UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS

Computación Paralela con PVM en una Red Local Heterogénea

TESIS

Que para obtener el título de

Lic. en Ciencias Computacionales

presenta:

Roberto Soto Amador

Ensenada, Baja California. Octubre de 1996.

BIBLIOTECA CENTRAL ENSENADA U.A.B.C.

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS

COMPUTACIÓN PARALELA CON PVM EN UNA RED LOCAL HETEROGÉNEA

TESIS PROFESIONAL

QUE PRESENTA

ROBERTO SOTO AMADOR

APROBADO POR:

DR. José Gómez Valdés

Presidente del Jurado

LCC Alberto Leopoldo Morán y Solares

Secretario

LCC José Ignacio Ascencio López

1er. Vocal

DEDICATORIA

A mi compañera y esposa por creer en mi y rezar por nosotros cada día.

A mi Mamá y su gran corazón. Te quiero mucho.

A mis hermanos Omar, Manuel, Juan y Raúl, por no olvidar quienes son.

Y a mi Padre después de todo.

AGRADECIMIENTOS

A mi asesor de tesis Dr. José Gómez por darme la oportunidad de titularme, por su colaboración en mi tesis y sobre todo por su apoyo en mi carrera profesional.

A mis sinodales LCC Ignacio Ascencio y LCC Leopoldo Morán por su pacienciay su tiempo invertidos en la revisión y sugerencias sobre mi tesis.

A CICESE por darme el ambiente para realizar mis objetivos.

Al departamento de Oceanografía Física por brindarme los recursos necesarios para elaborar mi tesis.

A CONACyT por financiar mi tesis a través del proyecto 225008-5-3523T.

A la SCyR de CICESE por sus asesorías, en especial a Carlos L. Famozo y Martín Gaynor.

A la Universidad Autónoma de Baja California por albergar la Facultad de Ciencias que fue parte de mi formación.

A las dos generaciones de LCC donde tome clases y donde encontre buenos amigos, Joé Godoy, Ray, Rambo, Neto, Marcos ("Delicioso o Nutritivo?"), Nancy, Imelda, Alicia, Dalila ("las muchachas"), Iram.

A los profesores que me instruyeron y me motivaron durante mi carrera en la Universidad.

A mis Amigos y los amigos de mis Amigos de siempre.

A todos y cada uno de los involucrados en mi tesis que escapan a mi memoria.

Y a ti, que siempre me has cuidado...

RESUMEN de la tesis de Roberto Soto Amador presentada como requisito parcial para la obtención de la Licenciatura en Ciencias Computacionales. Ensenada, Baja California, México. 26 de Septiembre de 1996.

Computación Paralela con PVM en una Red Local Heterogénea

Resumen aprobado:

Director de Tesis: Dr. José Gómez Valdes

En este trabajo se muestra la viabilidad de la computación paralela en computación científica implantando un programa paralelo de la ecuación de flujo transitorio de un acuífero simple. El modelo se discretiza en elementos finitos. Se prueba la herramienta PVM en una red local heterogénea. Encontrando un buen desempeño en la subred de Oceanografía Física de CICESE. El método de Jacobi se paraleliza facilmente, en cambio el de Gauss-Seidel depende de la estructura de la matriz y del número de iteraciones. Se obtuvo un speedup de 2 con cuatro procesadores para el método de Jacobi, debido a que la subred no esta diseñada para hacer cómputo en paralelo.

1 INTRODUCCIÓN

1.1 Presentación del problema

En este trabajo se desarrolla un programa paralelo en C a partir del trabajo en elementos finitos de la ecuación de Poisson en dos dimensiones establecida en el campo de la geohidrología por Wang y Anderson (1982), en una red local heterogénea en ambiente PVM (Parallel Virtual Machine)

1.2 Importancia

Académicos mexicanos empiezan a explorar la computación paralela, debido a la exigencia de rapidez de cómputo de los algoritmos, reducción de costos, y a la necesidad de sumarse al desarrollo de ésta tecnología de vanguardia. La cultura de computación paralela en México esta aún en su inicio, pero se fortalece con el paso del tiempo, debido al interés que muestran las instituciones que tienen computadoras paralelas y otras instituciones por adquirirlas, y también debido a la existencia de software de distribución gratuita para computación distribuida y paralela, como PVM.

Las instituciones en México que cuentan con computadoras paralelas son la Universidad Nacional Autónoma de México (UNAM) y el Instituto Tecnológico de Estudios Superiores de Monterrey (ITESM). Las principales aplicaciones que se ejecutan en estas máquinas son: modelos matemáticos utilizando elementos finitos y diferencias finitas y software de visualización científica.

La modelación númerica es una actividad que requiere de computación de alto rendimiento, cualquier técnica de programación nueva o tecnología de hardware de punta debe inmediatamente ser explorada en trabajos piloto.

2 ANTECEDENTES

La computación paralela (CP) es una tecnología de punta para resolver problemas computacionalmente intensivos, cuyo objetivo principal es reducir el tiempo de ejecución. Los aspectos básicos de la CP son: arquitectura de la computadora, lenguaje de programación, diseño de algoritmos, portabilidad de los programas y conversión de programas secuenciales a programas paralelos (Ragsdale, 1991).

Para realizar CP se necesita más de un procesador. El diseño de computadoras paralelas ha evolucionado en la última década. Hoy se cuenta con diferentes paradigmas en cuanto al funcionamiento e interrelación de los procesadores y los otros componentes del computador. De acuerdo a la relación memoria-procesador hay dos configuraciones básicas: memoria distribuida (Ver Fig. 1) y memoria compartida (Ver Fig. 2), también conocidas como arquitectura de paso de mensajes y arquitectura de espacio de memoria compartida, respectivamente. En los sistemas de memoria distribuida el contenido de la memoria en un nodo (procesador y demás dispositivos) puede sólo ser accesado por el procesador de ese nodo. Cuando un nodo requiere información que está almacenada en otro, la información debe ser solicitada y enviada explícitamente de nodo a nodo. La otra forma de organizar la memoria es compartirla, tal que la memoria es global. Las computadoras paralelas de este tipo generalmente obtienen acceso directo a la memoria a través de una red de interconexión.

La clasificación de computadoras mas popular es la de Flynn (Ragsdale 1991),

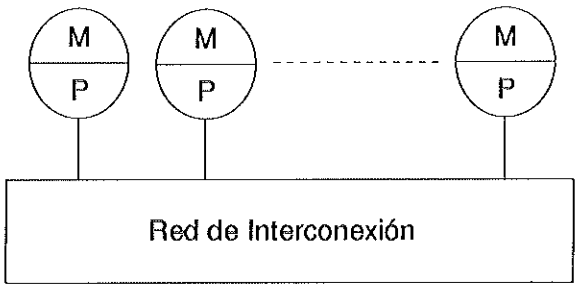

Figura 1. Diagrama de un sistema de memoria distribuida, donde M representa a la unidad de memoria y P al procesador.

y hace su clasificación en cuanto a la ejecución de instrucciones y el manejo de los datos. La clasificación es la siguiente :

- SISD (Simple Instruction stream, Single Data stream)
- SIMD (Simple Instruction stream, Multiple Data streams)
- MISD (Multiple Instruction strems, Single Data stream)
- MIMD (Multiple Instruction streams, Multiple Data streams)

Dentro de esta clasificación los sistemas más importantes para cómputo paralelo por su naturaleza son SIMD y MIMD, por lo que se enfatizará en estos.

El grupo de sistemas SISD representa a las computadoras convencionales con un solo procesador, donde cada instrucción se aplica a un dato tomado del flujo de datos.

En el grupo MISD hay muy pocas computadoras, las cuales no han tenido exito comercial o un impacto fuerte en la comunidad de cómputo. Su funcionamiento es aplicar diferentes instrucciones a un mismo flujo de datos.

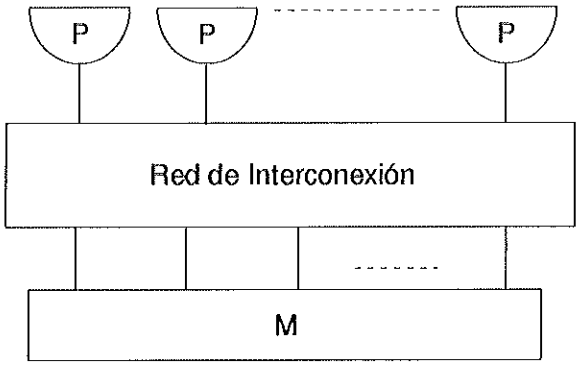

Figura 2. Diagrama de un sistema de memoria compartida, donde M representa a la unidad de memoria y P al procesador.

En los sistemas SIMD (Ver Fig. 3a) todos los procesadores operan simultáneamente con el mismo flujo de instrucciones, esto significa que, en cualquier unidad de tiempo, una operación está en el mismo estado de ejecución en todos los procesadores, cada uno operando sobre sus propios datos. La sincronización en estas máquinas es automática.

La mayoría de los sistemas con varios procesadores caen dentro de la categoría MIMD (Ver Fig. 3b). Debido a que consiste de varias unidades de procesamiento, ejecutando asincrónicamente flujos de instrucciones independientes, las máquinas MIMD son un buen ejemplo de computación paralela de propósito general. La sincronización en el modelo MIMD requiere programación. El modelo MIMD puede utilizar memoria compartida o memoria distribuida. Una máquina MIMD puede simular una máquina SIMD pero no viceversa (Ragsdale 1991).

La organización de tareas en un sistema paralelo se puede hacer al menos de

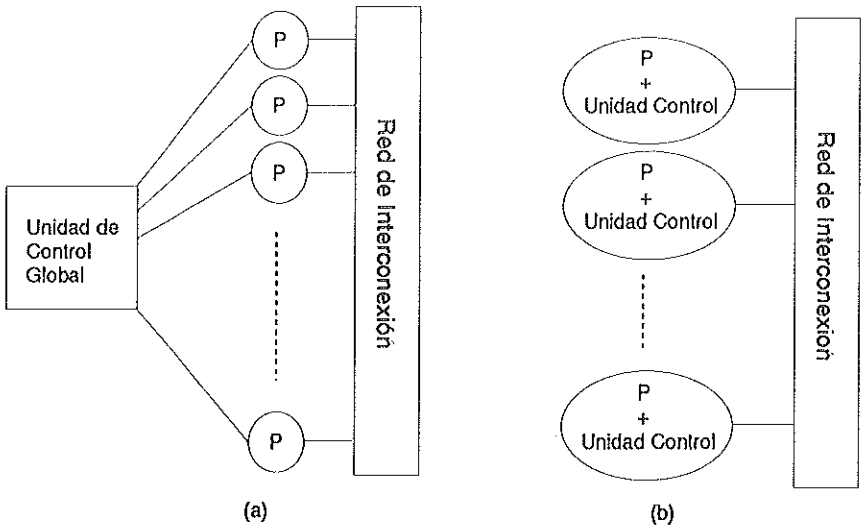

Figura 3. Diagrama de los sistemas a) SIMD, b)MIMD

tres formas. La más común se denomina computación conglomerada, la cual es una colección de procesos relacionados entre sí, ejecutando el mismo código e intercambiando periódicamente resultados intermedios. A su vez este tipo de programación puede ser dividida en dos categorías:

1. Modelo **Maestro-Esclavo**, que está formado por un programa de control llamado maestro que generalmente es el responsable de inicializar el programa, crear los procesos, recolectar y visualizar los resultados. El programa esclavo ejecuta el código y maneja los mensajes entre los nodos.
2. Modelo un sólo código o SPMD (Single Program, Multiple Data), donde el mismo código es ejecutado en todos los nodos en paralelo.

Otra forma de organizar las tareas es el modelo de computación en árbol. En el cual los procesos son creados, usualmente en forma dinámica, mientras la computación

progresa, estableciendo relaciones del tipo padre-hijo.

Finalmente se puede dar una combinación entre el modelo de árbol y computación conglomerada, denominado modelo híbrido

Cuando se diseña un programa paralelo se debe tener en cuenta el algoritmo para resolver el problema y el método mediante el cual el problema será mapeado en los nodos, entendiéndose por mapeo a la manera en que las funciones o datos son distribuidos entre los procesadores.

Los métodos de descomposición más usados son:

- Descomposición del dominio (descomposición de datos).
- Descomposición de control (descomposición funcional).

En diversas aplicaciones hay regularidad en el dominio o en la estructura de datos. La descomposición del dominio sirve para que los problemas de este tipo sean fácilmente mapeados a un sistema paralelo. Esta descomposición supone que en todo el problema se mezclan operaciones o transformaciones en una o más estructuras de datos que son generalmente estáticos. Estas estructuras de datos forman el dominio del problema, del cual se hace una partición formando subdominios. Una vez establecida la división del dominio las operaciones o transformaciones son aplicadas a cada subdominio.

En general, los pasos para aplicar la descomposición del dominio son:

1. distribuir el dominio,

2. restringir la computación a que cada procesador actualice sus propios datos,

3. comunicar los resultados.

Por ejemplo, si tenemos una matriz de datos a la cual se le aplica una transformación, se puede descomponer la matriz en dos partes y aplicar la tranformación a cada una al mismo tiempo (Fig. 4a).

La estrategia de descomposición de control, se utiliza cuando el dominio y las estructuras de datos en el problema son irregulares o impredecibles, y se debe enfocar a distribuir el flujo de control de la computación, antes que distribuir el dominio. Durante el desarrollo del programa paralelo se distribuye el dominio, pero la guía de desarrollo no es el dominio en sí mismo, sino los aspectos de control. El problema se ve como un conjunto de operaciones en término de sus funciones. Por ejemplo, si se tiene una imagen de satélite a la cual hay que aplicarle varios filtros independientes (Fig. 4b), entonces se deben aplicar los filtros al mismo tiempo.

Debido a que las computadoras paralelas son casi inaccesibles para muchos de los investigadores, por su alto costo y porque no hay muchas en operación, fue necesario desarrollar, para la comunidad académica una herramienta de computación paralela que fuera accesible en costo y funcionalidad. Generalmente hay una red de computadoras UNIX en cualquier centro de investigación, así que cualquier tecnología que las utilice representa una solución para hacer computación paralela, esto se puede hacer sin inversión monetaria alguna, dado que hay software que se puede obtener

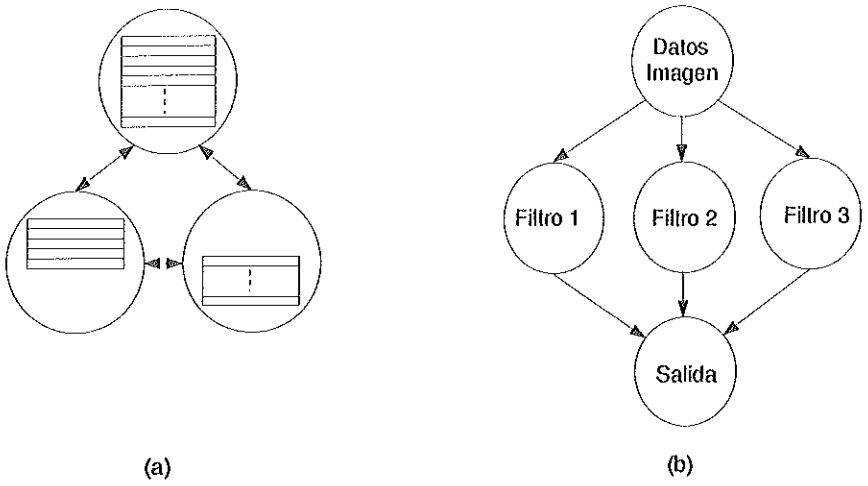

Figura 4. Métodos de Descomposición a)de Dominio, b)de Control

gratuitamente vía Internet. PVM es una de esas tecnologías.

2.1 PVM

El proyecto PVM inició en 1989 en la Universidad Emory, la Universidad de Tennessee y el Laboratorio Nacional de Oak Ridge (Geist et al, 1994), con el objetivo de implantar en una red de estaciones de trabajo, funcionando en UNIX, un ambiente computacional paralelo. Este proyecto continúa en marcha hoy en día, contando con la versión 3.3.11 a mediados de 1996.

Se utiliza PVM, entre otras partes, en compañías petroleras, en laboratorios de la NASA, en la industria farmacéutica, en la industria aeroespacial y en las universidades de casi todo el mundo, en un campo amplio de aplicaciones, entre las que destacan la dinámica de fluidos (Sethian, 1993) y la visualización científica (Pavlakos et al, 1993).

Con el software PVM se puede desarrollar programación paralela usando el hardware de la red, ya que habilita a una colección de computadoras como una máquina paralela. Maneja de forma transparente el enrutamiento de mensajes, la conversión de datos y la calendarización de las tareas a través de redes de arquitecturas diversas.

El modelo de computación de PVM es simple, pero general, y permite una variedad amplia de aplicaciones. La interfaz de programación es directa, permitiendo que estructuras simples puedan ser implantadas de manera intuitiva. El usuario programa su aplicación como una colección de tareas cooperativas a través de una biblioteca de rutinas de interfaz estándar, que permiten la iniciación y la terminación de tareas a través de la red, así como la comunicación y la sincronización de las mismas. Las primitivas que sirven para el paso de mensajes hacen operaciones heterogéneas, tienen estructuras para almacenamiento y comunicación. Las rutinas de comunicación incluyen envío y recepción de estructuras de datos, así como primitivas de alto nivel como difusión ("broadcast"), barreras a sincronización y también operaciones globales como suma y el cálculo de máximo y mínimo.

Las tareas PVM pueden tomar el control del ambiente paralelo. De forma tal que en cualquier punto en la ejecución de una aplicación concurrente, cualquier tarea puede iniciar o parar a otra tarea, o añadir o borrar computadoras del ambiente paralelo. También cualquier proceso puede comunicarse y sincronizarse con otro. Cualquier control específico y estructura dependiente pueden ser implantados dentro del sistema PVM, con el uso apropiado de los archivos de construcción del mismo

paquete y las especificaciones de la computadora huésped.

El sistema PVM está compuesto de dos partes. La primera es un proceso Unix, "daemon", llamado *pvmd*, que se ejecuta en todas las estaciones de trabajo involucradas manteniendo el funcionamiento de las computadoras en paralelo. Funciona como un controlador y enrutador, proveé el punto de contacto, autorizar o autenticar, control de procesos y detección de fallas. Aún cuando una aplicación fracase o aborte, los procesos *pvmd* continúan ejecutándose para ayudar en la depuración del proceso. Un *pvmd* denominado *pvmd* maestro se inicializa en la computadora huésped, mientras que los otros *pvmd* son inicializados por el *pvmd* maestro y se les denomina *pvmd* esclavos. Durante una operación todos son considerados iguales, pero sólo el maestro puede inicializar nuevos esclavos y añadirlos a la configuración. La segunda parte del sistema es una biblioteca de rutinas de interfaz de PVM, denominada *libpvm*. Esta biblioteca contiene rutinas que son usadas por el usuario para el paso de mensajes, creación de procesos, coordinación de tareas y modificación del sistema. *Libpvm* está escrita en lenguaje C y PVM soporta aplicaciones escritas en lenguaje C, C++ y FORTRAN.

Los componentes del sistema PVM han sido instalados, compilados y probados en las arquitecturas mostradas en la Tabla I. Cray Research, IBM, Convex, Intel, SGI y DEC ofrecen sus propias versiones para sus sistemas paralelos.

Arquitectura y Sistema Operativo (SO)

Alliant FX/8.
DEC Alpha/OSF-1.
DEC Alpha multiprocesador con SO >= 3.0
Sequent Balance
BBN Butterfly TC2000
80[34]86 corriendo BSDI, 386BSD, NetBSD, FreeBSD
Thinking Machines CM-2 Sun front
Thinking Machines CM-5
Convex usando IEEE floating-point
Convex usando floating-point nativo
Cray
Cray-2
Cray S-MP
Cray T3D
Convex Ejemplar SPP
Data General Aviion
Encore 88000
HP 9000 68000 cpu
HP 9000 PA-Risc
Intel RX Hypercube
Intel IPSC/2
Kendall Square
80[34]86 corriendo Linux
Maspar/Dec Mips front-end
Mips
NeXT
Intel Paragon
DEC/Mips arcquitectura (3100, 5000, etc.)
IBM Power-4
IBM/RS6000
IBM/RT
80[34]86 corriendo SCO Unix
Silicon Graphics IRIS
Silicon Graphics IRIS OS >= 5.0
Silicon Graphics IRIS multiprocesador con SO >= 5.0
Silicon Graphics IRIS OS >= 6.0
Silicon Graphics IRIS multiprocesador con SO >= 6.0
Sun 3
Sun 4, 4c, sparc, etc.
Sun 4 corriendo Solaris
Sun 4 multiprocessor
NEC SX-3
Sequent Symmetry
Stardent Titan
DEC/Microvax
Fujitsu corriendo UXP/M
Thinking Machines CM-2 Vax front

Tabla I. Arquitecturas donde el sistema PVM ha sido probado. Tabla obtenida de la distribución del software PVM versión 3.3 1994.

2.2 Elemento Finito

El método de elemento finito (MEF) es un procedimiento numérico para resolver ecuaciones diferenciales de la física e ingeniería. El concepto fundamental del MEF es que cualquier cantidad continua, tal como temperatura, presión o desplazamiento, puede ser aproximada por un modelo discreto compuesto de un conjunto de funciones seccionalmente continuas definidas sobre un número finito de subdominios. El método nació en la industria aeroespacial a principio de los 50's.

Puesto que se puede aplicar a todos los problemas que son gobernados por ecuaciones diferenciales, el MEF es ampliamente usado. Algunas ventajas son descritas a continuación:

1. Las propiedades del material en elementos adyacentes no tiene que ser las mismas. Esto permite que el método sea aplicado a cuerpos compuestos de diversos materiales.
2. Dominios con fronteras irregulares pueden ser aproximadas usando elementos con lados rectos o elementos con fronteras curvas.
3. El tamaño de los elementos puede ser modificado. Esta propiedad permite que el elemento sea expandido o redefinido cuando sea necesario.

Una desventaja del MEF es la complejidad de la programación.

En el MEF, una vez definido el modelo discreto, se deben ajustar los valores de las cantidades continuas de forma tal que resulte la mejor aproximación posi-

ble a la geometría del problema físico. Entonces, una funcional relacionada con la ecuación diferencial gobernante es minimizada, dando como resultado un conjunto de ecuaciones algebraicas lineales que pueden ser resueltas con métodos directos o con métodos iterativos. El modelo discreto se construye de la manera siguiente:

1. Se identifica un número finito de puntos en el dominio, estos puntos son llamados puntos nodales o nodos.
2. Se divide el dominio en un número finito de subdominios denominados elementos, los cuales se conectan a otros elementos por medio de nodos comunes, que colectivamente se aproximan a la figura del dominio.
3. Las cantidades continuas se aproximan sobre cada elemento por un polinomio. Cada elemento tiene asociado un polinomio diferente, los términos del polinomio se seleccionan de manera tal que se mantenga continuidad en todas las fronteras del elemento.

La manera de etiquetar los nodos y enumerar los elementos es crucial en el método. El etiquetado eficiente de los nodos trae consigo que la matriz asociada al sistema de ecuaciones contenga un *ancho de banda* mínimo, lo cual se puede aprovechar para ahorrar memoria.

2.3 La Ecuación de Poisson

Un modelo matemático de un acuífero rectangular donde el nivel del líquido baje súbitamente en un lado, se puede construir a partir del principio de conservación de volumen y de la ley de Darcy. En este caso, la conservación de volumen se obtiene de la igualdad entre el volumen de agua que sale del acuífero y el volumen de agua que entra más la razón en la cual el nivel de agua cambia con el tiempo. La ley de Darcy establece que la razón de flujo de volumen por unidad de área es directamente proporcional al gradiente del nivel del agua. A partir de estos dos principios se puede obtener directamente

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = \frac{S}{T}\frac{\partial h}{\partial t}, \tag{1}$$

donde $h(x, y, t)$ es la caída del nivel agua, S es el coeficiente de almacenamiento y T es la transmisividad, x y y son las coordenadas cartesianas y t el tiempo.

El acuífero confinado rectangular tiene las caracteristícas siguientes: la base va de $x = 0$ a $x = l = 100$ m, en la vertical el nivel agua es inicialmente, $t = 0$, 10 m. Repentinamente cae el nivel del agua en $x = l$, de 16 m a 10 m. Además los parámetros del acuífero son $T = 0.02$ m^2 min^{-1} y $S = 0.002$. Las condiciones a la frontera son $h(0, t) = 16$ m, $h(l, t) = 11$ m para $t > 0$. Asimismo, la condición inicial es $h(x, 0) = 164$ m en $0 \leq x \leq l$. Una solución aproximada se puede encontrar con el MEF.

2.4 Objetivos

2.4.1 General:

Mostrar la viabilidad de la computación paralela en computación científica y cuantificar la capacidad del recurso computacional paralelo PVM en una red local heterogénea de estaciones de trabajo.

2.4.2 Particular:

Implantar en la red local heterogénea de CICESE con ambiente PVM un programa paralelo de la ecuación del flujo transitorio en un acuífero, discretizada en elementos finitos.

3 METODOLOGÍA

Se usa el trabajo de Wang y Anderson (1982) en la solución numérica del problema del flujo transitorio. Los pasos son: se emplea el método de Galerkin para resolver vía elementos finitos la Ec. 1, de ese modo se obtiene un sistema de ecuaciones diferenciales que contienen derivadas de primer orden con respecto al tiempo. Este sistema de ecuaciones se puede escribir en forma matricial, y para su solución se utilizan métodos iterativos. El espacio del dominio del problema se divide en elementos rectangulares, los cuales están determinados por los nodos. Entonces se hace un programa secuencial C (Ver apéndice A).

Se utiliza el método de computación conglomerada para paralelizar el programa secuencial. Se emplea el modelo **Maestro-Esclavo** en la organización de las tareas. Se usa el método de descomposición del dominio para manejar el dominio físico, el cual se divide en un número determinado de subdominios (los elementos finitos) y los cálculos se hacen por separado en cada subdominio. El etiquetado de los nodos relacionados a cada elemento se eligen según Wang y Anderson (1982). Además, en cada una de las ejecuciones de los programas paralelos en este trabajo, a cada procesador le corresponde sólo un proceso, ya sea el proceso esclavo o el proceso maestro.

El programa **Maestro** hace las tres funciones principales de interfaz: (1) crear los procesos esclavos, (2) inicializar esclavos y (3) recolectar resultados en cada iteración. Gran parte de los datos son inicializados en cada proceso esclavo.

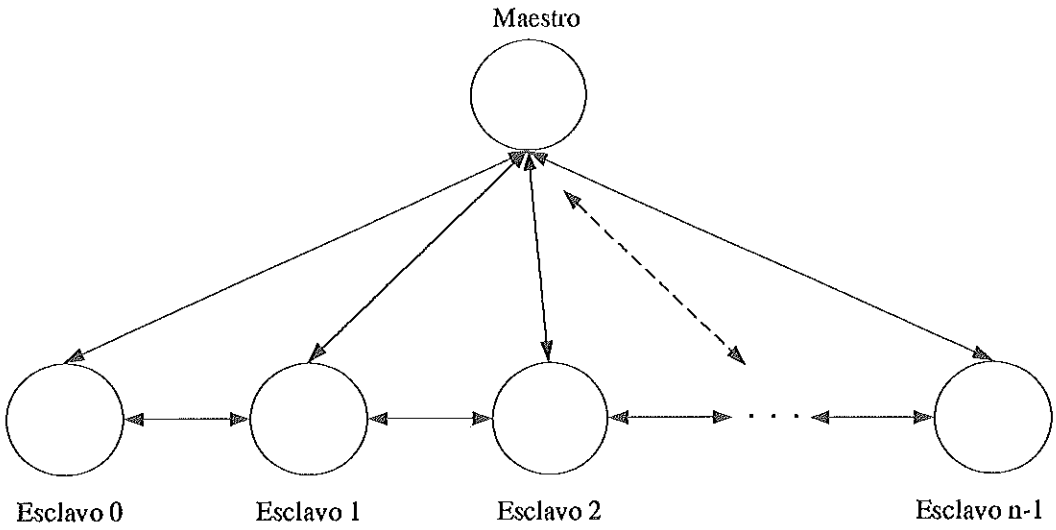

Figura 5. Topología de comunicación en el modelo Maestro-Esclavo.

En el programa esclavo se realiza toda la carga de computación así como la inicialización de los datos. Los esclavos mantienen comunicación con el maestro enviando los resultados, al igual que entre ellos mismos. La Fig. 5 muestra la manera en que se dan las relaciones **Maestro-Esclavo** y esclavo-esclavo.

Enseguida, se realiza el mapeo asignando el mismo número de elementos finitos a cada procesador. Si el número de elementos finitos no es divisible exactamente por el número de procesadores, se asigna un elemento finito sobrante a cada procesador en orden ascendente. Los elementos finitos asignados a cada procesador son consecutivos, por ejemplo, si se tienen 6 elementos y 4 procesadores el mapeo siguiendo el etiquetado de los nodos descrita por Wang y Anderson (1982) sería el que muestra la Tabla II.

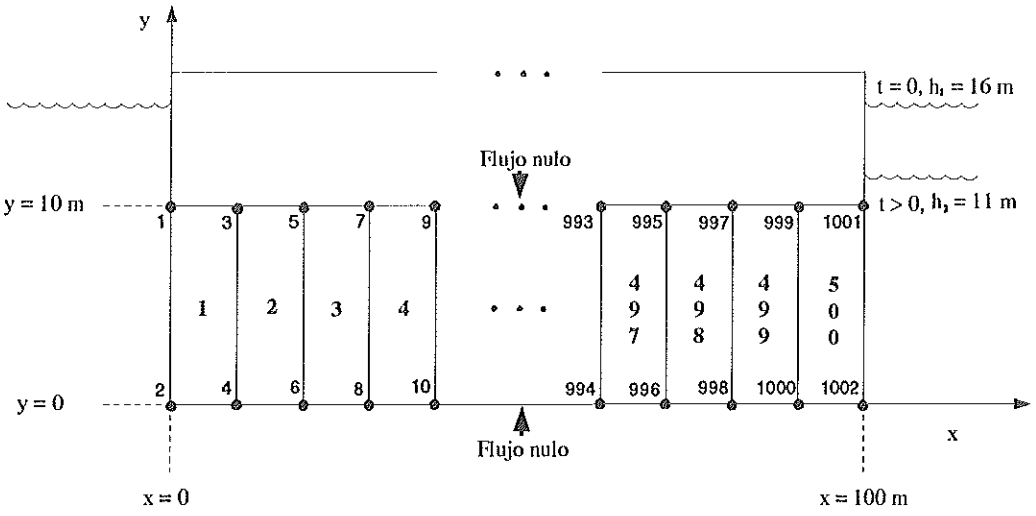

Figura 6. Diagrama del acuífero rectangular dividido en 500 elementos.

Procesador	Elemento Finito	Nodos
1	1,2	1,2,3,4,5,6
2	3,4	5,6,7,8,9,10
3	5	9,10,11,12
4	6	11,12,13,14

Tabla II. Ejemplo de mapeo de los elementos finitos.

Al realizar el mapeo se observa que cada procesador comparte al menos dos nodos frontera con los procesadores vecinos. Por claridad, en la Figura 6 se muestran el dominio, los elementos finitos y los nodos del problema.

El primer bloque del programa secuencial es de inicialización (Ver Apén. A), su paralelización involucra tanto al programa **Maestro** como al programa **Esclavo**. En

el programa **Maestro** se inicializan las variables *nnode* (número de nodos del MEF) y *nproc* (número de procesadores en la configuración). En el programa **Esclavo** primero se calcula el número de elementos del modelo *nelem = (nnode/2) -1*, y después se efectúa el mapeo del dominio del problema, seguido de la inicialización de las condiciones iniciales.

En el bloque dos del programa secuencial se construyen las matrices correspondientes al sistema de ecuaciones, las cuales se integran con el método de cuadratura gausiana. Este bloque se escribe en el programa esclavo. Contando con la descomposición, el mapeo e inicialización de los datos, el bloque dos del programa **Esclavo** es semejante al bloque dos del programa secuencial, sólo que aplicado a un subdominio del problema y teniendo como resultado un subdominio de las matrices que representan al sistema de ecuaciones. La dimensión de las submatrices esta dada por el número de nodos mapeado en cada esclavo que representan los renglones, por el número de nodos totales del dominio que representan las columnas.

Cómo se mostro en la Tabla II al hacer el mapeo se tiene que cada esclavo comparte al menos dos nodos con otro esclavo, dando como resultado que los valores en las submatrices obtenidas sean parciales en al menos dos renglones que representan los nodos compartidos. Para solucionar este problema se codifica el bloque dos en el programa esclavo, de acuerdo a los siguientes pasos:

1. El esclavo r envía al esclavo $r+1$ los dos renglones que comparte con él.

2. El esclavo r recibe los dos renglones del esclavo $r-1$.

3. Se completan los valores de los renglones recibidos, en todos los esclavos intermedios y el último.

4. El esclavo r envia los valores actualizados al esclavo $r-1$.

5. El esclavo r recibe del esclavo $r+1$ los valores actualizados enviados en el paso 1.

Estos cinco pasos sólo los efectúan los esclavos intermedios, el esclavo 0 hace los pasos 1 y 5, mientras que el esclavo n-1 realiza los pasos 2,3 y 4. La Figura 7 muestra los pasos anteriores y la matriz R de cada proceso en los cinco pasos. Los renglones de la matriz sombreados representan los renglones compartidos que están incompletos, y los renglones en blanco representan los renglones que están completos.

En el tercer bloque del programa secuencial se resuelve el sistema de ecuaciones que resulta del segundo bloque. Los métodos usados para resolver el sistema de ecuaciones son los iterativos de Gauss-Seidel y Jacobi se sigue a Kumar et al. 1994 en el proceso de paralelización.

3.1 MATERIALES

1. **Red local de computadoras con UNIX como sistema operativo.**

   Una red local de computadoras LAN (Local Area Network) es un sistema de comunicación compuesto de hardware y software. El hardware consiste en los cables y dispositivos que conectan una computadora con otras

BIBLIOTECA CENTRAL ENSENADA U.A.B.C.

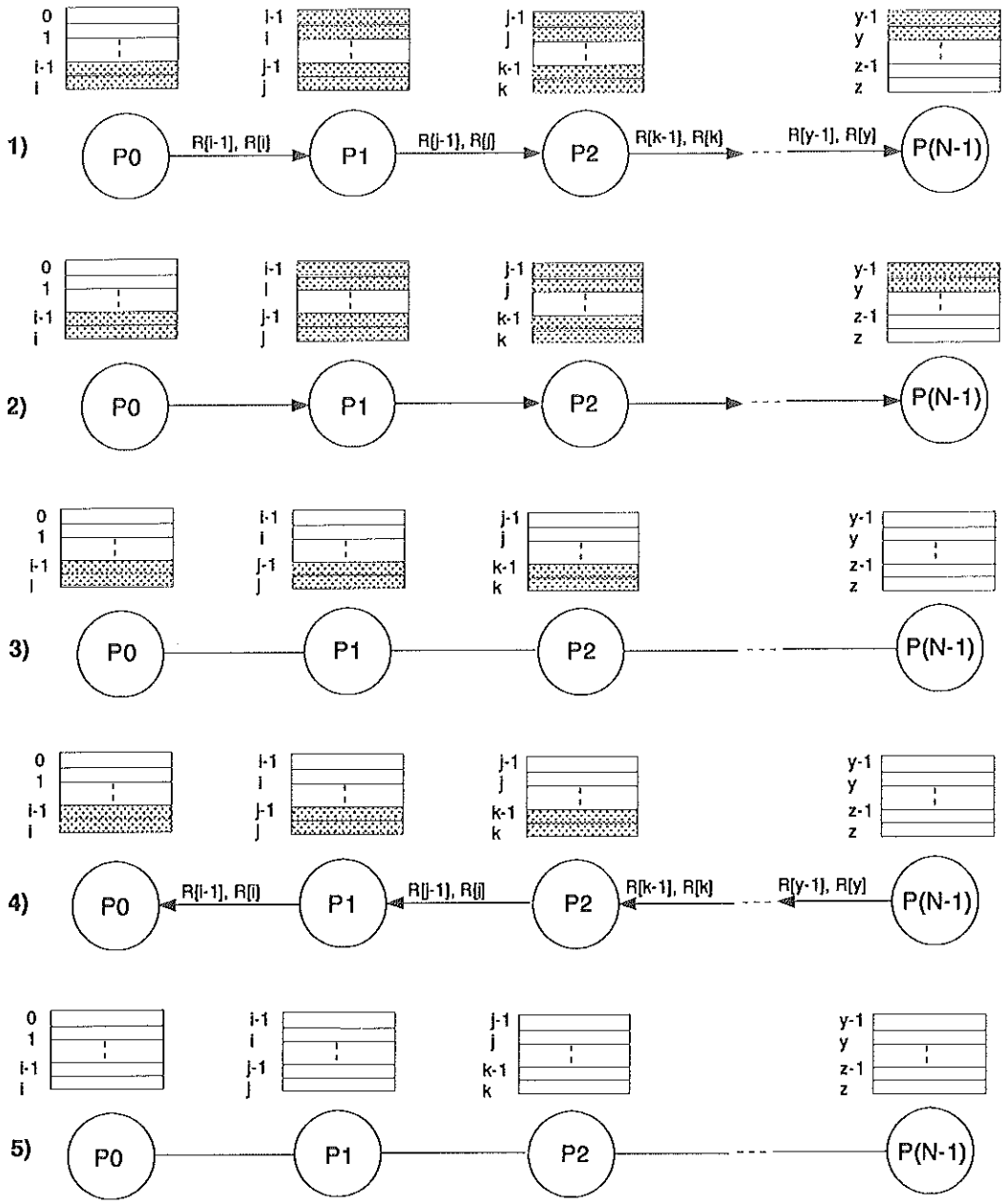

Figura 7. Secuencia de pasos en el bloque dos del programa esclavo, donde PX es el proceso X, R[y] es el renglón y de la matriz R

computadoras o periféricos. El software maneja los archivos y el sistema de comunicación. Un grupo de computadoras en red pueden comunicarse entre sí y compartir recursos.

Las computadoras disponibles en la red pueden presentar diferentes grados de heterogeneidad en: arquitectura, representación de datos, rapidez de computación, carga de trabajo de la máquina y carga de trabajo de la red, entre otros. La red utilizada en este trabajo es la sub-red heterogénea de estaciones de trabajo SUN del Depto. de Oceanografía Física de CICESE. Esta red local es *Ethernet* con topología de *ducto* utilizando CSMA/CD como protocolo. Las estaciones usadas son 2 SPARC SLC, 1 SPARC 1+, 2 SPARC IPC, 1 SPARC IPX .

2. **Lenguaje de programación FORTRAN.**

FORTRAN es un lenguaje de programación de alto nivel, orientado a operaciones aritméticas. Fue originalmente desarrollado para computación científica, y lanzado al mercado por IBM Corporation en 1957. En 1966 fue estandarizado por primera vez y tomado como el lenguaje de programación científica. Hoy en día FORTRAN cuenta con versiones y extensiones que lo han hecho pasar de ser un lenguaje estrictamente orientado a aplicaciones numéricas, a realizar aplicaciones más generales, involucrando manipulación de caracteres y archivos. Las estaciones SUN utilizadas cuentan con un compilador FORTRAN 77 V1.4.

3. **Lenguaje de programación C.**

   El lenguaje de programación C es un lenguaje estructurado de alto nivel y de propósito general. Fue desarrollado a principio de los 70's por los laboratorios Bell, pero no fue hasta el final de los 70's, que empezó a tener popularidad, cuando los compiladores estuvieron disponibles comercialmente. Otro punto que ayudó a su popularidad fue la aceptación del sistema operativo UNIX, que tomó al lenguaje C como su lenguaje de programación estándar. Más del 90% del sistema operativo UNIX está escrito en lenguaje C. Como el lenguaje C fue diseñado para la programación de aplicaciones como sistemas operativos, el programador C puede hacer modificaciones a su conveniencia. Las estaciones SUN utilizadas cuentan con un compilador C nativo del sistema operativo SunOS versión 4.1.

4. **Software PVM.**

   PVM es un conjunto integrado de herramientas de software y bibliotecas que simulan una estructura de computación concurrente heterogénea de propósito general y flexible, en computadoras interconectadas variadas en arquitectura. El sistema PVM utiliza el modelo de paso de mensajes para realizar computación paralela .

   Obtención de PVM

   El software PVM esta disponible en netlib, el cual es un servicio de dis-

tribución de software en Internet. Se puede obtener PVM vía ftp, WWW, xnetlib o email.

Vía **ftp** se obtiene conectándose a netlib2.cd.utk.edu, como anónimo (anonymous), buscando en el directorio pvm3, el archivo index describe el contenido de los archivos y los subdirectorios.

Vía WWW la dirección es http://www.netlib.org/pvm3/index.html

Desempacar PVM

El software PVM está empacado en formatos comunes en UNIX. Para desempacar sólo se verifica el formato de empaque dado por los últimos caracteres del nombre del archivo y se ejecuta la aplicación para desempacar ese formato.

Instalación y Construcción de PVM

La instalación y construcción de PVM es sencilla, los pasos a seguir están en los archivos de referencias que vienen incluidos en el software PVM.

Paso de Mensajes con PVM

Las funciones principales involucradas en el paso de mensajes se describen a continuación: Al enviar un mensaje en PVM se requiere de tres pasos. Primero debe ser inicializado el buffer de envío , llamando a la función

pvm_initsend(), cuyos argumentos y sintaxis de invocación se muestran a continuación:

int bufid = pvm_initsend(int cdigo),

donde

bufid : el identificador del buffer creado,

cdigo : PvmDataDefault, codificación XDR,
PvmDataRaw, sin codificar,
PvmDataInPlace, empaca hasta que el mensaje es enviado.

Enseguida el mensaje debe ser empacado en este buffer usando cualquier número y combinación de las funciones pvm_pk*(), e.g., la función pvm_pkint(), la cual empaca un número entero de la forma siguiente:

int info = pvm_pkint(int *i, int ndat, int avance),

donde

info : -1 si la llamada a la función falló,

i : vector de enteros,

ndat : número de datos a empacar,

avance : avance en el vector i.

Después el mensaje se envía a otro proceso llamando a la función pvm_send() o pvm_mcast() para un mensaje múltiple, en la forma :

int info = pvm_send(int tid, int msjid),

donde

info : -1 si la llamada a la función falló,

tid : identificador de tarea a la que se manda el mensaje,

msjid : identificador de mensaje.

o

int info = pvm_mcast(int *tids, int ntareas, int msjid),

donde

tids : arreglo de identificadores de tareas a los que manda el mensaje,

ntareas: número de tareas en el arreglo tids.

Un mensaje se recibe invocando rutinas de recepción con bloqueo, se espera hasta que el mensaje sea recibido, o sin bloqueo, después se utilizan las funciones pvm_upk*() para desempacar cada uno de los datos en el buffer recibido. Las rutinas de recepción de mensajes pueden usarse para

recibir cualquier mensaje, ya sea de una fuente específica y un identificador específico o mensajes con un identificador y fuente no determinados. La funciones de recepción con bloqueo y sin bloqueo son

int bufid = pvm_recv(int tid, int msjid) y

int bufid = pvm_nrecv(int tid, int msjid), respectivamente,

donde

bufid : identificador del buffer recibido,

tid : identificador de la tarea de quien se desea recibir,

msjid : identificador del mensaje que se desea recibir.

4 DESARROLLO

4.1 Solución por métodos iterativos

Para resolver el sistema de ecuaciones algebraicas que resulta del método de Galerkin se usaron los métodos iterativos Gauss-Seidel y Jacobi. Originalmente se tenía sólo los resultados del problema con el método Gauss-Seidel programado en FORTRAN, en este trabajo se agregó el método de Jacobi programado en C con la finalidad de comparar el grado de paralelismo entre ellos. Las salidas de los programas secuenciales aplicando cada uno de los métodos fueron equivalentes. A continuación se detalla el proceso de paralelización en cada uno de ellos.

4.1.1 Método de Jacobi

El algoritmo de Jacobi es un método estacionario que se basa en resolver localmente cada variable con respecto a las otras. En cada iteración se barre todo el dominio de definición en la forma siguiente

$$X_k[i] = \frac{1}{A[i,i]}(b[i] - \sum_{j=1}^{i-1} X_{k-1}[j]A[i,j] - \sum_{j=i+1}^{n} X_{k-1}[j]A[i,j]), \qquad (2)$$

donde i va de 1 a n, y la primera suma es el producto entre los coeficientes a la izquierda de la diagonal principal de la matriz y el correspondiente X_k y la

segunda es el producto entre los coeficientes a la derecha de la diagonal principal y el correspondiente X_k. El método de Jacobi es sencillo de implantar en un algoritmo secuencial pero una de las desventajas radica en que converge lentamente.

Este algoritmo luce paralelizable, dado que para calcular X_k se utilizan sólo los valores de la iteración anterior o sea $X_{k-1}[i]$, es decir los $X_k[i]$ para $1 \leq i \leq N$ se pueden calcular al mismo tiempo.

Después de calcular en paralelo X_k se necesita que todos los procesos comuniquen los resultados a los demás procesos para poder calcular X_{k+1}.

Una desventaja de utilizar el método de Jacobi para implantarlo en paralelo es que se necesita un mayor número de iteraciones para converger a una solución y no siempre converge. El mayor número de iteraciones para encontrar una solución con respecto a otro método, digamos Gauss-Seidel, trae como consecuencia un mayor número de mensajes lo cual aumenta el consumo de tiempo en comunicación.

La Figura 8 muestra la manera en que trabajan y se comunican los procesos. Y el tiempo de ejecución de la iteración es

$$T_{k+1} \leq Max(T0, T1, T2, T3) + Max(tt0 + tt1 + tt2 + tt3), \qquad (3)$$

donde T0, T1, T2, T3 son los tiempos de ejecución de los procesos 0,1,2 y 3 y los demás términos se refieren a la Figura 8.

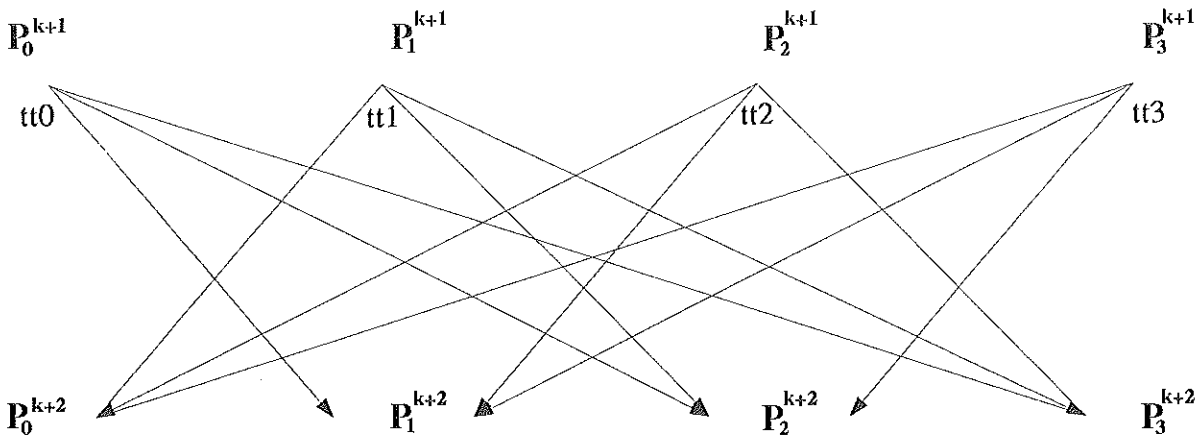

Figura 8. Diagrama del método Jacobi, donde P_s^{k+1} es el proceso s en la iteración $k+1$, tts es el tiempo de transmisión de mensaje del proceso s a todos los procesos restantes.

4.1.2 Método Gauss-Seidel

El método Gauss-Seidel es estacionario parecido al método de Jacobi excepto que usa valores actualizados tan pronto como están disponibles de la manera siguiente

$$X_k[i] = \frac{1}{A[i,i]}(b[i] - \sum_{j=1}^{i-1} X_k[j]A[i,j] - \sum_{j=i+1}^{n} X_{k-1}[j]A[i,j]), \tag{4}$$

donde i va de 1 a n, y la primera suma es el producto entre los coeficientes a la izquierda de la diagonal principal de la matriz y los valores ya calculados X_k y la segunda es el producto entre los coeficientes a la derecha de la diagonal principal y el correspondiente X_{k-1}. Converge más rápido que el método de Jacobi, aunque para algunos problemas su razón de convergencia es aún ineficiente. Es la base de otros

métodos estacionarios.

Existen dos casos concretos, primero si A es una matriz densa, es decir que la mayoría de los elementos de A son diferentes de cero, el algoritmo tiene un bajo grado de paralelización, dado que el valor $X_k[i]$ debe ser calculado antes de calcular $X_{k+1}[i+1]$ para toda $1 \leq i < N$. Sin embargo, si la matriz A es esparcida, es decir la mayoría de los elementos de A son cero, para la mayoría de los casos el cálculo de $X_k[i]$ no necesita esperar hasta que $X_k[0], ..., X_k[i-1]$ sean calculados, sino que si $A[i,j]$ es cero, tenemos que $X_k[i]$ en el lado izquierdo de la Ec. 4 no depende de los valores $X_k[j]$. Esto es, $X_k[i]$ puede ser calculada junto con los $X_k[j]$ para toda $j < i$ y $A[i,j] \neq 0$. Dado que el cálculo de $X_k[i]$ depende sólo de los elementos $A[i,j]$, diferentes de cero con $j < i$, el grado de paralelismo en el método Gauss-Seidel será función del patrón de esparcimiento en la parte inferior de A.

Por otro lado, una matriz A es simétrica si $A[i,j] = A[j,i]$ para toda i,j. La matriz A es en banda si todos los elementos $A[i,j] \neq 0$ se encuentran concentrados alrededor de la diagonal principal. En general, una matriz A para la cual $A[i,j] = 0$ si $j > i+p$ o $i > j+q$ es llamada matriz en banda con un ancho de banda igual a p+q+1. Por ejemplo una matriz con elementos diferentes de cero sólo en la diagonal principal tendrá $p = 0$ y $q = 0$ y un ancho de banda igual a $p+q+1 = 1$.

La implantación del método Gauss-Seidel se muestra en esta parte, en este caso la matriz A es una matriz simétrica en banda con un ancho de banda igual a 7, con p y q igual a 3. Los renglones de la matriz están distribuidos entre los procesos esclavos

en forma continua.

Desde el punto de vista secuencial, para poder calcular $X_{k+1}[i]$ se necesita de los valores $X_{k+1}[i-3], X_{k+1}[i-2]$, $X_{k+1}[i-1]$ y $X_k[i+1], X_k[i+2]$, $X_k[i+3]$, que corresponden a los elementos de A renglon i que forman parte de la banda o posibles no ceros. Si $i-3, i-2, i-1$ son cada uno menores que cero no se toman en cuenta, de igual forma si $i+1, i+2, i+3$ son cada uno mayores que el número de nodos. Con lo anterior tenemos que para calcular $X_{k+1}[i]$ se deben de conocer tres valores anteriores ($X_{k+1}[i-3], X_{k+1}[i-2]$, $X_{k+1}[i-1]$) de la iteración $k+1$. Visto de esta manera se puede decir que el algoritmo no es paralelizable dado que ninguna $X_{k+1}[i]$ puede ser calculada al mismo tiempo que $X_{k+1}[j]$ para $j < i$.

Analizando el paso de mensajes entre los procesos que contienen parte de la matriz A y parte del vector solución X, se observa que en el proceso 1 (P_1), para calcular $X_{k+1}[p]$ donde p es el primer índice local, se necesitan los tres últimos valores X_{k+1} del proceso 0 (P_0). En general en todos los procesos, menos en el primero, se necesita conocer los últimos tres valores X_{k+1} correspondientes al proceso anterior para calcular $X_{k+1}[p]$. Para el conocimiento de los tres valores X_{k+1} es necesario la comunicación entre los nodos. La Figura 9 muestra la comunicación entre los procesos para pasar los valores de X_{k+1}. El tiempo estimado en la iteración $k+1$ es

$$T_{k+1} = T0 + T1 + T2 + T3 + tt1 + tt2 + tt3, \tag{5}$$

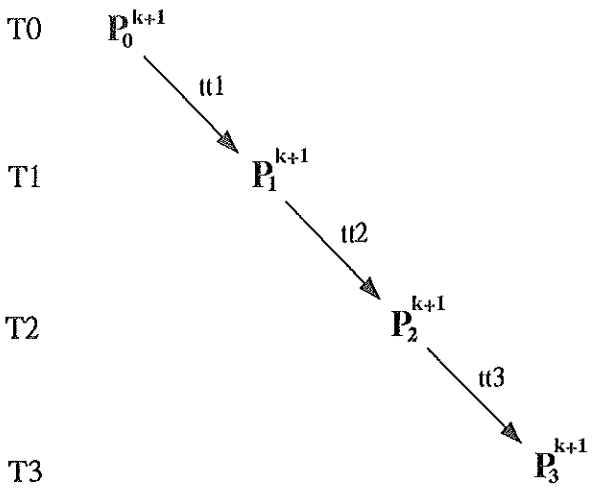

Figura 9. Diagrama del método Gauss-Seidel, donde P_s^{k+1} es el proceso s en la iteración $k+1$, tts es el tiempo de transmisión de mensaje del proceso $s-1$ al proceso s y Ts es el tiempo de ejecución del proceso s.

donde los términos se refieren a la Figura 9. Así el tiempo que tarda el algoritmo paralelo es mayor que el secuencial, dado que en el programa paralelo cada esclavo calcula sus X_{k+1} después que el proceso anterior, igual que se hace en el programa secuencial, además se agrega el tiempo de comunicación.

Ahora, si analizamos la iteración siguiente $k+2$, se puede ver que el P_O puede iniciar esta iteración sin conocer todos los valores de X_{k+1}, dado que sólo necesita $X_{k+2}[i+1], X_{k+2}[i+2] y X_{k+2}[i+3]$ para calcular $X_{k+2}[i]$, sólo faltarían los tres primeros valores del P_1, $X_{k+1}[p], X_{k+1}[p+1]$ y $X_{k+}1[p+2]$, para calcular los tres últimos valores X_{k+2} de P_0. Esto quiere decir que P_0 puede calcular los $X_{k+2}[i]$, excepto sus tres últimos, en forma paralela con los X_{k+1} de P_1.

Resumiendo, en general cada esclavo, excepto el último, requiere los tres primeros valores de X_k calculados en el proceso posterior, y cada proceso, excepto menos el primero, requiere los tres últimos valores de X_k calculados en el proceso anterior.

La Figura 10 muestra la forma en que los nodos se comunican. El tiempo estimado de la iteración es:

$$T_{k+2} \leq T0 + tt1 + Max(T0 + tc1, T1) + Max(tt1, tt2) + Max(T0 + tc1, T1, tc2, T2) +$$

$$2Max(T0 + tc1, T1 + tc2, T2 + tc3, T3) + Max(tt1, tt2, tt3), \qquad (6)$$

donde los términos se refieren a la Figura 10 y se supone que cada proceso S consume el mismo tiempo (TS) en cada una de las iteraciones y el tiempo de transmisión de un proceso E a un proceso F (ttf, tcs) es siempre igual.

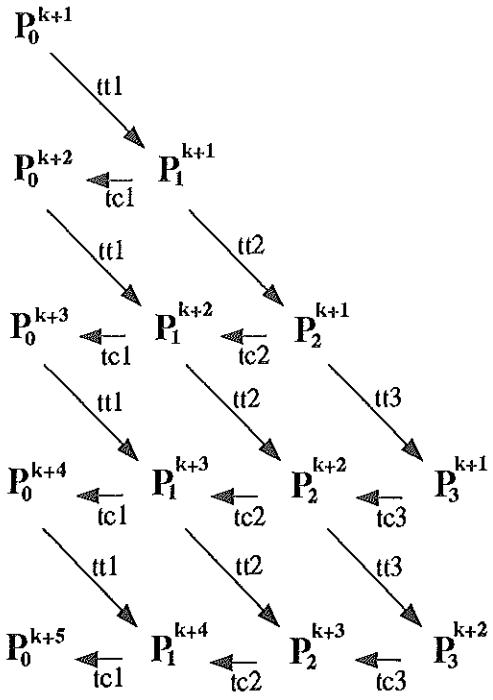

Figura 10. Diagrama del método Gauss-Seidel, donde P_s^{k+1} es el proceso s en la iteración $k+1$, Ts es el tiempo de ejecución del proceso s, tts es el tiempo de transmisión de mensaje del proceso $s-1$ al proceso s y tcs es el tiempo de transmisión de mensaje del proceso s al proceso $s-1$.

5 RESULTADOS

En el Apéndice B se muestra el programa C desarrollado a partir del trabajo de Wang y Anderson (1982). Se aprovechó la ventaja que proporciona el lenguaje C en el manejo de memoria dinámica para inicializar los arreglos vectoriales y matriciales.

5.1 Análisis de Rendimiento

El tiempo de ejecución de los programas secuencial y paralelo de cada uno de los métodos estan en la Tabla III.

Tiempo Ejecución en segundos	
Gauss-Seidel	Jacobi
2175.8	2238.2

Tabla III. Tiempos de ejecución de los programas secuenciales.

La Tabla IV muestra los tiempos y el número de procesadores empleados en la ejecución del programa paralelo para los métodos iterativos de Gauss-Seidel y Jacobi. Los tiempos que se muestran son un promedio de 5 ejecuciones con características similares como la carga de la computadora y la carga de trabajo en la red, asi como utilizar las mismas computadoras. Se utilizó el comando *time* del Unix para la medición de tiempo.

Tiempo de Ejecución en segundos		
Procesadores	Gauss-Seidel	Jacobi
3	1809.1	1142.2
4	1438.8	1073.1
5	1301.7	1389.5
6	1141.3	1792.7

Tabla IV. Tiempos de ejecución contra número de procesadores en los programas paralelos.

El speedup (incremento de la rapidez de procesamiento) definido como:

$$S = \frac{Tsec}{Tpar},$$

donde $Tsec$ es el tiempo de ejecución del algoritmo secuencial y $Tpar$ es el tiempo de ejecución del algoritmo paralelo. El "SpeedUp" ideal debe ser igual al número de procesadores utilizados por el algoritmo paralelo.

La Tabla V muestra el rendimiento de los algoritmos utilizando la Tabla IV.

	speedup	
Procesadores	Gauss-Seidel	Jacobi
3	1.2	1.96
4	1.51	2.09
5	1.67	1.61
6	1.91	1.25

Tabla V. speedup.

6 DISCUSIÓN

El presente estudio desarrolla dos algoritmos paralelos de la solución numérica de la ecuación de Poisson en elementos finitos en PVM usando un ambiente de red.

El desarrollo de algoritmos paralelos y su evaluación en una red local heterogénea de estaciones de trabajo es un área de investigación activa en las Ciencias Computacionales. Debido a que las estaciones de trabajo tienen diferentes procesadores, en este tipo de investigaciones es común encontrar el problema del balanceo de carga de los procesadores. También cuando el número de estaciones de trabajo se incrementa, generalmente se incrementa el número de mensajes, lo cual da lugar al problema de escalabilidad.

En este trabajo el problema del balanceo de procesadores fue resuelto en parte cargando siempre procesadores con aproximadamente el mismo poder de cómputo, por ejemplo las SPARC SLC y SPARC IPC son las más comunes de la red y fueron las que más se utilizaron en los experimentos realizados. Otro punto de interés a considerar en la computación paralela en redes es que la red no está cien por ciento disponible para este fin, sino que son redes de uso múltiple. De forma tal que al desarrollar programas en paralelo o hacer investigación científica usando estos códigos, se debe compartir los recursos computacionales existentes.

Por otro lado, el problema de escalabilidad es más complejo, pues involucra a todos los desbalances posibles: en los procesadores, en la carga computacional de cada estación de trabajo, en la carga de la red, y se suman los problemas de paralelización

de los algoritmos, entre otros.

Con respecto a los métodos empleados para resolver el sistema lineal $Ax = B$, el método de Jacobi es fácilmente paralelizable dado que las X_k pueden ser calculadas por partes, sin embargo la convergencia de este método es lenta, produciendo muchos más ciclos de CPU para llegar al resultado deseado. Es por ello que este método se recomienda sólo en la fase de aprendizaje del proceso de paralelización pero no en la fase de producción. Debido a que una lenta convergencia implica mayor número de mensajes, se espera que con este método se tenga el mayor deterioro de la eficiencia al aumentar el número de procesadores o al aumentar el grado de precisión en la solución.

En cambio con el método de Gauss-Seidel la paralelización no es fácil o intuitiva debido a la naturaleza del método, pero su convergencia es más rápida lo cual ofrece una ventaja. En este trabajo la forma en que se construye la matriz A es vital. Así, este método es recomendable cuando la matriz tiene muchos ceros, es simétrica y sus elementos distintos de cero se encuentran agrupados alrededor de la diagonal principal. Una ventaja adicional es que el tamaño de la matriz no importa en el proceso de paralelización. Se espera con este método tener un rendimiento simpre mayor que el método de Jacobi cuando el grado de precisión aumente. Hay que considerar que los dos métodos presentados aquí no son los únicos si no que hay una gran variedad de ellos, vease por ejemplo Barret et al (1994).

PVM trabaja bien en el ambiente de CICESE. Aparte de lo desarrollado en este

trabajo, también se ha implementado exitosamente un modelo del transporte de solutos en un acuífero (Gómez-Valdés et al, 1996). Sin embargo, estos trabajos han sido a nivel de prototipo. Para que PVM sea de utilidad a nivel de producción sería necesario conectar las computadoras usando otro tipo de topología y/o tecnología como por ejemplo, la topología anillo utilizando fibra óptica.

La computación paralela ha venido a resolver el problema de rapidez de CPU y ha originado un cambio en la mentalidad en la computación científica. Hoy en día ya no se justifica la adquisición de supercomputadoras con un sólo procesador, cuyo costo es de decenas de millones de dólares, pues con decenas de estaciones de trabajo, cuyo costo puede ser menor que un millón de dólares, se puede obtener la misma rapidez que con un sólo superprocesador, evitando además, entre otra cosas, los problemas de mantenimiento, calentamiento, etc. Sin embargo, hay que aprender a programar en paralelo, lo cual es un proceso que puede requerir de hacer el gasto en tiempo y esfuerzo. Debido a ello se recomienda la enseñanza de los elementos básicos de la computación paralela en las escuelas de computación del país.

7 CONCLUSIONES

La computación paralela es una tecnología a la vanguardia en la ciencia computacional, funciona muy bien en la solución de modelos matématicos, en particular, el método de elemento finito, el cual tiene una alta complejidad de programación, es factible de paralelizarse.

Con base en el trabajo de paralelización del problema del flujo transitorio en un acuífero se puede decir que el desempeño de PVM es bueno, aún y cuando la red local utilizada no sea la óptima para realizar cómputo paralelo.

Algunos algoritmos secuenciales no son fácilmente paralelizables, por lo que se requiere utilizar otros algoritmos o tomar en cuenta características particulares del problema, para lograr aprovechar el cómputo paralelo.

Al paralelizar un algoritmo secuencial, no es necesario que sea en su totalidad. Es recomendable paralelizar el bloque del algoritmo secuencial que consume mayor tiempo.

8 RECOMENDACIONES

1. Promover la enseñanza de la computacón paralela en las escuelas de computación.

2. Adquirir computadoras paralelas o clusters.

3. Usar en las redes topologias afines a la computación paralela para aprovechar el recurso.

9 REFERENCIAS

Barret R., M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eizkhout, R. Pozp, Ch. Romine y H. Vandrverst. 1994. Templates for the Solutions of Linear Systems: Building Blocks for Iterative Methods. SIAM, Philadelphia, ,112 pp.

Geist A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam . 1994. PVM: Parallel Virtual Machine, a users guide and tutorial for networked parallel computing. The MIT Press. Cambridge, MA. 279 pp.

Gómez-Valdes, J., J.A. Aranda-Alvarez y R. Soto Amador. 1996. PVM in Modeling Subsurface Solute Transport. Computers and Geosciences.

Kumar V., 1994. Introduction to Parallel Computing: Design and Analysis of Parallel algorithms. Benjamin/Cummings Pub. Co., Redwood City, Calif. 289 pp.

Pavlakos C.J, L.A. Schoor and J.F. Mareda. 1993. A visualization environment for supercomputing-based aplications in computational mechanics, En Proceedings of the Supercomputing '93 (Nov. 1993) 550-559 pp.

Ragsdale, S. 1991. Parallel Programming. McGraw-Hill, Inc.. 121 pp.

Sethian J.A. 1993. Computational fluid mechanics and massivelly parallel processors, proceedings of the Supercomputing '93. (Nov. 1993) 74-82 pp.

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

10 APÉNDICE A

Programas secuenciales en C utilzando los métodos de Gauss-Seidel y Jacobi.

```
/*************************************************************************/
/* Programa para el flujo transitorio utilizando elementos finitos y    */
/* el metodo de Gauss-Seidel para la solución del sistema de             */
/* ecuaciones                                                            */
/*                          Roberto Soto Amador          Enero/94        */
/*************************************************************************/
#include <stdio.h>
#include <math.h>

#define NNODE 1002
#define NELEM 500

double ns[4],nx[4],ny[4];
float hold[NNODE],hnew[NNODE],g[NNODE][NNODE],p[NNODE][NNODE],
      b[NNODE],x[NNODE],y[NNODE];
int node[4];
float s=0.002,t=0.02;
double xsi[4]={-0.57735,0.57735,0.57735,-0.57735},
      eta[4]={-0.57735,-0.57735,0.57735,0.57735};

main()
{

int nnode=NNODE,nelem=NELEM;
int i,j,m,n,jj,kk,l,k,iq;
int kount,kprint,nstep;
double time,dt,aa,bb,sum,err,oldval,amax;

/************************** Bloque 1 *********************************/
/* Generar coordenadas de los nodos, inicializar condiciones         */
/* iniciales y de frontera                                           */
/*********************************************************************/

        for(l=0; l<nnode; l+=2) {
                x[l]=(l)*5.0;
                x[l+1]=x[l];
                y[l]=10.0;
                y[l+1]=0.0;
        }

        for(l=0; l<nnode; l++) {
                hold[l]=hnew[l]=16.0;
                for(jj=0; jj<nnode;jj++)
                        g[l][jj]=p[l][jj]=0.0;
        }

        for(l=nnode-2; l<=nnode-1; l++)
                hold[l]=hnew[l]=11.0;

/************************  Bloque 2 *********************************/
/* Construcción y almacenamiento de matrices                         */
/*********************************************************************/
        for(k=1; k<=nelem; k++) {
                i=2*k-1;
                j=i+2;
                m=i+1;
                n=i-1;
                node[0]=i;
                node[1]=j;
                node[2]=m;
                node[3]=n;
                aa=fabs(x[j]-x[i])/2.0;
                bb=fabs(y[n]-y[i])/2.0;

                for(kk=0; kk<4; kk++) {
                        l=node[kk];
```

```
                        for(iq=0; iq<4; iq++) {
                                ns[0]=.25*(((double)1.0)-xsi[iq]) *
                                (((double)1.0)-eta[iq]);
                                ns[1]=.25*(((double)1.0)+xsi[iq]) *
                                (((double)1.0)-eta[iq]);
                                ns[2]=.25*(((double)1.0)+xsi[iq]) *
                                (((double)1.0)+eta[iq]);
                                ns[3]=.25*(((double)1.0)-xsi[iq]) *
                                (((double)1.0)+eta[iq]);

                                nx[0]=-.25*(((double)1.0)-eta[iq])/aa;
                                nx[1]=.25*(((double)1.0)-eta[iq])/aa;
                                nx[2]=.25*(((double)1.0)+eta[iq])/aa;
                                nx[3]=-.25*(((double)1.0)+eta[iq])/aa;

                                ny[0]=-.25*(((double)1.0)-xsi[iq])/bb;
                                ny[1]=-.25*(((double)1.0)+xsi[iq])/bb;
                                ny[2]=.25*(((double)1.0)+xsi[iq])/bb;
                                ny[3]=.25*(((double)1.0)-xsi[iq])/bb;

                                g[l][i] += (nx[0]*nx[kk]+ny[0]*ny[kk])*aa*bb;
                                g[l][j] += (nx[1]*nx[kk]+ny[1]*ny[kk])*aa*bb;
                                g[l][m] += (nx[2]*nx[kk]+ny[2]*ny[kk])*aa*bb;
                                g[l][n] += (nx[3]*nx[kk]+ny[3]*ny[kk])*aa*bb;

                                p[l][i] += ns[0]*ns[kk]*aa*bb*s/t;
                                p[l][j] += ns[1]*ns[kk]*aa*bb*s/t;
                                p[l][m] += ns[2]*ns[kk]*aa*bb*s/t;
                                p[l][n] += ns[3]*ns[kk]*aa*bb*s/t;
                        }
                }
        }

/************************   Bloque 3 ********************************/
/* Solución del sistema de acuaciones a través del tiempo ************/
/*******************************************************************/
        printf("                                HEAD");
        printf("                                 TIME\n");

        dt=5.0;
        kount=1;
        kprint=2;
        time=dt;
        for(nstep=1; nstep<=100; nstep++) {

                for(l=0; l<nnode; l++) {
                        b[l]=0.0;
                        for(jj=0; jj<nnode; jj++)
                                b[l]+=p[l][jj]*hold[jj]/dt;
                }
                do{
                        amax=0.0;
                        for(l=0; l<nnode; l++) {
                                if(l==0 || l==1 || l==nnode-2 || l==nnode-1)
                                        continue;
                                oldval=hnew[l];
                                sum=0;
                                for(jj=0; jj<nnode; jj++) {
                                        if(jj==l) continue;
                                        sum+=(g[l][jj]+p[l][jj]/dt)*hnew[jj];
                                }
                                hnew[l]=(-sum+b[l])/(g[l][l]+p[l][l]/dt);
                                err=fabs(oldval-hnew[l]);
                                if(err > amax)
                                        amax=err;
                        }
```

```
                } while(amax > 0.01);

                for(l=0; l<nnode; l++)
                        hold[l]=hnew[l];

                if(kount == kprint) {
                        for(i=0; i<nnode-1; i+=2)
                                printf("%5.2f ",hnew[i]);
                        printf("%5.2f\n",time);
                        kount=0;
                }
                time+=dt;
                kount++;
        }
}
```

```
/*************************************************************************/
/* Programa para el flujo transitorio utilizando elementos finitos y    */
/* el metodo de Jacobi para la solución del sistema de  ecuaciones       */
/*                                                                       */
/*                          Roberto Soto Amador            Marzo/95      */
/*************************************************************************/
#include <stdio.h>
#include <math.h>

#define NNODE 1002
#define NELEM 500

double ns[4],nx[4],ny[4];
float hold[NNODE],hnew[NNODE],g[NNODE][NNODE],p[NNODE][NNODE],
      b[NNODE],x[NNODE],y[NNODE];
int node[4];
float s=0.002,t=0.02;
double xsi[4]={-0.57735,0.57735,0.57735,-0.57735},
      eta[4]={-0.57735,-0.57735,0.57735,0.57735};

main()
{

int nnode=NNODE,nelem=NELEM;
int i,j,m,n,jj,kk,l,k,iq;
int kount,kprint,nstep;
double time,dt,aa,bb,sum,err,oldval,amax,dt1;

/************************** Bloque 1 *********************************/
/* Generar coordenadas de los nodos, inicializar condiciones         */
/* iniciales y de frontera                                           */
/*********************************************************************/
        for(l=0; l<nnode; l+=2) {
                x[l]=(l)*5.0;
                x[l+1]=x[l];
                y[l]=10.0;
                y[l+1]=0.0;
        }

        for(l=0; l<nnode; l++) {
                hold[l]=hnew[l]=16.0;
                for(jj=0; jj<nnode;jj++)
                        g[l][jj]=p[l][jj]=0.0;
        }

        for(l=nnode-2; l<=nnode-1; l++)
                hold[l]=hnew[l]=11.0;

/************************  Bloque 2 *********************************/
/* Construcción y almacenamiento de matrices                         */
/*********************************************************************/
        for(k=1; k<=nelem; k++) {
                i=2*k-1;
                j=i+2;
                m=i+1;
                n=i-1;
                node[0]=i;
                node[1]=j;
                node[2]=m;
                node[3]=n;
                aa=fabs(x[j]-x[i])/2.0;
                bb=fabs(y[n]-y[i])/2.0;

                for(kk=0; kk<4; kk++) {
                        l=node[kk];
                        for(iq=0; iq<4; iq++) {
```

BIBLIOTECA CENTRAL ENSENADA U.A.B.C.

```
                                ns[0]=.25*(((double)1.0)-xsi[iq]) *
                                (((double)1.0)-eta[iq]);
                                ns[1]=.25*(((double)1.0)+xsi[iq]) *
                                (((double)1.0)-eta[iq]);
                                ns[2]=.25*(((double)1.0)+xsi[iq]) *
                                (((double)1.0)+eta[iq]);
                                ns[3]=.25*(((double)1.0)-xsi[iq]) *
                                (((double)1.0)+eta[iq]);

                                nx[0]=-.25*(((double)1.0)-eta[iq])/aa;
                                nx[1]=.25*(((double)1.0)-eta[iq])/aa;
                                nx[2]=.25*(((double)1.0)+eta[iq])/aa;
                                nx[3]=-.25*(((double)1.0)+eta[iq])/aa;

                                ny[0]=-.25*(((double)1.0)-xsi[iq])/bb;
                                ny[1]=-.25*(((double)1.0)+xsi[iq])/bb;
                                ny[2]=.25*(((double)1.0)+xsi[iq])/bb;
                                ny[3]=.25*(((double)1.0)-xsi[iq])/bb;

                                g[l][i] += (nx[0]*nx[kk]+ny[0]*ny[kk])*aa*bb;
                                g[l][j] += (nx[1]*nx[kk]+ny[1]*ny[kk])*aa*bb;
                                g[l][m] += (nx[2]*nx[kk]+ny[2]*ny[kk])*aa*bb;
                                g[l][n] += (nx[3]*nx[kk]+ny[3]*ny[kk])*aa*bb;

                                p[l][i] += ns[0]*ns[kk]*aa*bb*s/t;
                                p[l][j] += ns[1]*ns[kk]*aa*bb*s/t;
                                p[l][m] += ns[2]*ns[kk]*aa*bb*s/t;
                                p[l][n] += ns[3]*ns[kk]*aa*bb*s/t;
                        }
                }
        }

/************************   Bloque 3 ********************************/
/* Solución del sistema de acuaciones a través del tiempo ************/
/*******************************************************************/
        printf("                                HEAD");
        printf("                                 TIME\n");

        dt=5.0;
        kount=1;
        kprint=2;
        time=dt;
        for(nstep=1; nstep<=100; nstep++) {

                for(l=0; l<nnode; l++) {
                        b[l]=0.0;
                        for(jj=0; jj<nnode; jj++)
                                b[l]+=p[l][jj]*hold[jj]/dt;
                }
                do{
                        amax=0.0;
                        for(l=0; l<nnode; l++) {
                                if(l==0 || l==1 || l==nnode-2 || l==nnode-1)
                                        continue;
                                oldval=hnew[l];
                                sum=0;
                                for(jj=0; jj<nnode; jj++) {
                                        if(jj==l) continue;
                                        sum+=(g[l][jj]+p[l][jj]/dt)*hold[jj];
                                }
                                hnew[l]=(-sum+b[l])/(g[l][l]+p[l][l]/dt);
                                err=fabs(oldval-hnew[l]);
                                if(err > amax)
                                        amax=err;
                        }
                        for(l=0; l<nnode; l++)
```

```
                        hold[l]=hnew[l];
                } while(amax > 0.01);


                if(kount == kprint) {
                        for(i=0; i<nnode-1; i+=2)
                                printf("%5.2f ",hnew[i]);
                        printf("%5.2f\n",time);
                        kount=0;
                }
                time+=dt;
                kount++;
        }


}
```

11 APÉNDICE B

Programas Paralelos (**Maestro-Esclavo**) utilizando el método de Gauss-Seidel

```
/*****************************************************************************/
/* Programa Maestro. Gauss Seidel                                            */
/*                                                                           */
/* Roberto Soto Amador                          4/Abril/95                   */
/*****************************************************************************/
#include <stdio.h>
#include "pvm3.h"
#include "ef.h"

main()
{
        float hnew[NNODE],g[NNODE][NNODE],p[NNODE][NNODE];
        int nnode,i,j,l,jj,kkk;
        float time,amax[NPROC];
        int mytid,nproc,tids[NPROC];

        nproc=NPROC;
        mytid=pvm_mytid();
        pvm_setopt(PvmRoute,PvmRouteDirect);
        /*************** Crea nproc tareas ***********/
        i=pvm_spawn("efe1",(char**)0,0,"",nproc,tids);
        pvm_initsend(PvmDataRaw);       /* Envío a todos los esclavos    */
        pvm_pkint(tids,nproc,1);        /* Arreglo de tareas,            */
        pvm_mcast(tids,nproc,0);

        printf("\n\n");
        printf("                              HEADING");
        printf("                                                  TIME\n");
        time=0.0;
        while(time<500){
                for(j=0; j<2; j++) {
                do {
                        for(i=0; i<nproc; i++) {
                                pvm_recv(tids[i],VerificaErr);
                                pvm_upkfloat(&amax[i],1,1);
                        }
                        for(i=0; i<nproc; i++)
                                if(amax[i]>0.01)
                                        break;
                        if(i==nproc)    /* Todo Bien */
                                i=0;
                        else
                                i=1;
                        pvm_initsend(PvmDataRaw);
                        pvm_pkint(&i,1,1);
                        pvm_mcast(tids,nproc,VerificaErr);
                }while(i);
                }

                for(i=0; i<nproc; i++) {
                        pvm_recv(-1,0);
                        pvm_upkint(&l,1,1);
                        pvm_upkint(&jj,1,1);
                        pvm_upkfloat(&hnew[l],jj,1);
                        pvm_upkfloat(&time,1,1);
                }
                for(i=0; i<NNODE-1; i+=2)
                        printf("%6.2f",hnew[i]);
                printf("  %6.2f\n",time);
        }
}
```

```
/*****************************************************************************/
/* Programa Esclavo. Gauss Seidel                                            */
/*                                                                           */
/* Roberto Soto Amador                          4/Abril   /94                */
/*****************************************************************************/
#include <stdio.h>
#include <math.h>
#include "pvm3.h"
#include "ef.h"

int   PosicionEsclavo();
#define AnchoBanda 4
#define AnchoBandaT (2*AnchoBanda-1)
#define maxncIzq      maxnc
#define minncIzq      minnc

main()
{
        int mytid,maestro,nproc,*tids;
        int nnode,i,j,jj,k,kk,iq,m,n,nn,l,ll,a1,b1,elemXproc,nresiduo,pos,ierr;
        int nodoini,nodofin,primernodo;
        int node[4],maxnc,maxncDer,minnc,minncDer;
        long int nelem,ndatos,kini,kfin,kprint,kount,ciclos,nstep;


        float aa,bb,time,dt,oldval,sum,err,dt1;
        double ns[4],nx[4],ny[4];
        float s=0.002,t=0.02;
        double xsi[4],eta[4];

        float *x,*y,**hold,*hnew,**g,**p,*b,*gp,*amax;
        char *actual;
        int **noceros;

        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;
        nproc=NPROC;
        nnode=NNODE;

        mytid=pvm_mytid();
        pvm_setopt(PvmRoute,PvmRouteDirect);
        maestro=pvm_parent();

        pvm_recv(maestro,0);        /* Recibe del maestro datos iniciales... */
        tids=(int *)malloc(nproc*sizeof(int));
        pvm_upkint(tids,nproc,1);   /* el arreglo de tareas,                 */

        pos=PosicionEsclavo(tids,nproc,mytid);/*Calcula posición de la tarea */
        /* esta posición es el nombre del procesador (procesador 0, etc)     */
        nelem = (nnode-2)/2;
        elemXproc=nelem/nproc;      /* Calcula # elementos que le corresponden*/
        nresiduo=nelem%nproc;       /* calcula si sobran elementos            */

        nodoini=1;                  /* Calcula el # del nodo inicial          */
        for(l=0;l<pos; l++)
                if(l<nresiduo)
                        nodoini+=(elemXproc+1)*2;
                else
                        nodoini+=elemXproc*2;

        if(nresiduo !=0 && pos<nresiduo)      /* Si sobran elementos y le     */
                elemXproc++;                  /* corresponde uno, agregalo    */

        nodofin=nodoini+elemXproc*2+1;        /* Calcula el # del nodo final  */
        ndatos=(nodofin-nodoini)+1;           /* Calcula el # de nodos        */
```

```
/***  Abre memoria según el número de datos que corresponde a este esclavo  ***/
        x=(float *)malloc(ndatos*sizeof(float));
        y=(float *)malloc(ndatos*sizeof(float));
        b=(float *)malloc(ndatos*sizeof(float));
        hold=(float **)malloc((nproc-pos)*sizeof(float *));
        hnew=(float *)malloc(nnode*sizeof(float));
        actual=(char *)malloc(nnode*sizeof(char));
        gp=(float *)malloc(nnode*sizeof(float));
        g=(float **)malloc(ndatos*sizeof(float *));
        p=(float **)malloc(ndatos*sizeof(float *));
        noceros=(int **)malloc(ndatos*sizeof(int *));
        amax=(float*)malloc((nproc-pos)*sizeof(float));

/********************** Si no hay memoria abortar **********************/
        if(x==NULL || y==NULL || b ==NULL || hold==NULL || hnew==NULL ||
        {       pvm_exit();
                exit(0);
        }

        for(l=0; l<ndatos; l++) {
                g[l]=(float *)malloc(nnode*sizeof(float));
                p[l]=(float *)malloc(nnode*sizeof(float));
                noceros[l]=(int *)malloc(2*sizeof(int));
                if(g[l]==NULL || p[l]==NULL || noceros[l]==NULL) {
                        pvm_exit();
                        exit(0);
                }
        }
        for(l=0; l<nproc-pos; l++) {
                hold[l]=(float *)malloc(nnode*sizeof(float));
                if(hold[l]==NULL) {
                        pvm_exit();
                        exit(0);
                }
        }

/***************************** Bloque 1 ******************************/
/********************** Inicialización de datos **********************/
        for(l=nodoini; l<=nodofin; l+=2) {
                i=l-nodoini;
                x[i]=x[i+1]=(l-1)*5;

                y[i]=10;
                y[i+1]=0;

                for(j=0; j<nnode; j++) {
                        g[i][j]=g[i+1][j]=p[i][j]=p[i+1][j]=0;
                        hnew[j]=hold[nproc-pos-1][j]=hold[0][j]=16.0;
                        if(j==NNODE-1 || j== NNODE-2)
                                hnew[j]=hold[nproc-pos-1][j]=hold[0][j]=11.0;

                }

        }

/*******************************  BlOQUE 2 ***********************************/
/************************** Generar matrices G y P ***************************/
        kini=nodoini/2+1;       /* Calcula el elemento inicial */
        kfin=kini+elemXproc;    /* Calcula el elemento final   */

        for(k=kini,nn=1; k<kfin; k++,nn++) { /* para cada uno de los elementos*/
                i=2*k-1;                     /* calcula los indices para las  */
                j=i+2;                       /* columnas de las matrices      */
                m=i+1;
                n=i-1;
                node[0]=2*nn-1;              /* calcula el indice para los    */
```

```
                node[1]=node[0]+2;          /* renglones de las matrices     */
                node[2]=node[0]+1;
                node[3]=node[0]-1;

                aa=fabs(x[node[1]]-x[node[0]])/2.0;
                bb=fabs(y[node[3]]-y[node[0]])/2.0;

                for(kk=0; kk<4; kk++) {
                        l=node[kk];
                        for(iq=0; iq<4; iq++) {
                                ns[0]=.25*(((double)1.0)-xsi[iq]) *
                                (((double)1.0)-eta[iq]);
                                ns[1]=.25*(((double)1.0)+xsi[iq]) *
                                (((double)1.0)-eta[iq]);
                                ns[2]=.25*(((double)1.0)+xsi[iq]) *
                                (((double)1.0)+eta[iq]);
                                ns[3]=.25*(((double)1.0)-xsi[iq]) *
                                (((double)1.0)+eta[iq]);

                                nx[0]=-.25*(((double)1.0)-eta[iq])/aa;
                                nx[1]=.25*(((double)1.0)-eta[iq])/aa;
                                nx[2]=.25*(((double)1.0)+eta[iq])/aa;
                                nx[3]=-.25*(((double)1.0)+eta[iq])/aa;

                                ny[0]=-.25*(((double)1.0)-xsi[iq])/bb;
                                ny[1]=-.25*(((double)1.0)+xsi[iq])/bb;
                                ny[2]=.25*(((double)1.0)+xsi[iq])/bb;
                                ny[3]=.25*(((double)1.0)-xsi[iq])/bb;

                                g[l][i] += (nx[0]*nx[kk]+ny[0]*ny[kk])*aa*bb;
                                g[l][j] += (nx[1]*nx[kk]+ny[1]*ny[kk])*aa*bb;
                                g[l][m] += (nx[2]*nx[kk]+ny[2]*ny[kk])*aa*bb;
                                g[l][n] += (nx[3]*nx[kk]+ny[3]*ny[kk])*aa*bb;

                                p[l][i] += ns[0]*ns[kk]*aa*bb*s/t;
                                p[l][j] += ns[1]*ns[kk]*aa*bb*s/t;
                                p[l][m] += ns[2]*ns[kk]*aa*bb*s/t;
                                p[l][n] += ns[3]*ns[kk]*aa*bb*s/t;
                        }
                }
        }


/*************************************************************************/
/*  Para completar los valores reales de las matrices P y G se envían los    */
/* renglones que comparten los procesadores. El procesador 0 comparte sus    */
/* últimos dos renglones con el procesador 1 (los dos primeros de 1) que a   */
/* su vez comparte sus dos últimos con el procesador 2, etc.                 */
/* Los procesadores que comparten sus renglones iniciales con otro procesador*/
/* son los que efectúan las operaciones y mandan los resultados al procesador*/
/* con quien comparten.                                                      */
/*************************************************************************/

        switch(pos) {
                case 0:
                        /************ Mando a 1 **************/
                        pvm_initsend(PvmDataRaw);
                        for(l=2; l>0; l--) {
                                pvm_pkfloat(g[ndatos-l],nnode,1);
                                pvm_pkfloat(p[ndatos-l],nnode,1);
                        }
                        pvm_send(tids[1],21);
                        /************ Recibe de 1 ************/
                        pvm_recv(tids[1],20);
                        for(l=2; l>0; l--) {
                                pvm_upkfloat(gp,nnode,1);
```

```
                        for(i=0; i<nnode; i++)
                                g[ndatos-1][i]=gp[i];

                        pvm_upkfloat(gp,nnode,1);
                        for(i=0; i<nnode; i++)
                                p[ndatos-1][i]=gp[i];
                }
                break;

        default:
        if(pos == nproc-1) {
                /************ Recibo de penúltimo *******/
                pvm_recv(tids[pos-1],20+nproc-1);
                for(l=0; l<2; l++) {
                        pvm_upkfloat(gp,nnode,1);
                        for(i=0;i<nnode;i++)
                                g[l][i]+=gp[i];

                        pvm_upkfloat(gp,nnode,1);
                        for(i=0;i<nnode;i++)
                                p[l][i]+=gp[i];
                }
                /************ Envio a penúltimo ********/
                pvm_initsend(PvmDataRaw);
                for(l=0; l<2; l++) {
                        pvm_pkfloat(g[l],nnode,1);
                        pvm_pkfloat(p[l],nnode,1);
                }
                pvm_send(tids[pos-1],20+pos-1);
        }else{
                /********** Mando pos+1 ***************/
                pvm_initsend(PvmDataRaw);
                for(l=2; l>0; l--) {
                        pvm_pkfloat(g[ndatos-l],nnode,1);
                        pvm_pkfloat(p[ndatos-l],nnode,1);
                }
                pvm_send(tids[pos+1],(20+pos+1));
                /*********** Recibo de pos-1 ***********/
                pvm_recv(tids[pos-1],(20+pos));
                for(l=0; l<2; l++) {
                        pvm_upkfloat(gp,nnode,1);
                        for(i=0;i<nnode;i++)
                                g[l][i]+=gp[i];

                        pvm_upkfloat(gp,nnode,1);
                        for(i=0;i<nnode;i++)
                                p[l][i]+=gp[i];
                }
                /********** Mando a pos-1 **************/
                pvm_initsend(PvmDataRaw);
                for(l=0; l<2; l++) {
                        pvm_pkfloat(g[l],nnode,1);
                        pvm_pkfloat(p[l],nnode,1);
                }
                pvm_send(tids[pos-1],(20+pos-1));

                /*********** Recibo de pos+1 ************/
                pvm_recv(tids[pos+1],(20+pos));
                for(l=2; l>0; l--) {
                        pvm_upkfloat(g[ndatos-l],nnode,1);
                        pvm_upkfloat(p[ndatos-l],nnode,1);
                }
                /*break;*/
        }/*fin de else*/
        break; /*de default*/
}
```

```
/***************************** BLOQUE 3 ************************************/
/********** Solución del sistema de ecuaciones a través del tiempo  *********/

        dt=5.0;
        kount=1;
        kprint=2;
        time=dt;
        if(pos==0) primernodo=0;
        else       primernodo=2;

        for(l=primernodo; l<ndatos; l++) {
                ll=l+nodoini-1;
                noceros[l][0]=ll-AnchoBanda+1;
                noceros[l][1]=ll+AnchoBanda-1;
                if(noceros[l][0] < 0)
                        noceros[l][0]=0;

                if(noceros[l][1] >= nnode)
                        noceros[l][1]=nnode-1;
        }
        minnc=noceros[primernodo][0];
        maxnc=noceros[ndatos-1][1];

        switch(pos) {
        case 0:
                pvm_initsend(PvmDataRaw);
                pvm_pkint(&minnc,1,1);
                pvm_pkint(&maxnc,1,1);
                pvm_send(tids[1],1);
                pvm_recv(tids[pos+1],0);
                pvm_upkint(&minnc,1,1);
                pvm_upkint(&maxnc,1,1);
                break;
        case NPROC-1:
                pvm_initsend(PvmDataRaw);
                pvm_pkint(&minnc,1,1);
                pvm_pkint(&maxnc,1,1);
                pvm_send(tids[pos-1],pos-1);
                pvm_recv(tids[pos-1],nproc-1);
                pvm_upkint(&minnc,1,1);
                pvm_upkint(&maxnc,1,1);
                break;
        default:
                pvm_initsend(PvmDataRaw);
                pvm_pkint(&minnc,1,1);
                pvm_pkint(&maxnc,1,1);
                pvm_send(tids[pos+1],pos+1);
                pvm_initsend(PvmDataRaw);
                pvm_pkint(&minnc,1,1);
                pvm_pkint(&maxnc,1,1);
                pvm_send(tids[pos-1],pos-1);

                pvm_recv(tids[pos-1],pos);
                pvm_upkint(&minncIzq,1,1);
                pvm_upkint(&maxncIzq,1,1);
                pvm_recv(tids[pos+1],pos);
                pvm_upkint(&minncDer,1,1);
                pvm_upkint(&maxncDer,1,1);
        }


        for(nstep=0; nstep<100; nstep++) { /* repite 100 veces            */
        /***************** Inicializar Actual *****************/
        if(!pos)
                for(l=0; l<nnode; l++)
```

```
                actual[l]=1;
        else
                for(l=0; l<nnode; l++){
                        if(l < nodoini+1) actual[l]=0;
                        else              actual[l]=1;
                }

                /****************** Inicializa el vector b ******************/
                for(l=0; l<ndatos; l++) {
                        b[l]=0.0;
                        for(jj=0; jj<nnode; jj++) {
                                b[l]+=p[l][jj]*hold[nproc-pos-1][jj]/dt;
                        }
                }

                ciclos=1;
                do {
                        /* recorres los errores  */
                        for(l=0;l<nproc-pos-1;l++) {
                                amax[l]=amax[l+1];
                        }
                        amax[nproc-pos-1]=0.0;
                        for(l=primernodo; l<ndatos; l++) {
                                ll=l+nodoini-1;
                                if(l==primernodo || ndatos-1 <=3) {
                                for(jj=noceros[l][0]; jj<=noceros[l][1]; jj++) {
                                        if(!actual[jj]) {
                                        switch(pos) {
                                        case 0:
                                                pvm_recv(tids[1],jj);
                                                pvm_upkfloat(&hnew[jj],1,1);
                                                actual[jj]=1;
                                        break;
                                        case NPROC-1:
                                                pvm_recv(tids[pos-1],jj);
                                                pvm_upkfloat(&hnew[jj],1,1);
                                                actual[jj]=1;
                                        break;
                                        default:
                                                pvm_recv(-1,jj);
                                                pvm_upkfloat(&hnew[jj],1,1);
                                                actual[jj]=1;
                                        }/*switch*/
                                        }/*if actual */
                                }/*for jj*/
                                }/*if*/

                                oldval=hnew[ll];

                                if(!(ll==0||ll==1||ll==NNODE-2||ll==NNODE-1)) {
                                sum=0;
                                for(jj=0; jj<NNODE; jj++) {
                                        if(jj==ll) continue;
                                        sum+=(g[l][jj]+p[l][jj]/dt)*hnew[jj];
                                }
                                hnew[ll]=(-sum+b[l])/(g[l][ll]+p[l][ll]/dt);
                                }/* if (!) */

                                switch(pos) {
                                case 0: if(minnc <= ll && ll <= maxnc) {
                                                pvm_initsend(PvmDataRaw);
                                                pvm_pkfloat(&hnew[ll],1,1);
                                                pvm_send(tids[1],ll);
                                        }
                                        break;
                                 case NPROC-1:
```

```
                        if(minnc <= ll && ll <= maxnc) {
                                pvm_initsend(PvmDataRaw);
                                pvm_pkfloat(&hnew[ll],1,1);
                                pvm_send(tids[pos-1],ll);
                        }
                        break;
                default:
                        if(minncDer <= ll && ll <= maxncDer) {
                                pvm_initsend(PvmDataRaw);
                                pvm_pkfloat(&hnew[ll],1,1);
                                pvm_send(tids[pos+1],ll);
                        }
                        if(minncIzq <= ll && ll <= maxncIzq) {
                                pvm_initsend(PvmDataRaw);
                                pvm_pkfloat(&hnew[ll],1,1);
                                pvm_send(tids[pos-1],ll);
                        }
                }/*switch*/


                err=fabs(oldval-hnew[ll]);
                if(err > amax[nproc-pos-1])
                        amax[nproc-pos-1]=err;
        }

        /* recorre  HOLD */
        for(l=0;l<nproc-pos-1;l++) {
                for(i=0; i<NNODE; i++)
                        hold[l][i]=hold[l+1][i];
        }
        for(i=0; i<NNODE; i++)
                hold[nproc-pos-1][i]=hnew[i];

        /*********** Actualización de actual ***************/

        if(pos!=NPROC-1)
                for(jj=nodofin; jj<nodofin+3; jj++)
                        actual[jj]=0;
        if(pos)
                for(jj=nodoini-2; jj<nodoini+1; jj++)
                        actual[jj]=0;

        if(ciclos >= (nproc-pos)) {
                pvm_initsend(PvmDataRaw);
                pvm_pkfloat(&amax[0],1,1);
                pvm_send(maestro,VerificaErr);
                pvm_recv(maestro,VerificaErr);
                pvm_upkint(&jj,1,1);
        }
        ciclos++;
}while(jj);

/** recupera datos enviados **/
if(pos) {
        for(jj=nodoini-2; jj<nodoini+1; jj++){
                pvm_recv(tids[pos-1],jj);
                pvm_upkfloat(&hnew[jj],1,1);
                actual[jj]=1;
        }
}
/** Envía los resultados para completar una solución **/
if(pos) {
        pvm_initsend(PvmDataRaw);
        pvm_pkfloat(&hold[0][primernodo+nodoini-1],3,1);
        pvm_send(tids[pos-1],CompletaHN);
}
```

```
                /** Recibe los resultados para completar una solución **/
                if(pos!=NPROC-1) {
                        pvm_recv(tids[pos+1],CompletaHN);
                        pvm_upkfloat(&hold[0][nodofin],3,1);
                }

                /** Colocarlos en nivel mas alto para la siguiente solución **/
                /** hnew con valores de la solución                          **/
                for(i=0; i<NNODE; i++)
                        hnew[i]=hold[nproc-pos-1][i]=hold[0][i];

                        /***** Envía resultados al Maestro ******/
                if(kount == kprint) {
                        i=nodoini-1+primernodo;
                        l=ndatos-primernodo;
                        pvm_initsend(PvmDataRaw);
                        pvm_pkint(&i,1,1);
                        pvm_pkint(&l,1,1);
                        pvm_pkfloat(&hold[0][i],1,1);
                        pvm_pkfloat(&time,1,1);
                        pvm_send(maestro,0);
                        kount=0;
                }
                time+=dt;
                kount++;
        } /* Fin del FOR 100 veces  */

        pvm_exit(); /* salir de pvm */
}


/* Función que regresa la posición que ocupa en el arreglo de tareas una tarea*/
/* específica, el arreglo de tareas y la tarea entran como parámetros         */
int PosicionEsclavo(tids,nproc,mytid)
        int *tids,nproc,mytid;
{
        int i;
        for(i=0; i<nproc; i++)
                if(tids[i]==mytid)
                        return i;

        return -1;
}
```

12 APÉNDICE C

Programas Paralelos (**Maestro-Esclavo**) utilizando el método de Jacobi

```
/*****************************************************************************/
/* Programa Maestro. de efinito.c paralelo con Jacobi                        */
/*                                                                           */
/* Roberto Soto Amador                    12/Marzo/95                        */
/*****************************************************************************/
#include <stdio.h>
#include "pvm3.h"

#define NPROC 3
main()
{
        float *hnew;
        int nnode,i,j;
        float time;
        int mytid,nproc,tids[NPROC];

/********************** Inicialización de datos ****************************/
        nnode=22;
        nproc=NPROC;

        hnew=(float *)malloc(nnode*sizeof(float));
        mytid=pvm_mytid();
        pvm_setopt(PvmRoute,PvmRouteDirect);
        /*************** Crea NPROC tareas ***********/
        i=pvm_spawn("esclavoB3j",(char**)0,0,"",nproc,tids);

        pvm_initsend(PvmDataRaw);       /* Envío a todos los esclavos    */
        pvm_pkint(&nproc,1,1);
        pvm_pkint(tids,nproc,1);        /* Arreglo de tareas,            */
        pvm_pkint(&nnode,1,1);          /* # de nodos,                   */
        pvm_mcast(tids,nproc,0);

        printf("                              HEADING");
        printf("                                                TIME\n");
        time=0.0;

        while(time<500){
                pvm_recv(-1,10);    /* Recibe del último procesador      */
                pvm_upkfloat(hnew,nnode,1);    /* vector HNEW y TIME     */
                pvm_upkfloat(&time,1,1);
                for(i=0; i<nnode-1; i+=2) /*21*/
                        printf("%6.2f ",hnew[i]);
                printf("%6.2f\n",time);
        }
}
```

```
/*****************************************************************************/
/* Programa Esclavo. de efinito.c paralelo con Jacobi                        */
/*                                                                           */
/* Roberto Soto Amador                    12/Marzo/95                        */
/*****************************************************************************/
#include <stdio.h>
#include <math.h>
#include "pvm3.h"

extern void PvmMin();
extern void PvmMax();
extern void PvmSum();
extern void PvmProduct();

int   PosiciónEsclavo();
#define NPROC nproc
main()
{

        int mytid,maestro,nproc,raiz,*tids;
        int nnode,i,j,jj,k,kk,iq,m,n,nn,l,ll,elemXproc,nresiduo,pos;
        int nodoini,nodofin,primernodo;
        int node[4];
        long int nelem,ndatos,kini,kfin,kprint,kount,ciclos,nstep;

        float aa,bb,time,dt,amax,oldval,sum,err;
        double ns[4],nx[4],ny[4];
        float s=0.002,t=0.02;
        double xsi[4],eta[4];

        float *x,*y,*hold,*hnew,**g,**p,*b,*gp;
        char *GrupoE="esclavos";

        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;

        mytid=pvm_mytid();
        pvm_setopt(PvmRoute,PvmRouteDirect);
        maestro=pvm_parent();

        pvm_recv(maestro,0);          /* Recibe del maestro datos iniciales... */
        pvm_upkint(&nproc,1,1);
        tids=(int *)malloc(nproc*sizeof(int));
        pvm_upkint(tids,nproc,1);     /* el arreglo de tareas,                 */
        pvm_upkint(&nnode,1,1);       /* número de nodos totales,              */

        pos=PosiciónEsclavo(tids,nproc,mytid);/*Calcula posición de la tarea */
        /* esta posición es el nombre del procesador (procesador 0, etc)     */

        nelem = (nnode-2)/2;
        elemXproc=nelem/nproc;        /* Calcula # elementos que le corresponden*/
        nresiduo=nelem%nproc;         /* calcula si sobran elementos           */

        nodoini=1;                    /* Calcula el # del nodo inicial         */
        for(l=0;l<pos; l++)
                if(l<nresiduo)
                        nodoini+=(elemXproc+1)*2;
                else
                        nodoini+=elemXproc*2;

        if(nresiduo !=0 && pos<nresiduo)      /* Si sobran elementos y le    */
                elemXproc++;                  /* corresponde uno, agrégalo   */

        nodofin=nodoini+elemXproc*2+1;        /* Calcula el # del nodo final */
        ndatos=(nodofin-nodoini)+1;           /* Calcula el # de nodos       */
```

```
                if(pos == nproc-1) {
                        /************ Recibo de penúltimo *******/
                        pvm_recv(tids[pos-1],20+nproc-1);
                        for(l=0; l<2; l++) {
                                pvm_upkfloat(gp,nnode,1);
                                for(i=0;i<nnode;i++)
                                        g[l][i]+=gp[i];

                                pvm_upkfloat(gp,nnode,1);
                                for(i=0;i<nnode;i++)
                                        p[l][i]+=gp[i];
                        }
                        /************ Envio a penúltimo ********/
                        pvm_initsend(PvmDataRaw);
                        for(l=0; l<2; l++) {
                                pvm_pkfloat(g[l],nnode,1);
                                pvm_pkfloat(p[l],nnode,1);
                        }
                        pvm_send(tids[pos-1],20+pos-1);
                }else{
                        /********** Mando pos+1 ****************/
                        pvm_initsend(PvmDataRaw);
                        for(l=2; l>0; l--) {
                                pvm_pkfloat(g[ndatos-l],nnode,1);
                                pvm_pkfloat(p[ndatos-l],nnode,1);
                        }
                        pvm_send(tids[pos+1],(20+pos+1));
                        /*********** Recibo de pos-1 ***********/
                        pvm_recv(tids[pos-1],(20+pos));
                        for(l=0; l<2; l++) {
                                pvm_upkfloat(gp,nnode,1);
                                for(i=0;i<nnode;i++)
                                        g[l][i]+=gp[i];

                                pvm_upkfloat(gp,nnode,1);
                                for(i=0;i<nnode;i++)
                                        p[l][i]+=gp[i];
                        }
                        /********** Mando a pos-1 **************/
                        pvm_initsend(PvmDataRaw);
                        for(l=0; l<2; l++) {
                                pvm_pkfloat(g[l],nnode,1);
                                pvm_pkfloat(p[l],nnode,1);
                        }
                        pvm_send(tids[pos-1],(20+pos-1));

                        /*********** Recibo de pos+1 ************/
                        pvm_recv(tids[pos+1],(20+pos));
                        for(l=2; l>0; l--) {
                                pvm_upkfloat(g[ndatos-l],nnode,1);
                                pvm_upkfloat(p[ndatos-l],nnode,1);
                        }
                }/*fin de else*/
                break; /*de default*/
        }

/****************************** BLOQUE 3 ************************************/
/************ Solución del sistema de ecuaciones a través del tiempo ********/

        raiz=pvm_joingroup(GrupoE);
        pvm_barrier(GrupoE,nproc);

        dt=5.0;
        kount=1;
        kprint=2;
        time=dt;
```

```
/* El primer proceso calcula los nodos límites de izquierda y derecha */
/* y los otros procesos no calculan los nodos límites de su izquierda */
if(pos==0) primernodo=0;
else       primernodo=2;

ciclos=1;
for(nstep=0; nstep<100; nstep++) { /* repite 100 veces            */

        /****************** Inicializa el vector b ******************/
        for(l=0; l<ndatos; l++) {
                b[l]=0.0;
                for(jj=0; jj<nnode; jj++)
                        b[l]+=p[l][jj]*hold[jj]/dt;
        }

        do {
                amax=0.0;
                /* Ceros a los datos que no afecta el proceso */
                for(l=0; l< nnode; l++)
                        if(l<(nodoini-1+primernodo) || l>=nodofin)
                                hnew[l]=0.0;

                for(l=primernodo; l<ndatos; l++) {
                        ll=l+nodoini-1;
                        if(ll==0 || ll==1 || ll==nnode-1 || ll==nnode-2)
                                continue;
                        oldval=hnew[ll];
                        sum=0;
                        for(jj=0; jj<nnode; jj++) {
                                if(jj==ll) continue;
                                sum+=(g[l][jj]+p[l][jj]/dt)*hold[jj];
                        }
                        hnew[ll]=(-sum+b[l])/(g[l][ll]+p[l][ll]/dt);

                        err=fabs(oldval-hnew[ll]);
                        if(err > amax)
                                amax=err;
                }
                /* espera que todos tengan los datos calculados*/

                pvm_barrier(GrupoE,nproc);
                l=pvm_reduce(PvmSum,hnew,nnode,PVM_FLOAT,6,GrupoE,0);
                l=pvm_reduce(PvmMax,&amax,1,PVM_FLOAT,7,GrupoE,0);
                pvm_barrier(GrupoE,nproc);

                if(raiz==0) {
                        pvm_initsend(PvmDataRaw);
                        pvm_pkfloat(hnew,nnode,1);
                        pvm_pkfloat(&amax,1,1);
                        pvm_bcast(GrupoE,5);
                }else {
                        pvm_recv(-1,5);
                        pvm_upkfloat(hnew,nnode,1);
                        pvm_upkfloat(&amax,1,1);
                }
                if(ciclos)
                        ciclos=0;
                for(l=0; l< nnode; l++)
                        hold[l]=hnew[l];

        }while(amax > 0.01);

        if(raiz==0) {              /* Si es el último procesador */
                if(kount == kprint) {  /* y se imprime           */
                        /* Envia HNEW y TIME al nodo maestro     */
```

```
                                pvm_initsend(PvmDataRaw);
                                pvm_pkfloat(hnew,nnode,1);
                                pvm_pkfloat(&time,1,1);
                                pvm_send(maestro,10);
                                kount=0;
                        }
                }
                time+=dt;
                kount++;
        } /* Fin del FOR 100 veces  */

        pvm_exit(); /* salir de pvm */
}


/* Función que regresa la posición que ocupa en el arreglo de tareas una tarea*/
/* específica, el arreglo de tareas y la tarea entran como parémetros         */
int PosiciónEsclavo(tids,nproc,mytid)
        int *tids,nproc,mytid;
{
        int i;
        for(i=0; i<nproc; i++)
                if(tids[i]==mytid)
                        return i;

        return -1;
}
```