);
 void Asignar(struct Punto, struct Punto, struct Punto, struct Punto);
 void Matriz(float** G, float** U, float** P, int** r, int** c);
  float Nx(int,int);
  float Ny(int,int);
  float Ns(int,int);
  float Nxsi(int,int);
  float Neta(int,int);
  float XSIx(int);
  float XSIy(int);
  float ETAx(int);
  float ETAy(int);
  float Xxsi(int);
  float Xeta(int);
  float Yxsi(int);
  float Yeta(int);
  float DET(int);
  Punto operator = (struct Punto);
};

Punto Elemento::EvaluarPunto(float x, float y, int flag) {
  Punto A;
  A.x=x;  A.y=y;  A.flag=flag;
  return A;
}

void Elemento::Asignar(struct Punto I, struct Punto J, struct Punto M,
 struct Punto N) {
  i=I;  j=J;  m=M;  n=N;
}

void Elemento::Matriz(float** G, float** U, float** P, int** r, int** c) {
  int PUNTOS[NNODOELEM], R, C;
  float Vx=0.1, dx=10*Vx, dy=Vx;
  PUNTOS[0]=i.flag;  PUNTOS[1]=j.flag;  PUNTOS[2]=m.flag; PUNTOS[3]=n.flag;
  for(int I=0; I<NNODOELEM; I++) {
    R=PUNTOS[I];
    for(int J=0; J<NNODOELEM; J++) {
```

```
            C=PUNTOS[J];
            for(int K=0; K<NNODOELEM; K++) {
              G[I][J]+= (dx*Nx(J,K)*Nx(I,K)+dy*Ny(J,K)*Ny(I,K))*DET(K);
              U[I][J]+= Vx*Nx(J,K)*Ns(I,K)*DET(K);
              P[I][J]+= Ns(J,K)*Ns(I,K)*DET(K);
            }
            r[I][J]=R;
            c[I][J]=C;
    }
  }
}

float Elemento::Nx(int p, int q) {
  float nx = Nxsi(p,q)*XSIx(q)+Neta(p,q)*ETAx(q);
  return nx;
}

float Elemento::Ny(int p, int q) {
  float ny = Nxsi(p,q)*XSIy(q)+Neta(p,q)*ETAy(q);
  return ny;
}

float Elemento::Ns(int p, int q) {
  float ns;
  float xsi[4], eta[4];

  xsi[0] = -0.57735;
  xsi[1] =  0.57735;
  xsi[2] =  0.57735;
  xsi[3] = -0.57735;

  eta[0] = -0.57735;
  eta[1] = -0.57735;
  eta[2] =  0.57735;
  eta[3] =  0.57735;

  switch(p)
    {
    case(0):
            ns=0.25*(1-xsi[q])*(1-eta[q]); break;
    case(1):
            ns=0.25*(1+xsi[q])*(1-eta[q]); break;
    case(2):
            ns=0.25*(1+xsi[q])*(1+eta[q]); break;
    case(3):
            ns=0.25*(1-xsi[q])*(1+eta[q]); break;
    }
  return ns;
}

float Elemento::Nxsi(int p, int q) {
  float nxsi;
  float eta[4];

  eta[0] = -0.57735;
  eta[1] = -0.57735;
  eta[2] =  0.57735;
  eta[3] =  0.57735;

  switch(p)
    {
```

```
    case(0):
        nxsi=-0.25*(1-eta[q]); break;
    case(1):
        nxsi=0.25*(1-eta[q]); break;
    case(2):
        nxsi=0.25*(1+eta[q]); break;
    case(3):
        nxsi=-0.25*(1+eta[q]); break;
    }
  return nxsi;
}

float Elemento::Neta(int p, int q) {
  float neta;
  float xsi[4];

  xsi[0] = -0.57735;
  xsi[1] =  0.57735;
  xsi[2] =  0.57735;
  xsi[3] = -0.57735;

  switch(p)
    {
    case(0):
        neta=-0.25*(1-xsi[q]); break;
    case(1):
        neta=-0.25*(1+xsi[q]); break;
    case(2):
        neta=0.25*(1+xsi[q]); break;
    case(3):
        neta=0.25*(1-xsi[q]); break;
    }
  return neta;
}

float Elemento::XSIx(int q) {
  float xsix = Yeta(q)/DET(q);
  return xsix;
}

float Elemento::XSIy(int q) {
  float xsiy;
  xsiy = - Yxsi(q)/DET(q);
  return xsiy;
}

float Elemento::ETAx(int q) {
  float etax;
  etax = -Xeta(q)/DET(q);
  return etax;
}

float Elemento::ETAy(int q) {
  float etay;
  etay = Xxsi(q)/DET(q);
  return etay;
}

float Elemento::Xxsi(int q) {
  float xxsi;
  xxsi = Nxsi(0,q)*i.x+Nxsi(1,q)*j.x+Nxsi(2,q)*m.x+Nxsi(3,q)*n.x;
```

```
  return xxsi;
}

float Elemento::Xeta(int q) {
  float xeta;
  xeta = Neta(0,q)*i.x+Neta(1,q)*j.x+Neta(2,q)*m.x+Neta(3,q)*n.x;
  return xeta;
}

float Elemento::Yxsi(int q) {
  float yxsi;
  yxsi = Nxsi(0,q)*i.y+Nxsi(1,q)*j.y+Nxsi(2,q)*m.y+Nxsi(3,q)*n.y;
  return yxsi;
}

float Elemento::Yeta(int q) {
  float yeta;
  yeta = Neta(0,q)*i.y+Neta(1,q)*j.y+Neta(2,q)*m.y+Neta(3,q)*n.y;
  return yeta;
}

float Elemento::DET(int q) {
  float det;
  det = Xxsi(q)*Yeta(q) - Yxsi(q)*Xeta(q);
  return det;
}

Punto Elemento::operator = (struct Punto A) {
  Punto L;
  L.x=A.x;  L.y=A.y;  L.flag=A.flag;
  return L;
}
```

2. Código objdom2.hxx

```
/* OBJDOM2.HXX : Archivo de encabezado para ESCLAV2.CXX en el cual se de-
                 fine la clase Subdominio que es la que resuelve la matiz
                 de conductancia para cada uno de los elementos del sub-
                 dominio por medio del objeto Elemento (definido en el ar-
                 chivo de encabezado OBJELEM.HXX) y despues la manda al
                 programa maestro MASTER2. */
#define NNODOELEM 4
#define TOL 0.01

#include "objelem2.hxx"
#include <stdio.h>

class Subdominio {
 int me;                          // Quien soy.
 int* tids;
 int SLAVES;
 int nelemmaquina;                // Numero de elementos que le tocan.
 Elemento* e;                     // Objeto elemento.
 int* puntos;                     // Los numeros de los puntos
 float* ord;                      // Las coordenadas de los puntos
 float* abs;                      // de la malla en el mundo real.
 int*** r;                        // r y c son arreglos de matrices que
 int*** c;                        // indican las posiciones de G, U y P
 float*** G;                      // en la matriz global siendo estas
 float*** U;                      // tambien arreglos de matrices, pero
 float*** P;                      // por elemento.
```

```
public:
 Subdominio(void);
 ~Subdominio(void){pvm_exit();}
 void InicializarPVM(void);
 void AsignarDominio(void);
 void Enviar(void);
 void FreeMem(void);
 void Jacobi(void);
 void GradienteConjugado(void);
 void MatrizVector(double** M, int rM, int cM, double* x, double* Mx);
 double ProductoInterno(double *a, double* b, int sa);
};

/***************************************************************************/
/* CONSTRUCTOR Subdominio, Inicializa PVM y recibe los datos de la malla     */
/*                  que le corresponden e inicializa G, U y P.               */
/***************************************************************************/
Subdominio::Subdominio()
{
 int msgtype, dim;

 InicializarPVM();

 msgtype = 1;

 pvm_recv(-1, msgtype);
 pvm_upkint(&nelemmaquina, 1, 1);

 dim = nelemmaquina*NNODOELEM;

 e = new Elemento[nelemmaquina];

 puntos = new int[dim];
 ord = new float[dim];
 abs = new float[dim];

 pvm_upkint(puntos, dim, 1);
 pvm_upkfloat(ord, dim, 1);
 pvm_upkfloat(abs, dim, 1);

 G = new float**[nelemmaquina];
 U = new float**[nelemmaquina];
 P = new float**[nelemmaquina];
 r = new int**[nelemmaquina];
 c = new int**[nelemmaquina];

 for(int i=0; i<nelemmaquina; i++) {
  G[i] = new float*[NNODOELEM];
  U[i] = new float*[NNODOELEM];
  P[i] = new float*[NNODOELEM];
  r[i] = new int*[NNODOELEM];
  c[i] = new int*[NNODOELEM];
 }
 for(i=0; i<nelemmaquina; i++) {
  for(int j=0; j<NNODOELEM; j++) {
    G[i][j] = new float[NNODOELEM];
    U[i][j] = new float[NNODOELEM];
    P[i][j] = new float[NNODOELEM];
    r[i][j] = new int[NNODOELEM];
    c[i][j] = new int[NNODOELEM];
  }
```

```
    }

    for(int p=0; p<nelemmaquina; p++) {
     for(int q=0; q<NNODOELEM; q++) {
      for(int o=0; o<NNODOELEM; o++) {
           G[p][q][o] = 0;
       U[p][q][o] = 0;
       P[p][q][o] = 0;
       r[p][q][o] = 0;
       c[p][q][o] = 0;
      }
     }
    }
   }
/**************************************************************************/
/* Funcion InicializarPVM :                                              */
/**************************************************************************/

void Subdominio::InicializarPVM(void)
  {
   int mytid = pvm_mytid(),
       msgtype = 0;
   pvm_recv(-1, msgtype);
   pvm_upkint(&SLAVES, 1, 1);
   tids = new int[SLAVES];
   pvm_upkint(tids, SLAVES, 1);
   for(int i=0; i<SLAVES; i++)
     if(mytid==tids[i]) { me=i; break; }
  }
/**************************************************************************/
/* Funcion AsignarDominio : Con los valores de la malla recibidos, los re-  */
/*                    parte por elemento para que los objetos resuel-       */
/*                    van su propia matriz de conductancia.                */
/**************************************************************************/
void Subdominio::AsignarDominio(void)
{
 int fi, fj, fm, fn, l;
 Punto i, j, m, n;
 for(int p=0; p<nelemmaquina; p++) {
  l = p*NNODOELEM;
  fi = puntos[l];  fj = puntos[l+1];  fm = puntos[l+2];  fn = puntos[l+3];
  i = e[p].EvaluarPunto(ord[l],abs[l],fi);
  j = e[p].EvaluarPunto(ord[l+1],abs[l+1],fj);
  m = e[p].EvaluarPunto(ord[l+2],abs[l+2],fm);
  n = e[p].EvaluarPunto(ord[l+3],abs[l+3],fn);
  e[p].Asignar(i, j, m, n);
  e[p].Matriz(G[p], U[p], P[p], r[p], c[p]);
 }
}

/**************************************************************************/
/* Funcion Enviar : Ya que se obtuvieron los arreglos de matrices (donde    */
/*            cada matriz del arreglo es la matriz de conductancia         */
/*            de cada elemento), se obtuvo tambien dos arreglos de         */
/*            matrices en los cuales se obtienen la posicion (ren-         */
/*            glon y columna) donde deben de ir en la matriz global        */
/*            los resultados de los arreglos G, U y P;                     */
/**************************************************************************/
void Subdominio::Enviar(void)
 {
 int l=0,
```

```
    msgtype = 5,
    master = pvm_parent(),
    dim = nelemmaquina*NNODOELEM*NNODOELEM;
  float* g;
  float* u;
  float* p;
  int* R;
  int* C;

  g = new float[dim];
  u = new float[dim];
  p = new float[dim];
  R = new int[dim];
  C = new int[dim];

  for(int i=0; i<nelemmaquina; i++) {
    for(int j=0; j<NNODOELEM; j++) {
      for(int k=0; k<NNODOELEM; k++) {
        g[l] = G[i][j][k];
        u[l] = U[i][j][k];
        p[l] = P[i][j][k];
        R[l] = r[i][j][k];
        C[l] = c[i][j][k];
        l++;
      }
    }
  }

  pvm_initsend(PvmDataDefault);
  pvm_pkint(&nelemmaquina, 1, 1);
  pvm_pkfloat(g, dim, 1);
  pvm_pkfloat(u, dim, 1);
  pvm_pkfloat(p, dim, 1);
  pvm_pkint(R, dim, 1);
  pvm_pkint(C, dim, 1);
  pvm_send(master, msgtype);

  FreeMem();
}

void Subdominio::FreeMem(void)
 {
  delete[nelemmaquina] e;
  delete[nelemmaquina*NNODOELEM] puntos;
  delete[nelemmaquina*NNODOELEM] ord;
  delete[nelemmaquina*NNODOELEM] abs;

  for(int i=0; i<nelemmaquina; i++) {
    for(int j=0; j<NNODOELEM; j++) {
      delete[NNODOELEM] G[i][j];
      delete[NNODOELEM] U[i][j];
      delete[NNODOELEM] P[i][j];
      delete[NNODOELEM] r[i][j];
      delete[NNODOELEM] c[i][j];
    }
  }
  for(i=0; i<nelemmaquina; i++) {
    delete[NNODOELEM] *G[i];
    delete[NNODOELEM] *U[i];
    delete[NNODOELEM] *P[i];
    delete[NNODOELEM] *r[i];
```

```
    delete[NNODOELEM] *c[i];
  }
  delete[nelemmaquina] **r;
  delete[nelemmaquina] **c;
  delete[nelemmaquina] **G;
  delete[nelemmaquina] **U;
  delete[nelemmaquina] **P;
}

void Subdominio::Jacobi(void)
{
  int NNODOS;
  int SLAVES;
  int msgtype;                float* B;
  int dt = 5;                 float* c;
  int renglones;              float* f;
  int master;                 //float* t;
  int x = 0;                  float* o;
  int a;                      float* CNEW;
  int ANCHO;                  float** g;
  int nstep=0;                //float** u;
  int* r;                     float** p;
  int* kode;                  float amax;
                              float oldval;
                              float sum;
                              float err;


  master = pvm_parent();

  msgtype = 69;

  pvm_recv(master, msgtype);
  pvm_upkint(&SLAVES, 1, 1);
  pvm_upkint(&NNODOS, 1, 1);

  r = new int[SLAVES];
  CNEW = new float[NNODOS];

  pvm_upkint(&ANCHO, 1, 1);
  pvm_upkint(r, SLAVES, 1);
  pvm_upkfloat(CNEW, NNODOS, 1);

  for(int i=0; i<me; i++) x += r[i];

  msgtype = 7;

  pvm_recv(-1, msgtype);
  pvm_upkint(&renglones, 1, 1);

  kode = new int[renglones];
  B = new float[renglones];
  c = new float[renglones];
  g = new float*[renglones];
//  u = new float*[renglones];
  p = new float*[renglones];
  f = new float[renglones*NNODOS];
//  t = new float[renglones*NNODOS];
  o = new float[renglones*NNODOS];

  for(i=0; i< renglones; i++) {
```

```
    g[i] = new float[NNODOS];
//    u[i] = new float[NNODOS];
    p[i] = new float[NNODOS];
  }
  pvm_upkfloat(f, renglones*NNODOS, 1);
//  pvm_upkfloat(t, renglones*NNODOS, 1);
  pvm_upkfloat(o, renglones*NNODOS, 1);
  pvm_upkint(kode, renglones, 1);

  a = 0;

  for(i=0; i<renglones; i++) {
    for(int h=0; h<NNODOS; h++) {
      g[i][h] = f[a];
//      u[i][h] = t[a];
      p[i][h] = o[a];
      a++;
    }
  }

  delete[renglones*NNODOS] f;
//  delete[renglones*NNODOS] t;
  delete[renglones*NNODOS] o;

  do
  {
    for(i=0; i<renglones; i++)
    {
      B[i] = 0;
      for(int j=0; j<NNODOS; j++)
      {
        if(j==(x+i))
        {
          B[i]+= p[i][j]*CNEW[j]/dt;
          if((j%ANCHO)!=0) B[i]+= p[i][j-1]*CNEW[j-1]/dt;
          if(((j+1)%ANCHO)!=0) B[i] += p[i][j+1]*CNEW[j+1]/dt;
          a = j-ANCHO;
          if(a>=0)
          {
            B[i]+= p[i][a]*CNEW[a]/dt;
            if((a%ANCHO)!=0) B[i] += p[i][a-1]*CNEW[a-1]/dt;
            if(((a+1)%ANCHO)!=0) B[i] += p[i][a+1]*CNEW[a+1]/dt;
          }
          a = j+ ANCHO;
          if(a<NNODOS)
          {
            B[i]+= p[i][a]*CNEW[a]/dt;
            if((a%ANCHO)!=0) B[i] += p[i][a-1]*CNEW[a-1]/dt;
            if(((a+1)%ANCHO)!=0) B[i] += p[i][a+1]*CNEW[a+1]/dt;
          }
        }
      }
    }
    do
    {
      amax = 0;
      for(i=0; i<renglones; i++)
      {
            if(kode[i]!=1)
        {
              oldval = CNEW[x+i];
```

```
              sum = 0;
              for(int j=0; j<NNODOS; j++)
         {
                if(j==(x+i))
           {
           if((j%ANCHO)!=0) sum += (g[i][j-1]) * CNEW[j-1]; //+ u[i][j-1] + p[i][j-1]/dt) * CNEW[j-1];
           if(((j+1)%ANCHO)!=0) sum += (g[i][j+1]) * CNEW[j+1]; // + u[i][j+1] + p[i][j+1]/dt) * CNEW[j+1];
           a = j-ANCHO;
           if(a>=0)
            {
             sum += (g[i][a]) * CNEW[a]; // + u[i][a] + p[i][a]/dt) * CNEW[a];
             if((a%ANCHO)!=0) sum += (g[i][a-1]) * CNEW[a-1];// + u[i][a-1] + p[i][a-1]/dt) * CNEW[a-1];
             if(((a+1)%ANCHO)!=0) sum += (g[i][a+1]) * CNEW[a+1]; // + u[i][a+1] + p[i][a+1]/dt) * CNEW[a+1];
            }
           a = j+ANCHO;
           if(a<NNODOS)
            {
             sum += (g[i][a]) * CNEW[a]; // + u[i][a] + p[i][a]/dt) * CNEW[a];
             if((a%ANCHO)!=0) sum += (g[i][a-1]) * CNEW[a-1]; // + u[i][a-1] + p[i][a-1]/dt) * CNEW[a-1];
             if(((a+1)%ANCHO)!=0) sum += (g[i][a+1]) * CNEW[a+1]; // + u[i][a+1] + p[i][a+1]/dt) * CNEW[a+1];
            }
           }
              }
              c[i] = (-sum+B[i])/(g[i][x+i]); // + u[i][x+i] + p[i][x+i]/dt);
              err = oldval - c[i];
              if(err<0) err = -err;
              if(err>amax) amax = err;
             }
      else
       {
        c[i] = CNEW[x+i];
       }
     }

     msgtype = 77;

     pvm_initsend(PvmDataDefault);
     pvm_pkint(&me, 1, 1);
     pvm_pkfloat(&amax, 1, 1);
     pvm_pkfloat(c, renglones, 1);

     pvm_send(master, msgtype);

     msgtype = 777;

     pvm_recv(master, msgtype);
     pvm_upkfloat(CNEW, NNODOS, 1);
     pvm_upkfloat(&amax, 1, 1);

    }while(amax>TOL);

   msgtype = 666;

   pvm_recv(master, msgtype);
   pvm_upkint(&nstep, 1, 1);

  }while((nstep+1)<100);

}
```

```
void Subdominio::GradienteConjugado(void)
{
 FILE *file;
 file = fopen("exe/i.dat", "w");

 int k, who, iteration=0;
 int master, msgtype, SLAVES, n, a, e=0, nstep=0, dt =5;
 double delta0, deltanew, deltaold, alfa, beta;

 int* ren;                    double* x;double* tempA;
 double* b;                        double** A;    double* Ax;
 double* r;                   double* d;      double* q;
 double* subx;                double* subd;

 master = pvm_parent();

 msgtype = 66;

 pvm_recv(master, msgtype);
 pvm_upkint(&SLAVES, 1, 1);
 pvm_upkint(&n, 1, 1);

 ren    = new int[SLAVES];
 x      = new double[n];

 pvm_upkint(ren, SLAVES, 1);

 b      = new double[ren[me]];
 tempA  = new double[ren[me]*n];
 A      = new double*[ren[me]];
 Ax     = new double[ren[me]];
 r      = new double[ren[me]];
 q      = new double[ren[me]];
 subx   = new double[ren[me]];
 subd   = new double[ren[me]];
 d      = new double[n];

 for(int i=0; i<ren[me]; i++) A[i] = new double[n];

 msgtype =69;

 pvm_recv(-1, msgtype);
 pvm_upkdouble(tempA, ren[me]*n, 1);

 a=0;

 for(i=0; i<ren[me]; i++)
  {
  for(int h=0; h<n; h++) {
   A[i][h] = tempA[a++];
  }
  }

 delete[ren[me]*n] tempA;

// Aqui todos los esclavos cuentan con A[ren[me]][n]
// Comenzamos integracion en el tiempo.

//  do {

 msgtype = 7;
```

```
  pvm_recv(-1, msgtype);
  pvm_upkdouble(x, n, 1);
  pvm_upkdouble(b, ren[me], 1);

// En este punto ya tienen A[ren[me]][n], x[n], b[ren[me]].
// Comienza Gradiente conjugado

  MatrizVector(A,ren[me], n, x, Ax);

  for(i=0; i<ren[me]; i++) {
    r[i] = b[i] - Ax[i];
    subd[i] = r[i];
  }
  for(i=0; i<ren[me]; i++) d[i]=subd[i];
  msgtype = 77;
  if(SLAVES > 1) {
   if(me==0) {
     for(i=1; i<SLAVES; i++)
       {
       pvm_recv(-1, msgtype);
       pvm_upkint(&who, 1, 1);
       tempA = new double[ren[who]];
       pvm_upkdouble(tempA, ren[who], 1);
       int beginat = 0;
       for(int j=0; j<who; j++) beginat += ren[j];
       for(j=0; j<ren[who]; j++) d[beginat+j] = tempA[j];
       delete[ren[who]] tempA;
       }

     msgtype = 88;

     for(int i=1; i<SLAVES; i++) {
       pvm_initsend(PvmDataDefault);
       pvm_pkdouble(d, n, 1);
       pvm_send(tids[i], msgtype);
     }
   }

   else {
     pvm_initsend(PvmDataDefault);
     pvm_pkint(&me, 1, 1);
     pvm_pkdouble(r, ren[me], 1);
     pvm_send(tids[0], msgtype);
     msgtype = 88;
     pvm_recv(-1, msgtype);
     pvm_upkdouble(d, n, 1);
   }
  }

  deltanew = ProductoInterno(r, r, ren[me]);

  delta0 = deltanew;

  do {

    MatrizVector(A, ren[me], n, d, q);

    alfa = deltanew/ProductoInterno(subd, q, ren[me]);
```

```
    for(i=0; i<ren[me]; i++) subx[i] += alfa*subd[i];

fprintf(file, "start-> %d\n", iteration);

    if((iteration%5)==0 && iteration!=0)
      {
fprintf(file, "if\n");
      for(i=0; i<ren[me]; i++) x[i]=subx[i];
      if(SLAVES > 1) {
        if(me==0)
         {
         for(i=1; i<SLAVES; i++)
           {
            pvm_recv(-1, msgtype);
            pvm_upkint(&who, 1, 1);
            tempA = new double[ren[who]];
            pvm_upkdouble(tempA, ren[who], 1);
            int beginat = 0;
            for(int j=0; j<who; j++) beginat += ren[j];
            for(j=0; j<ren[who]; j++) x[beginat+j] = tempA[j];
            delete[ren[who]] tempA;
           }

              msgtype = 88;

         for(int i=1; i<SLAVES; i++)
           {
            pvm_initsend(PvmDataDefault);
            pvm_pkdouble(x, n, 1);
            pvm_send(tids[i], msgtype);
           }
         }
        else {
         pvm_initsend(PvmDataDefault);
         pvm_pkint(&me, 1, 1);
         pvm_pkdouble(subx, ren[me], 1);
         pvm_send(tids[0], msgtype);

         msgtype = 88;

         pvm_recv(-1, msgtype);
         pvm_upkdouble(x, n, 1);
         }
       }

      MatrizVector(A, ren[me], n, x, Ax);
      for(i=0; i<ren[me]; i++) r[i] = b[i] - Ax[i];
    }
    else {
fprintf(file, "else\n");
      for(i=0; i<ren[me]; i++) r[i] -= alfa*q[i];
    }

fprintf(file, "end-> %d\n", iteration);

    deltaold = deltanew;

    deltanew = ProductoInterno(r, r, ren[me]);

    beta = deltanew/deltaold;
```

```
 for(i=0; i<ren[me]; i++)
   subd[i] = r[i] + beta*subd[i];

 msgtype = 777;

 for(i=0; i<ren[me]; i++) {
  x[i]=subx[i];
  d[i]=subd[i];
 }

if(SLAVES > 1) {
  if(me==0)
   {
   double* tempB;
   for(i=1; i<SLAVES; i++)
     {
     pvm_recv(-1, msgtype);
     pvm_upkint(&who, 1, 1);
     tempA = new double[ren[who]];
     tempB = new double[ren[who]];
     pvm_upkdouble(tempA, ren[who], 1);
     pvm_upkdouble(tempB, ren[who], 1);
     int beginat = 0;
     for(int j=0; j<who; j++) beginat += ren[j];
     for(j=0; j<ren[who]; j++) {
       x[beginat+j] = tempA[j];
       d[beginat+j] = tempB[j];
     }
     delete[ren[who]] tempA;
     delete[ren[who]] tempB;
     }

   msgtype = 888;

   for(int i=1; i<SLAVES; i++)
     {
     pvm_initsend(PvmDataDefault);
     pvm_pkdouble(x, n, 1);
     pvm_pkdouble(d, n, 1);
     pvm_send(tids[i], msgtype);
     }
   }
  else {
     pvm_initsend(PvmDataDefault);
     pvm_pkint(&me, 1, 1);
     pvm_pkdouble(subx, ren[me], 1);
     pvm_pkdouble(subd, ren[me], 1);
     pvm_send(tids[0], msgtype);

     msgtype = 888;

     pvm_recv(-1, msgtype);
     pvm_upkdouble(x, n, 1);
     pvm_upkdouble(d, n, 1);
   }
 }

 iteration++;

}while(deltanew > (TOL*delta0));
```

```
fclose(file);

   if(me == 0) {
     msgtype = 8888;
     pvm_initsend(PvmDataDefault);
     pvm_pkdouble(x, n, 1);
     pvm_send(master, msgtype);
   }

  msgtype = 7777;

  pvm_recv(-1, msgtype);
  pvm_upkint(&k, 1, 1);
//  }while(k<nstep);
}

void Subdominio::MatrizVector(double** M, int rM, int cM, double* x, double* Mx)
  {
  for(int i=0; i<rM; i++)
    {
    Mx[i] = 0;
    for(int j=0; j<cM; j++) Mx[i] += M[i][j]*x[j];
    }
  }

double Subdominio::ProductoInterno(double *a, double* b, int sa)
  {
  int msgtype;
  double result = 0;
  double resultp = 0;

  for(int i=0; i<sa; i++) resultp += a[i]*b[i];

  msgtype = 11;

  result = resultp;

  if(SLAVES > 1) {
  if(me==0)
   {

    for(i=1; i<SLAVES; i++)
      {
      pvm_recv(-1, msgtype);
      pvm_upkdouble(&resultp, 1, 1);
      result += resultp;
      }

   msgtype = 22;

   for(int i=1; i<SLAVES; i++)
     {
     pvm_initsend(PvmDataDefault);
     pvm_pkdouble(&result, 1, 1);
     pvm_send(tids[i], msgtype);
     }
   }
  else {
   pvm_initsend(PvmDataDefault);
   pvm_pkdouble(&resultp, 1, 1);
```

```
    pvm_send(-1, msgtype);

    msgtype = 22;

    pvm_recv(-1, msgtype);
    pvm_upkdouble(&result, 1, 1);
   }
  }

  return result;
 }
```

3. Código esclav2.cxx

```
/*****************************************************************************/
/* ESCLAV2.CXX : Programa esclavo que genera las matrices de conductancia    */
/*              de los elementos, para la ecuacion de transporte de so-      */
/*              luto en un acuifero homogeneo con elementos rectagulares    .*/
/*****************************************************************************/

#include "pvm3.h"
#include "objdom2.hxx"

void main()
{
 Subdominio Acuifero;

 Acuifero.AsignarDominio();

 Acuifero.Enviar();

 Acuifero.Jacobi();

// Acuifero.GradienteConjugado();

}
```

4. Código master2.hxx

```
/*****************************************************************************/
/* MASTER2.HXX : Archivo de encabezado para MASTER2.CXX, el cual resuelve */
/*              la ecuacion de adveccion-dispersion en un acuifero          */
/*****************************************************************************/
#define nstep 100
#define TOL 0.01

#include <string.h>
#include <stdio.h>              // Rutinas de salida a pantalla
#include "pvm3.h"               // Rutinas de pvm

#define SLAVENAME "esclav2"        // Nombre del programa esclavo
#define NNODOELEM 4               // Numero de nodos por elemento
                                  // (rectangular)
class dominio
 {
  int SLAVES,                 // Numero total de procesos esclavos
      NNODOS,                 // Numero total de nodos en la malla
      NELEM;                  // Numero total de elementos
  int* tids;                  // Etiquetas de cada proceso esclavo
```

```
  int* KODE;                    // Condicion de flujo en el punto
  int* NODE[NNODOELEM];              // Puntos designados a cada elemento
  float* CNEW;                   // Concentracion en cada punto
  float* X;                    // Ordenada del punto n en la malla
  float* Y;                    // abscisa del punto n en la malla
  float** G;                   // Matrices de conductancia G
  float** U;                                   // U y
  float** P;                                   // P.
  char f[12];                  // Nombre del archivo en el cual estan
public:                       // los datos de la malla
  dominio(void);                // Constructor(inicializa PVM)
  void CrearMalla(void);            // Crea malla y el archivo de datos
  void CargarMalla(void);           // Carga el archivo de datos
  void Inicializar(void);           // Reparte los elementos a cada esclavo
  void RecibirdeNodos(void);          // Recibe las matrices de conductancia
  void MostrarMatriz(void);           // Muestra la matriz de conductancia.
  void repartir(int* r);            // Reparte el numero de renglones
  void Jacobi(void);               // Jacobi en paralelo
  void GradienteConjugado(void);   // Gradiente Conjugado en paralelo
  ~dominio(void) {pvm_exit();}     // destructor
};

/***************************************************************************/
/* CONSTRUCTOR dominio, carga los datos de la malla del archivo f, e ini-  */
/*          cializa la maquina virtual.                                    */
/***************************************************************************/
dominio::dominio(void)
 {
 int opcion,                    // Se crea o carga nueva malla ?
     narch,                                    // # de diferentes arquitecturas
     mytid,                    // mi etiqueta (del maestro)
     msgtype;                   // etiqueta del mensaje
 struct pvmhostinfo *hostp;         // caracteristicas de las maquinas en
                                               // la maquina virtual.
//-----------------------------------------------------------------------
// Este es el menu principal, se crea un nuevo archivo en el cual se
// almacena la malla en la cual se va a trabajar o se carga una ya hecha
// previamente
//-----------------------------------------------------------------------
  printf("\nEscoge una de las siguientes opciones:");
  printf("\n  (1) Crear una nueva malla,");
  printf("\n  (2) Cargar una malla previamente hecha.");
  printf("\nopcion: ");
  scanf("%d", &opcion);
  if(opcion==1) { CrearMalla();}
  if(opcion==2) {
    printf("\nDame el nombre del archivo: ");
    scanf("%s", &f);
  }

//opcion = 2;
//strcpy(f,"malla2.dat");
  CargarMalla();
//-----------------------------------------------------------------------
  G=new float*[NNODOS];             // G, U, y P se dimensionan segun el
  U=new float*[NNODOS];             // el numero de nodos que contenga
  P=new float*[NNODOS];             // la malla en la cual se trabaja

  for(int i=0; i<NNODOS; i++) {
    G[i] = new float[NNODOS];
    U[i] = new float[NNODOS];
```

```
    P[i] = new float[NNODOS];
  }
  for(i=0; i<NNODOS; i++) {
   for(int j=0; j<NNODOS; j++) {
          G[i][j] = 0;              // Y aqui se incializan a cero las
      U[i][j] = 0;              // las matrices
      P[i][j] = 0;
   }
  }
//-----------------------------------------------------------------------
// Aqui se inicializa la maquina virtual, se abren los procesos esclavos y
// se le asigna su etiqueta (o tid)
//-----------------------------------------------------------------------
  mytid = pvm_mytid();
  pvm_config( &SLAVES, &narch, &hostp);
  tids = new int[SLAVES];
  pvm_spawn(SLAVENAME, (char **)0, 0, "", SLAVES, tids);
  msgtype = 0;
  pvm_initsend(PvmDataDefault);
  pvm_pkint(&SLAVES, 1, 1);
  pvm_pkint(tids, SLAVES, 1);
  pvm_mcast(tids, SLAVES, msgtype);
//-----------------------------------------------------------------------
// Aqui se muestra los datos obtenidos del archivo f y de los procesos
// anteriores.(opcional)
//-----------------------------------------------------------------------
/* printf("\nNumero de Nodos: %d\n", NNODOS);
  printf("\nNumero de Elementos: %d\n", NELEM);
  printf("\nNumero de esclavos: %d\n", SLAVES);
  printf("\nPunto\n");
  printf("i\tX(i)\tY(i)\tC(i)\tK(i)\n");
  for(i=0; i< NNODOS; i++) {
   printf("\n%d\t%2.1f\t%2.1f\t%2.1f\t%d", i, X[i], Y[i], CNEW[i], KODE[i]);
  }
  printf("\n\nEle\ti\tj\tm\tn\n");
  for(i=0; i<NELEM; i++) {
   printf("\n%d\t%d\t%d\t%d\t%d",i,NODE[0][i],NODE[1][i],NODE[2][i],NODE[3][i]);
  }
  printf("\n");*/
}

/****************************************************************************/
/* Funcion CrearMalla: En caso que no exista un archivo de la malla que se */
/*                  propone a solucionar, crea el archivo pidiendole los*/
/*                  valores iniciales y de frontera                      */
/****************************************************************************/
void dominio::CrearMalla(void)
 {
   FILE *file;                  // Apuntador a un archivo
   int inc,                                    // incremento nodos
     incelem,                          // incremento elementos
     cnode,                    // nodo actual
     juan,
     oldnod;                   // nodo anterior.
   float ngenp1,
       xinc,                   // incremento en x;
       yinc;                   // incremento en y;

   NNODOS = 0;                                 // Numero de nodos creados.
   oldnod = 2200;
```

```
    X = new float[oldnod];          // Dimensiona al tamano del nodo
    Y = new float[oldnod];          // anterior, en este caso un max de 200

    CNEW = new float[oldnod];
    KODE = new int[oldnod];
//---------------------------------------------------------------------
// Pide los datos para crear una malla, en caso que se salte unos puntos
// el programa crea los puntos en la malla que se salto.Para esto el pro-
// grama le pide el incremento en el numero nodal yendo de un punto a
// otro.
//---------------------------------------------------------------------
    printf("Proporcione los datos a la malla\n");
    printf("Incremento en el numero del nodo: "); scanf("%d", &inc);
    printf("\n");
    do {
      printf("Numero del Nodo: "); scanf("%d", &cnode);
      printf("Coordenadas Nodales:\n");
      printf("x = "); scanf("%f", &X[cnode]);
      printf("y = "); scanf("%f", &Y[cnode]);
      if(cnode > NNODOS) NNODOS = cnode;
      ngenp1 = (cnode - oldnod) / inc;
      if(ngenp1 > 0) {
              xinc = (X[cnode] - X[oldnod])/ ngenp1;
              yinc = (Y[cnode] - Y[oldnod])/ ngenp1;
              for(int j=(oldnod + inc); j<cnode; j+=inc) {
                X[j] = X[j-inc] + xinc;
                Y[j] = Y[j-inc] + yinc;
              }
      }
      oldnod = cnode;
    }while(cnode!=-1);
    NNODOS++;
    printf("\n");
//---------------------------------------------------------------------
// Ya creada toda la malla, ya se por el usuario o por la computadora,
// se procede a introducir los datos iniciales en la red y las condiciones
// a la frontera. Especificando si hay o no hay flujo en un cierto punto.
// Aqui se piden los valores de concentracion y flujo en cada punto de la
// malla.
//---------------------------------------------------------------------
    for(int i=0; i<NNODOS; i++) {
      CNEW[i] = 0;
      KODE[i] = 0;
    }
    for(i=0; i<inc; i++) {
      CNEW[i] = 10;
      KODE[i] = 1;
    }
    for(i=(NNODOS-inc); i<NNODOS; i++) {
      CNEW[i] = 0;
      KODE[i] = 1;
    }
    printf("\nDame el nombre del archivo:  "); scanf("%s", &f);
    file = fopen(f, "w");
    fprintf(file, "%d\n", NNODOS);
    for(i=0; i<NNODOS; i++) {
      fprintf(file, "%f\t%f\t%f\t%d\n", X[i], Y[i], CNEW[i], KODE[i]);
    }
//---------------------------------------------------------------------
// La maquina necesita saber que puntos pertenecen a los diferentes ele-
// mentos, por lo tanto se da el numero del punto que ocupa en la malla.
```

```
// Empezando por el punto ubicado en la esquina inferior izquierda y pro-
// siguiendo en sentido contrario a las manecillas del reloj, llevando los
// siguientes nombres: punto i (esquina inferior izquierda), punto j (es-
// quina inferior derecha), punto m (esquina superior derecha) y punto n
// esquina superior izquierda).
//-----------------------------------------------------------------------
   printf("\nDame el numero de elementos: "); scanf("%d", &NELEM);
   for(i=0; i<NNODOELEM; i++) NODE[i]=new int[NELEM];
   incelem = inc -1;
   for(i=0; i<NELEM; i++) {
     juan = i/incelem;
     NODE[0][i] = (i+juan)+1;
     NODE[1][i] = NODE[0][i]+inc;
     NODE[2][i] = NODE[1][i]-1;
     NODE[3][i] = NODE[2][i]-inc;
   }
   fprintf(file, "%d\n", NELEM);
   for(i=0; i<NELEM; i++) {
    fprintf(file,"%d\t%d\t%d\t%d\n",NODE[0][i],NODE[1][i],NODE[2][i],NODE[3][i]);
   }
   fclose(file);

   delete[oldnod] X;
   delete[oldnod] Y;
   delete[oldnod] CNEW;
   delete[oldnod] KODE;
   for(i=0; i<NNODOELEM; i++) delete[NELEM] NODE[i];
}

/***********************************************************************/
/* Funcion CargarMalla: Utilizando un archivo previamente creado carga los */
/*                  valores iniciales y de frontera.                 */
/***********************************************************************/
void dominio::CargarMalla(void)
 {
   FILE *file;

   file = fopen(f, "r");
   fscanf(file, "%d\n", &NNODOS);

   X = new float[NNODOS];
   Y = new float[NNODOS];
   CNEW = new float[NNODOS];
   KODE = new int[NNODOS];

   for(int i=0; i< NNODOS; i++) {
     fscanf(file,"%f\t%f\t%f\t\%d\n", &X[i], &Y[i], &CNEW[i], &KODE[i]);
   }

   fscanf(file, "%d\n", &NELEM);

   for(i=0; i<NNODOELEM; i++) NODE[i]=new int[NELEM];

   for(i=0; i<NELEM; i++) {
    fscanf(file,"%d\t%d\t%d\t%d\n",&NODE[0][i],&NODE[1][i],&NODE[2][i],&NODE[3][i]);
   }

   fclose(file);
 }
/***********************************************************************/
/* Funcion Inicializar: Reparte los elementos entre el numero total de es- */
```

```
/*              clavos para que estos resuelvan la matriz de con-  */
/*              ductancia.                                          */
/***********************************************************************/
void dominio::Inicializar(void)
 {
  int s, t, u, v=0, dim, msgtype, residuo = NELEM%SLAVES;
  int* NELEMMAQUINA;            // Numero de elementos por esclavo
  int* puntos;
  float* ord;
  float* abs;

  NELEMMAQUINA=new int[SLAVES];

  for(int i=0; i<SLAVES; i++) NELEMMAQUINA[i] = NELEM/SLAVES;
  for(i=0; i<residuo; i++) NELEMMAQUINA[i]++;

  for(i=0; i<SLAVES; i++)
   {
    s = 0;
    t = NELEMMAQUINA[i];

    puntos = new int[t*NNODOELEM];
    ord = new float[t*NNODOELEM];
    abs = new float[t*NNODOELEM];

    for(int j=0; j<t; j++){
     for(int k=0; k<NNODOELEM; k++) {
      u = NODE[k][v];
      puntos[s] = u;
      ord[s] = X[u];
      abs[s] = Y[u];
      s++;
     }
     v++;
    }

    msgtype = 1;
    dim = t*NNODOELEM;
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&t, 1, 1);
    pvm_pkint(puntos, dim, 1);
    pvm_pkfloat(ord, dim, 1);
    pvm_pkfloat(abs, dim, 1);
    pvm_send(tids[i], msgtype);

    delete[t*NNODOELEM] puntos;
    delete[t*NNODOELEM] ord;
    delete[t*NNODOELEM] abs;
   }
 }

/***********************************************************************/
/* Funcion RecibirdeNodos : Recibe los resultados de los diferentes escla- */
/*                          vos y crea la matriz de conductancia global   */
/***********************************************************************/
void dominio::RecibirdeNodos(void) {
   int  msgtype, dim, R, C, nelemmaquina;
   int maxdim=0;
   int* r;
   int* c;
   float* g;
```

```
  float* u;
  float* p;

  msgtype = 5;

  for(int l=0; l<SLAVES; l++) {
   pvm_recv(-1, msgtype);
   pvm_upkint(&nelemmaquina, 1, 1);

   dim = nelemmaquina*NNODOELEM*NNODOELEM;

   r = new int[dim];
   c = new int[dim];
   g = new float[dim];
   u = new float[dim];
   p = new float[dim];

   pvm_upkfloat(g, dim, 1);
   pvm_upkfloat(u, dim, 1);
   pvm_upkfloat(p, dim, 1);
   pvm_upkint(r, dim, 1);
   pvm_upkint(c, dim, 1);

   for(int i=0; i<dim; i++) {
     R = r[i];
          C = c[i];
          G[R][C] += g[i];
     U[R][C] += u[i];
          P[R][C] += p[i];
   }
  }

  delete[dim] r;
  delete[dim] c;
  delete[dim] g;
  delete[dim] u;
  delete[dim] p;
 }

void dominio::MostrarMatriz(void)
 {
  printf("\n");
  for(int i=0; i<NNODOS; i++) {
   for(int j= 0; j<NNODOS; j++) {
     printf("%1.1f ", G[i][j]);
   }
   printf("\n");
  }
  printf("\n");
  printf("\n");
  for(i=0; i<NNODOS; i++) {
   for(int j= 0; j<NNODOS; j++) {
     printf("%1.1f ", U[i][j]);
   }
   printf("\n");
  }
  printf("\n");
  printf("\n");
  for(i=0; i<NNODOS; i++) {
   for(int j= 0; j<NNODOS; j++) {
     printf("%1.1f ", P[i][j]);
```

```
    }
    printf("\n");
   }
  }

/***********************************************************************/
/* Funcion repartir : Reparte el numero de renglones entre el numero de   */
/*                         esclavos                                    */
/***********************************************************************/
void dominio::repartir(int* r)
 {
  int residuo = NNODOS%SLAVES;
  for(int i=0; i<SLAVES; i++) r[i] =0;
  for(i=0; i<SLAVES;  i++) r[i] = NNODOS/SLAVES;
  for(i=0; i<residuo; i++) r[i]++;
 }

void dominio::Jacobi(void)
{
 int dt = 5;                            float* g;
 int ANCHO;                       //float* u;
 int msgtype;                           float* p;
 int kount = 1;                   float* c;
 int kprint = 2;                  float amax=0;
 int* kode;                             float amaxnodo = 0;
 int* r;                          float tiempo = dt;
 int renglones;                   float tab;
 int x = 0;
 int quien;
 int s, stop, set=0;

 r = new int[SLAVES];

 repartir(r);

 msgtype = 69;

 ANCHO = NODE[2][0] - NODE[3][0];

 do
 {
  set+=ANCHO;
  tab=X[set]-X[0];
  stop = (int)(tab/10);
 }while(stop!=1);

 pvm_initsend(PvmDataDefault);
 pvm_pkint(&SLAVES, 1, 1);
 pvm_pkint(&NNODOS, 1, 1);
 pvm_pkint(&ANCHO, 1, 1);
 pvm_pkint(r, SLAVES, 1);
 pvm_pkfloat(CNEW, NNODOS, 1);
 pvm_mcast(tids, SLAVES, msgtype);

 msgtype = 7;

 for(int i=0; i<SLAVES; i++)
  {
   s=0;
   renglones = r[i];
   kode = new int[renglones];
```

```
    g = new float[renglones*NNODOS];
//   u = new float[renglones*NNODOS];
    p = new float[renglones*NNODOS];

    for(int j=0; j< renglones; j++)
     {
      kode[j] = KODE[x];
      for(int k=0; k<NNODOS; k++)
       {
        g[s] = G[x][k] + U[x][k] + P[x][k]/dt;;
//        u[s] = U[x][k];
        p[s] = P[x][k];
        s++;
       }
      x++;
     }
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&renglones, 1, 1);
    pvm_pkfloat(g, renglones*NNODOS, 1);
//   pvm_pkfloat(u, renglones*NNODOS, 1);
    pvm_pkfloat(p, renglones*NNODOS, 1);
    pvm_pkint(kode, renglones, 1);
    pvm_send(tids[i], msgtype);

    delete[renglones] kode;
    delete[renglones*NNODOS] g;
//   delete[renglones*NNODOS] u;
    delete[renglones*NNODOS] p;
  }

    delete[NNODOS] KODE;
    for(i=0; i<NNODOELEM; i++) delete[NELEM] NODE[i];
    delete[NNODOS] X;
    delete[NNODOS] Y;
    for(i=0; i<NNODOS; i++) {
      delete[NNODOS] G[i];
      delete[NNODOS] U[i];
      delete[NNODOS] P[i];
    }
    delete[NNODOS] *G;
    delete[NNODOS] *U;
    delete[NNODOS] *P;

  printf("\n\t\t\t\tConcentracion\t\t\t\tTime\n\n");

  printf("\nChilo Juan!\n");

  for(int k=0; k<nstep; k++)
   {
    do
     {
      amax = 0;
      msgtype = 77;
      x=0;
      for(i=0; i<SLAVES; i++)
       {
//         c = new float[r[i]];
//         pvm_recv(tids[i], msgtype);
             pvm_recv(-1, msgtype);
        pvm_upkint(&quien, 1, 1);
        pvm_upkfloat(&amaxnodo, 1, 1);
```

```
        c = new float[r[quien]];

        pvm_upkfloat(c, r[quien], 1);
        for(int j=0; j<quien; j++) x += r[j];
        renglones = r[quien];
        for(j=0; j<renglones; j++)
        {
          CNEW[x] = c[j];
          x++;
        }
        x=0;
        if(amaxnodo>amax) amax = amaxnodo;
        delete[r[quien]] c;
//printf("\namax: %f\n", amaxnodo);

      }

      msgtype = 777;

      pvm_initsend(PvmDataDefault);
      pvm_pkfloat(CNEW, NNODOS, 1);
      pvm_pkfloat(&amax, 1, 1);
      pvm_mcast(tids, SLAVES, msgtype);

    }while(amax>TOL);

//    if(kount==kprint)
//    {
      for(i=0; i<NNODOS; i+=set) printf("%2.2f ", CNEW[i]);
           printf("%3.2f\n", tiempo);
           kount = 0;
//    }
    tiempo = tiempo+dt;
    kount++;

    msgtype = 666;

    pvm_initsend(PvmDataDefault);
    pvm_pkint(&k, 1, 1);
    pvm_mcast(tids, SLAVES, msgtype);
  }

}

void dominio::GradienteConjugado(void)
 {

  int l=0;
  int dt = 5;
  int msgtype;
  int set=0, stop, tiempo=dt, w, n, kount=1, kprint=2;
  int* ren;
  float tab;
  float* COLD;
  double* x;
  double* b;
  double** A;

  ren  = new int[SLAVES];
  COLD = new float[NNODOS];
```

```
for(int i=0; i<NNODOS; i++) COLD[i] = CNEW[i];

w = NODE[2][0] - NODE[3][0];

n = NNODOS - 2*w;

do
{
 set+=w;
 tab=X[set]-X[0];
 stop = (int)(tab/10);
}while(stop!=1);



x = new double[n];
b = new double[n];
A = new double*[n];
for( i=0; i<n; i++) A[i] = new double[n];

for(i=0; i<n; i++) {
  for(int j=0; j<n; j++) {
    A[i][j] = G[w+i][w+j] + U[w+i][w+j] + P[w+i][w+j]/dt;
  }
}

for(i=0; i<SLAVES; i++) ren[i] = n/SLAVES;
for(i=0; i<(n%SLAVES); i++) ren[i]= ren[i]+1;

msgtype = 66;

pvm_initsend(PvmDataDefault);
pvm_pkint(&SLAVES, 1, 1);
pvm_pkint(&n, 1, 1);
pvm_pkint(ren, SLAVES, 1);
pvm_mcast(tids, SLAVES, msgtype);

msgtype = 69;

for(i=0; i<SLAVES; i++)
  {
  double* a;
  int renglones = ren[i];
  int s = 0;
  a = new double[renglones*n];

  for(int j=0; j<renglones; j++)
    {
    for(int k=0; k<n; k++) a[s++] = A[l][k];
    l++;
    }

  pvm_initsend(PvmDataDefault);
  pvm_pkdouble(a, renglones*n, 1);
  pvm_send(tids[i], msgtype);

  delete[renglones*n] a;
  }

// Aqui todos los esclavos cuentan con A[ren[me]][n]
```

```
// Comenzamos integracion en el tiempo.

  printf("\n\t\t\t\tConcentracion\t\t\t\tTime\n\n");

// for(int k=0; k<nstep; k++)
//   {
int k = 0;

    for(i=0; i<n; i++) {
      b[i] = 0;     x[i] = COLD[w+i];
      for(int j=0; j<NNODOS; j++) {
        b[i] += P[w+i][j] * COLD[j]/dt;
        //if(KODE[j]==1) b[i] -= (G[w+i][j] + U[w+i][j] + P[w+i][j]/dt)*COLD[j];
      }
    }

 for(i=0; i<n; i++) {
   for(int j=0; j<n; j++)
      printf("%2.2f ", A[i][j]);
   printf("\n");
 }

    msgtype = 7;

    l = 0;

    for(i=0; i<SLAVES; i++)
      {
      int s = 0;
      int renglones = ren[i];
      double* v;
      v = new double[renglones];

      for(int j=0; j<renglones; j++)
        v[s++] = b[l++];

      pvm_initsend(PvmDataDefault);
      pvm_pkdouble(x, n, 1);
      pvm_pkdouble(v, renglones, 1);
      pvm_send(tids[i], msgtype);

      delete[renglones] v;
      }

    delete[NNODOS] KODE;
    for(i=0; i<NNODOELEM; i++) delete[NELEM] NODE[i];
    delete[NNODOS] X;
    delete[NNODOS] Y;
    for(i=0; i<NNODOS; i++) {
      delete[NNODOS] G[i];
      delete[NNODOS] U[i];
      delete[NNODOS] P[i];
    }
    delete[NNODOS] *G;
    delete[NNODOS] *U;
    delete[NNODOS] *P;

// En este punto ya tienen A[ren[me]][n], x[n], b[ren[me].
// Comienza Gradiente conjugado

   msgtype = 8888;
```

```
   pvm_recv(tids[0], msgtype);
   pvm_upkdouble(x, n, 1);

   for(i=0; i<n; i++) CNEW[w+i] = x[i];

//    if(kount==kprint)
//    {
        for(i=0; i<NNODOS; i+=set) printf("%2.2f ", CNEW[i]);
            printf("%3.2f\n", tiempo);
            kount = 0;
//    }
   tiempo = tiempo+dt;
   kount++;

   msgtype = 7777;

   pvm_initsend(PvmDataDefault);
   pvm_pkint(&k, 1, 1);
   pvm_mcast(tids, SLAVES, msgtype);


//  }
}
```

5. *Código master2.cxx*

```
/***************************************************************************/
/* MASTER2.CXX : Programa maestro que resuelve la ecuacion de transporte  */
/*             de soluto en un acuifero homogeneo con elementos rectan-    */
/*             gulares.                                                    */
/***************************************************************************/

#include "master2.hxx"
#include <time.h>

void main(void)
 {
  float tiempo[3];
  struct timeval tv1, tv2;
  dominio Acuifero;              // Objeto Acuifero.

  gettimeofday(&tv1, (struct timezone*)0);

  Acuifero.Inicializar();        // Inicializa los esclavos mandandoles
                                 // los valores de la malla.
  gettimeofday(&tv2, (struct timezone*)0);

  tiempo[0] = (tv2.tv_sec-tv1.tv_sec)*1000000+tv2.tv_usec-tv1.tv_usec;

  gettimeofday(&tv1, (struct timezone*)0);

  Acuifero.RecibirdeNodos();       // Recibe las matrices de los elementos.

  gettimeofday(&tv2, (struct timezone*)0);

  tiempo[1] = (tv2.tv_sec-tv1.tv_sec)*1000000+tv2.tv_usec-tv1.tv_usec;

  gettimeofday(&tv1, (struct timezone*)0);

  Acuifero.Jacobi();
```

```
// Acuifero.GradienteConjugado();

gettimeofday(&tv2, (struct timezone*)0);

tiempo[2] = (tv2.tv_sec-tv1.tv_sec)*1000000+tv2.tv_usec-tv1.tv_usec;

printf("\n\nBloque Inicializar:\t%6.1f seg.\t%10.1f Mseg.\n", tiempo[0]/1000000, tiempo[0]);
printf("\n\nBloque Recibir de Nodos:\t%6.1f seg.\t%10.1f Mseg.\n", tiempo[1]/1000000, tiempo[1]);
printf("\n\nBloque 3:\t%6.1f seg.\t%10.1f Mseg.\n", tiempo[2]/1000000, tiempo[2]);
}
```