UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"Paralelización del algoritmo de Schwarz usando MPI para la resolución de sistemas lineales de gran escala"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN
PRESENTA:

CLEMENTE GONZÁLEZ JULIO CÉSAR

DIRECTOR DE TESIS:
ING. JUAN JOSÉ CARREÓN GRANADOS

MÉXICO, D. F., CD. UNIVERSITARIA, NOVIEMBRE DE 2008

Agradecimientos

Al grupo de Simulación Numérica de Yacimientos y a la Facultad de Ingeniería de la UNAM por todas las facilidades otorgadas para la realización del presente trabajo.

Índice de contenido

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"Paralelización del algoritmo de Schwarz usando MPI para la resolución de sistemas lineales de gran escala"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN
PRESENTA:

CLEMENTE GONZÁLEZ JULIO CÉSAR

DIRECTOR DE TESIS:
ING. JUAN JOSÉ CARREÓN GRANADOS

MÉXICO, D. F., CD. UNIVERSITARIA, NOVIEMBRE DE 2008

Agradecimientos

Al grupo de Simulación Numérica de Yacimientos y a la Facultad de Ingeniería de la UNAM por todas las facilidades otorgadas para la realización del presente trabajo.

Índice de contenido

INTRODUCCIÓN

En la industria es necesario disponer de herramientas de simulación, que permitan representar en un tiempo corto, procesos de producción. Ésto es con el propósito de generar un marco de referencia para la toma de decisiones, o para evaluar el impacto del uso de algún método productivo, por mencionar algunos.

Por ejemplo, en la ingeniería petrolera en el área de producción, un simulador de yacimientos es una herramienta muy útil, porque permite generar y probar esquemas de explotación de yacimientos antes de ponerlos en práctica.

La programación de estos simuladores consta de muchas partes, y en ellos existe dentro un motor numérico que se emplea para generar soluciones a sistemas de ecuaciones diferenciales parciales. Ecuaciones que buscan representar lo mejor posible los muchos fenómenos que se encuentran interrelacionados en el área de interés, que en este caso es el de yacimientos de petróleo.

Una de las partes cruciales es la encargada de la solución de los sistemas lineales, que aparecen posterior a la discretización de las ecuaciones diferenciales parciales.

Como es un proceso iterativo para simular un instante de tiempo, se generan varios sistemas lineales a resolver para un paso de tiempo. Como la simulación involucra varios pasos de tiempo, es necesario resolver muchos sistemas lineales durante todo el proceso de una simulación.

Se estima que más del 60 % del tiempo de simulación se consume en la solución de los muchos sistemas lineales que se generan, por ello es muy importante encontrar una metodología que permita resolver este problema en el menor tiempo posible.

Los simuladores que se emplean en la ingeniería petrolera como consumen muchos recursos computacionales, el yacimiento que pueden simular será tan grande como lo permita la computadora en la que se ejecute. En este contexto también es viable usar la paralelización no solo para acelerar el proceso, sino para poder simular problemas de yacimientos más grandes.

Para resolver el sistema lineal se plantea el uso de una teoría denominada descomposición de dominios de *Schwarz*, que básicamente establece la posibilidad de resolver un problema complejo perteneciente a un dominio, a partir de dividirlo en subdominios de menor complejidad y reuniendo las partes para conformar la solución final.

Está teoría es aplicada a nuestra matriz de coeficientes, recordando que un sistema de ecuaciones lineales se puede representar con matrices como:

$$Ax = b$$

Donde:

- *A* es la matriz de coeficientes
- *x* es el vector de incógnitas
- *b* es el vector de términos independientes.

La matriz *A* es dispersa, diagonal dominante y muy grande (desde 70,000 variables), por lo que resolverla mediante un método directo y con una representación densa, implica un costo computacional muy alto. Es por estas razones que se utiliza un método iterativo y una representación compacta para la matriz de coeficientes.

Como se mencionó, el tamaño de la matriz es muy grande, y es por ello que se buscó alguna forma de dividirla en porciones menores para generar problemas más simples y menos costosos, siendo esta forma, la implementación de la descomposición de dominios antes mencionada. Esto además permitió la posibilidad de paralelizarlo en una computadora del tipo MIMD[1] (un *cluster* tipo *Beowulf*).

Ahora bien, un *solver* es un medio empleado para obtener la solución de un sistema de ecuaciones lineales. Se le dan como argumentos un sistema, una primera aproximación y el vector de términos independientes y se espera obtener la solución del sistema.

En este caso, el *solver* es la implementación de un algoritmo de un método de solución iterativo, lográndose crear una función que recibe el sistema de ecuaciones lineales y devuelve una solución del mismo. El resultado es un *solver* paralelo escrito en Fortran 90 que permite manejar con mayor rapidez sistemas lineales muy grandes.

El presente trabajo pretende mostrar un método computacional para solucionar un tipo particular de sistemas lineales que se pueden representar como matrices ralas de gran escala.

1 MIMD es un modelo de arquitectura de computadora paralela que significa Multiple Instrucction Multiple Data

Capítulo 1 Arquitectura de computadoras paralelas

El cómputo paralelo consiste en la ejecución de más de una instrucción o cálculo al mismo tiempo, en al menos dos o más procesadores, los cuales pueden estar ubicados en la misma máquina o distribuidos en varias de ellas.

Las razones para utilizar computadoras paralelas es el de reducir el tiempo total de realización de una tarea (tiempo total de cómputo), al distribuir la carga de trabajo entre los diferentes procesadores disponibles; así como el poder atacar problemas mucho más grandes que los posibles de resolver en una única computadora.

Para lograrlo, se debe considerar el proceso de paralelización como un factor base y así obtener un alto desempeño en la máquina paralela sobre la que se trabaja. Considerar además el hecho de que un *cluster* tipo *Beowulf* es un tipo de máquina paralela.

1.1 Tipos de arquitectura de computadoras

Debido al crecimiento de la tecnología en el campo del cómputo paralelo, existen diferencias entre las máquinas, su clasificación según Flynn es:

- **Single Instruction Single Data (SISD)**

  Este es el esquema tradicional de una computadora secuencial, solo se puede efectuar una instrucción por vez sobre un dato. Por lo que para ejecutar un programa completo, se hace instrucción a instrucción.

Este tipo de computadoras fueron las más comunes a finales del 2006, cuando se dio la

transición hacia los procesadores multinúcleo, que comenzaron a aparecer a principios de 2005.

- **Single Instruction Multiple Data (SIMD)**

  Este esquema corresponde ya a una computadora paralela, en donde existe al menos dos unidades de procesamiento y un método común para compartir el programa a ejecutar, aunque los datos pueden estar o no compartidos.

Una característica que deben presentar los datos para trabajar con estas computadoras es que la interdependencia entre ellos no debe ser muy estrecha, lo que permite realizar operaciones sobre ellos simultáneamente, de forma que los resultados entre pasos simples se puedan efectuar, por ser independientes entre ellos.

Básicamente lo que se distribuyen son los datos, pero el procesamiento es el mismo para cada grupo, por lo tanto el programa utilizado es el mismo que se aplica sobre los diferentes datos.

En este tipo de esquema, la memoria puede ser o no compartida,

- **Multiple Instruction Single Data (MISD)**

  Se pueden realizar varias operaciones sobre un mismo conjunto de datos al mismo tiempo. Se considera que al mismo conjunto de datos se les realiza una serie de procesamientos diferentes e independientes, lo que permite distribuir las diferentes instrucciones en cada unidad de procesamiento, empelando los mismos datos.

- **Multiple Instruction Multiple Data (MIMD)**

  Máquinas que poseen varias unidades de procesamiento (como pueden ser varios

procesadores), en las cuales se realiza múltiples instrucciones sobre datos diferentes de forma simultánea. Cada unidad de procesamiento tiene su propia copia del código que puede ejecutar a diferente velocidad e incluso diferentes fragmentos del mismo.

Un *cluster* tipo *Beowulf* pertenece a este último esquema, porque a pesar de ser una máquina con memoria distribuida, es posible repartir diferentes instrucciones a realizarse sobre diferentes grupos de datos simultáneamente.

1.2 Acoplamiento

Se entiende por acoplamiento al grado de integración que tenga todos los elementos que existen en la máquina paralela. Dependiendo cuan integrados estén los componentes que la conforman, pueden estar más acoplada o menos acoplada.

Gran parte de la integración del sistema depende directamente de la comunicación entre las unidades de procesamiento, y es uno de los puntos más importantes utilizado para definir el acoplamiento; distinguiendo tres casos.

- **Acoplamiento fuerte**

  En este tipo de máquinas paralelas, sus elementos están altamente interrelacionados y se realizan las actividades de cómputo de manera totalmente distribuida. El caso de mayor acoplamiento está dado por una computadora diseñada y construida de forma que todos los componentes trabajen de manera paralela como por ejemplo las *Cray*, en donde el software incluyendo el sistema operativo puede acceder a todos los recursos de la máquina, que están estrechamente relacionadas entre sí.

Al ejecutar una aplicación, ésta ve al sistema homogéneo, por lo que el sistema operativo debe conocer todos los recursos distribuidos, para presentar al usuario aquello que requiere, aunque éstos se encuentren dispersos.

- **Acoplamiento medio**

  En este grupo están las máquinas paralelas conformadas por elementos que no necesitan un conocimiento tan exhaustivo de todos los recursos, pero que los emplea para aplicaciones de muy bajo nivel. Ejemplos de estas máquinas paralelas son los *cluster* basados en *OpenMosix*.

En un *cluster OpenMosix* se realiza una división de procesos en una parte local y en otra remota, requiriendo los recursos de otro nodo, pero sin el conocimiento total de sus componentes tanto de software como de hardware.

- **Acoplamiento pobre**

  Son los casos en los que los recursos están divididos físicamente entre diferentes componentes de hardware, no estando a un nivel alto de integración. Generalmente se basan en aplicaciones construidas por bibliotecas preparadas para este propósito, agrupadas de forma específicas que generan la máquina paralela en sí.

Las bibliotecas de mayor difusión para un acoplamiento pobre son *PVM* y *MPI*, las cuales están basadas en el esquema de paso de mensajes.

1.3 Granularidad

La importancia de identificar la granularidad de un problema computacional dado, es porque a partir de ella se puede determinar las porciones en las que se puede subdividir

y la interdependencia que hay entre cada uno de las subdivisiones hechas.

Si se pueden realizar subdivisiones sobre un problema computacional, es posible definir un esquema de paralelización viable. Por ello primero se enumeran los grados de granularidad más generales.

La granularidad es el tamaño de las piezas en que se divide una aplicación; dichas piezas puede ser una sentencia de código, una función o un proceso que se ejecutarán en paralelo.

Hay que tomar en cuenta que al paralelizar una aplicación es necesario contar con un lenguaje o biblioteca, incluso un ambiente que brinde las herramientas necesarias para que se programe y se pueda ejecutar en paralelo; segmentando el código en fragmentos que se correrán simultáneamente en varias unidades de procesamiento.

La granularidad se clasifica como: grano fino, medio y grueso.

- **De grano fino**

  Es cuando el código se divide en una gran cantidad de piezas pequeñas. A nivel de sentencia el ciclo se divide en varios subciclos que se ejecutarán en paralelo (hilos); se le conoce además como paralelismo de datos.

Para poder efectuar efectivamente estas actividades es necesario tener un sistema de cómputo altamente acoplado.

- **De grano grueso**

  Es a nivel de subrutinas o funciones, donde las piezas son pocas y de cómputo más intensivo que las de grano fino. Se le conoce como paralelismo de tareas.

- **De grano medio**

  Es un grado intermedio entre grano grueso y grano fino, y corresponde a la paralización de segmentos de código que conforman a las funciones, pero que no llega a trabajar a un nivel tan fino como a nivel de sentencias.

Si un análisis realizado sobre una aplicación, indica que su granularidad es de grano medio, se puede implementar en paralelo sobre un sistema pobremente acoplado, como un sistema de servidores o un *cluster.*

1.4 Esquema de un cluster Beowulf

Como se puede apreciar, sobre una arquitectura *MIMD* se puede trabajar bajo un esquema del tipo *SIMD*, si en lugar de distribuir diferentes instrucciones, se distribuye una copia igual.

En un *cluster* tipo *Beowulf* se puede considerar que tiene una arquitectura del tipo *MIMD*, dado que cada nodo es en sí capaz de operar de manera independiente al resto de los demás y compartir información entre todas las partes, sin embargo es posible programarlo pensando en un esquema del tipo *SIMD* o *MIMD*, de acuerdo al problema que se desea resolver.

Además, un *cluster Beowulf* tiene una baja integración dado que las unidades de procesamiento se encuentra confinadas con sus propios recursos y se necesita de un método externo para compartirlos por lo que se considera pobremente acoplado,

Un esquema *SIMD* es mucho más fácil de implementar que un esquema *MIMD*, porque en un programa siempre existe una interdependencia entre los datos que se pretenden

procesar, y que resulta complejo de eliminar o de reducir; dado que es más fácil hacer las mismas tareas sobre diferentes datos, que hacer diferentes tareas sobre diferentes datos e integrar los diferentes resultados.

Sin embargo como se mencionó, es posible programar en el *cluster* pensando en ambos esquemas, pero debido a la pobre integración que hay; la granularidad a considerar en los programas debe ser de grano medio tendiendo a grueso, de acuerdo a lo eficiente de las comunicaciones entre los nodos. Si las comunicaciones resulta muy lentas se considera conveniente programar a nivel de grano grueso.

Capítulo 2 Construcción del *cluster* tipo *Beowulf*

El construir un *cluster* tipo *Beowulf*, es más una serie de recomendaciones acerca de como interconectar los componentes y como configurarlos (una filosofía), que una metodología estricta de un procedimiento de armado.

Los *clusters* tipo *Beowulf* fueron en su inicio supercomputadoras de hechura casera. Aunque la idea resultó tan conveniente que recientes fechas, es posible contactar a grandes empresas dedicadas a la venta de equipo de cómputo, para que construyan un *cluster* a la medida, tal es el caso de *HP* o *Dell*, incluso compañías más pequeñas que se pueden localizar en la región donde se desea adquirir el hardware.

2.1 Armado el cluster

Para la construcción de un *cluster Beowulf* es necesario contar con un conjunto de computadoras, preferentemente que sean iguales en sus características hardware, como lo es: el mismo tipo de procesador, la misma velocidad de comunicaciones en el bus de datos, la mayor cantidad de memoria RAM. Ésta última no es necesario que sea igual para todas las computadoras, sin embargo siempre es conveniente que se pueda colocar la mayor cantidad posible de ella.

En realidad solo importa que sean los procesadores de la misma familia, y que sean lo más avanzados de los que se pueda disponer, al igual que para el bus de datos de la tarjeta principal (*mainboard* ó *motherboard)*, dado que la comunicación entre los componentes es vital para este tipo de supercomputadoras.

Se le llama nodo a cada uno de las computadoras que componen el *cluster Beowulf* y básicamente podemos encontrar dos tipos distintos: nodo maestro y nodo esclavo.

Además de las computadoras es necesaria la existencia de un sistema de comunicaciones entre ellas, por lo que cada computadora debe contar con algún medio para comunicarse con otras, que generalmente es un recurso de red. Es importante que sea un sistema de comunicación muy rápido, porque al ser el único medio de intercambio de información entre los diferentes nodos, conviene reducir al máximo el tiempo que tarda la información en viajar de un nodo a otro.

Aunque se han hecho grandes logros en la capacidad de procesamiento, el avance no se da en las mismas proporciones en todas las partes que componen a una computadora y sigue siendo una computadora tan rápida, como el componente más lento con el que interactúe.

Cuando aparecen los primeros *clusters* tipo *Beowulf*, el cuello de botella fue tradicionalmente el medio de comunicación entre nodos, es decir la velocidad de la red. Así que para obtener el máximo beneficio de una máquina de este tipo, se buscó reducir lo más posible el tiempo en las comunicaciones y aumentar al máximo el uso de los procesadores.

Actualmente se han hecho muchos avances en esta línea, para lograr reducir al máximo la latencia[2] de una red, además de aumentar la cantidad de datos que pueden viajar, así como la velocidad de respuesta de los periféricos. De estas investigaciones han surgido varios productos que permiten un alto rendimiento en comunicaciones como es *Myrinet*, *Infiniband*, *Quadrics* y por supuesto la tecnología *Ethernet* que ya permite transferencias de 1 Gbs y siguen avanzando.

Estas nuevos esquemas de comunicación permiten disminuir el tiempo de intercambio de información entre nodos dando la posibilidad de tener programas más granulados y una mayor interacción entre los procesos.

2 Tiempo de retraso que se da entre dos puntos durante la comunicación

Hay que aclarar que en entornos más especializados, existen compañías dedicadas a la construcción de computadoras que han sido reducidas en dimensión para poder armar supercomputadoras; buscando eliminar componentes genéricos innecesarios y poder incluir los elementos importantes con la intención de disminuir el consumo de espacio y energía.

Cuando se construye un *cluster* tipo *Beowulf*, se deben hacer las mismas evaluaciones que para un *data center*[3]. Es indispensable considerar cuanta energía eléctrica se va a consumir y cuanta energía térmica se generará con el funcionamiento del equipo, tanto de manera individual como en conjunto.

Generalmente el consumo eléctrico de equipo dedicado al uso de un *cluster*, puede requerir de adecuaciones eléctricas, como pude ser el cambio del voltaje por el tipo adecuado de consumo que realice, así como consideraciones para el uso de equipo de respaldo, como lo es el sistema de regulado eléctrico y las baterías, para evitar variaciones e interrupciones.

Para mantener adecuadamente la temperatura del lugar, es importante calcular la cantidad de calor generado por hora y encontrar un sistema de refrigeración que logre eliminar un 30 % más que el estimado, además de considerar la forma en que el calor generado por el equipo se distribuye por el cuarto. Por supuesto que las estimaciones se hacen considerando que el equipo se encuentra trabajando a su máxima capacidad y que por lo tanto, el calor generado por cada equipo es el máximo. Todas estas consideraciones se pueden conseguir del proveedor del equipo de cómputo.

Las adecuaciones del sitio para poner en funcionamiento el equipo que conforma al *cluster* consistió en proporcionar instalaciones eléctricas adecuadas (bifásica con 220 volts), lo que implicó conseguir un transformador para cambiar el voltaje de 440 a 220.

3 Un data center habitualmente es el sitio donde se instalan mainframes o servidores

El sistema de refrigeración consistió en la instalación de un sistema evaporador – condensador que pudiera manejar alrededor de 16000 btus / hora.

Después de ello, se instaló el hardware que corresponde al *cluster*, como lo es la colocación del *rack*, colocación de los equipos de cómputo, colocación de dos *switches* para diferente tipo de red, colocación de un *switch* del tipo pantalla-teclado-ratón, así como el tendido eléctrico, el alambrado de la red 10/100 *ethernet*, instalación de las tarjetas y del alambrado de la red Myrinet; además de cableado de las conexiones del *switch* pantalla-teclado-ratón con cada uno de los cpus.

2.2 Instalación del sistema operativo

Después de colocar todos los elementos físicos que conforman al *cluster*, lo siguiente es la instalación del sistema operativo, que en este caso el escogido es *Red Hat Enterprise Server 3.0* porque es el que, a recomendación de los fabricantes de software, resulta compatible con la paquetería que se instalará posteriormente.

La instalación se realiza desde los CDROMs provistos por el vendedor del hardware, lo que permite la instalación y configuración de todos los paquetes que son necesarios, además de aquellos paquetes que conforman al sistema operativo.

En el proceso de instalación se configura lo general, la fecha y lugar, el idioma que se desea usar, la dimensión de las particiones deseadas y la habilitación de las tarjetas de red.

El esquema de particiones es diferente el del nodo maestro, del resto de los nodos solamente en el tamaño asignado a las particiones, porque el nodo maestro es el que contiene los directorios de los usuarios y los directorios que posteriormente son

distribuidos. Sin embargo todos cuentan con las siguientes particiones: *swap, /, /boot, /home, /usr /var, /tmp*.

Como originalmente los nodos adquiridos son diseñados por el vendedor de hardware para que funcionen como servidores, las particiones hechas corresponden con un esquema adecuado para una computadora que trabajará como servidor; y como este esquema conviene para la configuración del *cluster*, se considera su utilización solo modificando los tamaños de las particiones.

En el caso del nodo maestro, se agrega además una partición llamada */netfsr* que posteriormente se distribuye en red mediante *NFS*[4].

Como cada uno de los nodos tiene incrustadas dos tarjetas de red, durante el proceso de instalación se puede configurar tanto el nombre del *host*, como las IP's asignadas a las tarjetas. El nombre de *host* fue csnyXX, en donde las XX representan números consecutivos del 1 al 16. Los segmentos de red asignados a los nodos esclavos fueron rangos de IP's privadas, para la primera fue 192.168.1.X, y para la segunda fue 192.168.3.X.

En el caso del nodo maestro, como es el que se conecta al *firewall* para que se pueda interactuar con el *cluster* desde cualquier punto, a la primera tarjeta de red se le asignó la IP 192.168.1.1, y a la segunda que se conecta hacia el *firewall* la IP 192.168.2.2.

Como todos los nodos son iguales, se instala el software de la misma forma en cada uno de los 16 nodos que componen al *cluster*.

2.3 Configuración del server con las herramientas gráficas

4 NFS significa Network File System, es una forma nativa en UNIX de compartir un directorio en red

En ambiente gráfico, todas las herramientas de administración de *Red Hat* se pueden invocar desde una consola a partir del prefijo *redhat-config-*<*servicio*>, en donde <*servicio*> es cualquiera de los elementos del servidor que se quieran configurar. Por mencionar algunos:

- *redhat-config-date*: permite configurar la fecha
- *redhat-config-screen*: para configurar tanto el monitor como la tarjeta de video
- *redhat-config-services*: para configurar que servicios y *daemons*[5] se requieren iniciar al momento de encender la computadora
- *redhat-config-network*: permite configurar las tarjetas de red
- *redhat-config-packages*: para la instalación o remoción de paquetes
- *redhat-config-nfs*: permite configurar el servidor de *NFS*

2.4 Configurando el ambiente gráfico

Aunque no es indispensable el contar con un servidor gráfico en cada uno de los nodos y de hecho es recomendable deshabilitar y desinstalar, para el caso del nodo maestro se configura y conserva por comodidad, para poder hacer uso de las herramientas gráficas de configuración.

Como el proveedor del hardware también proporciona los controladores[6] para la tarjeta de video, solo hay que hacer algunos ajustes para que se despliegue adecuadamente. Se usa la herramienta *redhat-config-screen*, que permite además de tratar de configurar la tarjeta de video, también el monitor, que en este caso es un monitor *LCD* genérico con una resolución de 1024 x 768.

Estos parámetros se le dan y se reinicia el servidor gráfico. Posteriormente se hacen pruebas para exportar el ambiente gráfico desde una sesión remota.

5 Es un programa que se ejecuta al iniciar el sistema operativo y que trabaja en "background"
6 También llamados drivers, programas necesarios para que el S. O. pueda interactuar con el hardware

2.5 Configurando los aspectos de red

Se cuenta con diferentes redes tanto en la conexión exterior como entre los nodos, por ello a cada una se le asigna un segmento diferente, usando como principal el 192.168.x.x. que se asignaron al hardware respectivo durante el proceso de instalación tal como se comentó previamente.

Después de instalado del hardware de red, las conexiones con los respectivos *switches* y del sistema operativo; así como la configuración de cada uno de estos elementos, se prueba la comunicación entre los nodos mediante el comando *ping,* para mandar paquetes entre los nodos y verificar que éstos respondan a la solicitud.

Como únicamente se tiene un *switch ethernet*, solo es posible conectar en red una de las dos tarjetas que trae cada nodo. Así que se escoge la primera de ellas para que constituya la red *ethernet*.

El comando *ping* se utiliza empleando las IP's de las tarjetas que sí tienen una conexión al *switch ethernet*, que corresponden al segmento 192.168.1.x.

Desde el nodo maestro al primer nodo esclavo:

```
$ ping 192.168.1.2
```

No se instala un servidor de *DNS* porque resulta innecesario; en su lugar se crea y utiliza el archivo */etc/hosts* que contiene las IP*'s* y los nombres para la resolución de IP*'s* a partir de nombres.

Considerando a la red *Myrinet* como primordial al garantizar una latencia menor, se

construye el archivo *hosts* con las IP*'s* asignadas a las tarjetas *Myrinet*. El archivo a continuación:

```
192.168.0.1       csny01
192.168.0.2       csny02
192.168.0.3       csny03
192.168.0.4       csny04
192.168.0.5       csny05
192.168.0.6       csny06
192.168.0.7       csny07
192.168.0.8       csny08
192.168.0.9       csny09
192.168.0.10      csny10
192.168.0.11      csny11
192.168.0.12      csny12
192.168.0.13      csny13
192.168.0.14      csny14
192.168.0.15      csny15
192.168.0.16      csny16
```

Este archivo se crea en cada uno de los nodos, para permitir el uso del nombre del *host* en todas las actividades que corresponden a la comunicación entre los nodos del *cluster*.

En el caso del nodo maestro, además de utilizar el archivo */etc/host* para resolver las IP's a partir de los nombres, también se le configuró las direcciones de un servidor de *DNS* en la configuración de la segunda tarjeta de red, porque el nodo maestro si tiene contacto con la Internet y se requiere en ocasiones navegar por ella.

2.6 Montando el servidor NFS

Para este servicio hay que instalar en el nodo maestro el servidor *NFS,* mientras que en

el caso de los esclavos hay que instalar los clientes. Esto por supuesto se puede gestionar desde el administrador de paquetes que se invoca con el comando *redhat-config-packages*, seleccionando en el apartado de servidores todo lo referente a *NFS*.

Después se edita el archivo para la configuración del servidor *NFS* que se encuentra en */etc/export* cuya estructura es:

```
directorio  maquina-anfitrion(opciones)
```

Sin embargo la configuración y habilitación del servicio *Network File System*, se puede realizar desde la herramienta *redhat-config-nfs*, en donde solo hay que especificar cual es el directorio que se quiere compartir y con que atributos se desea hacerlo.

El archivo queda de la siguiente forma:

```
/netfsr     192.168.0.1(rw,sync)
```

En los nodos esclavos es necesario montar el directorio que se comparte con el servidor *NFS* desde el nodo maestro para que puedan accesarlo, lo cual se logra con el comando *mount*. En cada nodo esclavo se ejecuta como root:

```
# mount 192.168.0.1:/netfsr /netfsr
```

2.7 Montando el servidor DHCP

Aunque los nodos son pocos y se pueden manejar IP's estáticas para asignarlos a cada uno de ellos, resulta útil la existencia de un servidor *DHCP* el cual se coloca en el nodo maestro. Esto permite que en caso de agregar o perder nodos, las IP's se reasigne de manera automática, aunque solo en el caso de los nodos agregados sería además inmediata la asignación. Además fue necesario disponer de un servidor *DHCP* porque

el *switch Myrinet* empleado, se puede monitorear vía *web*, y necesita que se le asigne una IP de manera dinámica.

Para montar el servidor *DHCP* solo hay que instalar los paquetes necesarios que son el servidor y opcionalmente se puede instalar el cliente.

Para la instalación, se pueden hacer por consola o por ambiente gráfico. En el caso de la consola, el comando utilizado es *rpm*; por ambiente gráfico se emplea el programa *redhat-config-packages*.

Después de instalarlo, se crea y modifica el archivo de configuración del servicio llamado *dhcpd.conf* que se encuentra en */etc*. El archivo es semejante a este:

```
# DHCP Server Configuration file.
#   see /usr/share/doc/dhcp*/dhcpd.conf.sample
#
ddns-update-style interim;
subnet 192.168.1.0 netmask 255.255.255.0 {
        option routers                  192.168.1.1;
        option subnet-mask              255.255.255.0;

        option domain-name-servers ip.servidor.dns1, ip.servidor.dns1,
132.248.204.2;

        option time-offset              -18000;     # Eastern Standard Time

        range 192.168.1.2 192.168.1.200;
}
```

Habiendo instalado y configurado el servidor, solo hay que iniciarlo. En *Red Hat* y otras distribuciones basadas en ésta, pueden iniciarse ejecutando */etc/init.d/<servicio> <estado>*, en este caso <*servicio*> es *dhcpd* y <*estado*> es *start*. Como root:

```
# /etc/init.d/dhcpd start
Iniciando dhcpd:                                           [  OK  ]
```

Para que esto se inicie automáticamente al encender el servidor, se marca el servicio desde la herramienta *redhat-config-services*, que es la que permite administrar los servicios.

2.8 Configurando el servicio remote shell

Anteriormente en los sistemas tipo *UNIX*, cuando se deseaba realizar la ejecución de comando o tarea de manera remota, se hacia uso del comando *rsh*, siempre y cuando el usuario que intentara realizar la tarea dispusiera de una cuenta en el sistema donde quisiera ejecutar de manera remota el comando. Esto se realizaba de la siguiente forma:

```
$ rsh <host> <comando>
```

Donde *host* correspondía a la IP o nombre completo de la máquina remota, y *comando* correspondía a la tarea o comando que se desease ejecutar.

Con la popularización de la Internet y el acceso masivo a la misma, llegaron también los problemas de seguridad y confiabilidad; con lo que se idearon otras formas de realizar las conexiones, buscando garantizar que se preservara lo anterior.

El problema que aparece con el comando *rsh*, es que se da por hecho que quien hace la solicitud ha sido un usuario que ya se ha validado en el sistema anfitrión y por lo tanto es un usuario confiable, además de que los mensajes, como la ejecución de los comandos viaja sin ningún tipo de cifrado, entonces sí existe un medio que interfiera entre el sistema anfitrión y al que se realiza el acceso remoto, todo queda visible.

Razones por las cuales el comando *rsh* no se considera ni seguro ni confiable.

Todas las tareas que antes se realizaban con comandos para acceso remoto como el caso del *telnet* o el *rsh* son realizadas ahora por *ssh*, *scp* y *sftp*; por lo cual todos los comandos para conexión remota han sido deshabilitados y en la mayoría de los casos ni siquiera se instalan en conjunto con el resto del sistema operativo.

Aunque *mpich* se puede instalar para que haga uso de *ssh* en sustitución de *rsh*, dado que se puede configurar *ssh* para que actúe de la misma forma que *rsh*, existe una reducción en el rendimiento entre las comunicaciones, porque además del tiempo consumido en el envío o recepción de datos, ahora hay que agregar el tiempo que tarda un paquete en ser cifrado o descifrado. Además en el caso del *cluster*, el *mpich* resultó un poco inestable cuando se usó *ssh* en lugar de *rsh*.

El comando *rsh* es utilizado para establecer comunicación entre los diferentes nodos, tanto el maestro como los esclavos, pero no se permite la ejecución del comando *rsh* desde cualquier parte de la Internet al *cluster*. Como los paquetes solo viajan entre los nodos y éstos a su vez solo están a disponibilidad del nodo maestro y no se pueden acceder a ellos excepto de manera directa, se puede prescindir del comando *ssh*, dado que solo se puede comprometer a un nodo esclavo a través del nodo maestro, pero si se tiene control de este último, no tiene sentido comprometer a ninguno de los nodos esclavos.

Como *rsh* por razones de seguridad, no se instala y está deshabilitado, es necesario realizar algunos pasos para poder hacer uso de él.

Se instala el paquete que contiene el *rsh*, tanto cliente como servidor en cada uno de los nodos desde el gestor de paquetes de *Red Hat* (*redhat-config-packages*).

Se habilita el servicio para que cuando se enciendan los nodos, inicien

automáticamente el servidor rsh; ésto se habilita desde el gestor de servicios del sistema operativo (*redhat-config-services*).

Se crea un archivo llamado *hosts.equiv* que contienen los nombres de las máquinas que se consideran confiables.

El archivo *.rhost* que se coloca en el directorio */etc/securetty* es el siguiente:

```
csny01 root
csny02 root
csny03 root
csny04 root
csny05 root
csny06 root
csny07 root
csny08 root
csny09 root
csny10 root
csny11 root
csny12 root
csny13 root
csny14 root
csny15 root
csny16 root
```

2.9 Instalación del compilador Fortran 90 y de mpich.

Primero se instala el compilador de Fortran 90 de la compañía *Portland Group*, el cual cuenta con un *script* de instalación, por lo que solo hay que ejecutarlo y éste hace todo el proceso de ubicación y configuración de los programas y archivos necesarios.

Como el hardware de red *Myrinet* es especializado, no se puede utilizar la versión

mpich oficial, pero la compañía *Myrinet* provee de una versión modificada que trabaja con las tarjetas de red que distribuye.

El paquete provee de un *script* que realiza todo el proceso de configuración, compilado e instalación; solo hay que modificar el método de ejecución de comandos remotos y en el caso de existir algún compilador diferente a los que trae indicados, también se modifica. Inicialmente trabaja con *ssh*, así que se modifica para especificar que se usa *rsh* y se cambia el compilador *GNU*[7] de Fortran por el de *Portland Group*.

Previamente se crea la carpeta */opt/mpich*, donde se instalará todo lo relacionado a *mpich*.

Después de ejecutado el *script* de instalación y ya teniendo habilitado el servidor *NFS*, se pueden copiar el directorio de ejemplos a los directorios de cada usuario, o se puede distribuir por *NFS*, para probar la compilación y ejecución de los ejemplos de programas paralelos.

2.10 Instalación y configuración de un firewall

Los intrusos potenciales buscan computadoras en la Internet explorando puertos que puedan vulnerar y entrar. Sus motivaciones en general son obtener información privada, como contraseñas, cuentas financieras y otro tipo de información sensible, además de poder utilizar las computadoras comprometidas para realizar otras actividades como:

- Lanzar ataques de negación de servicio (*DoS*, por sus siglas en inglés) contra sitios web.
- Distribuir software ilegal
- Utilizarlas para realizar ataques de intrusión hacia otras computadoras.

7 GNU es un proyecto para la generación de software libre

- Utilizarlas para la difusión de correo no deseado, comúnmente llamado *SPAM*.

Se puede añadir una capa importante de protección entre una computadora y la Internet utilizando un sistema llamado *firewall*.

Un *firewall* puede bloquear entradas no autorizadas a una computadora, así como restringir el trafico tanto entrante como saliente, de ello existen dos tipos de *firewall*: por software y por hardware.

Los *firewalls* por software son programas que se encargan de administrar las conexiones y los puertos como cerrar, abrir, o filtrar un puerto, así como autorizar o denegar las peticiones y paquetes que llegan o van a la Internet.

Los *firewall* por hardware, son equipos de conexión a red, independientes de las computadoras que se han de conectar, construidos para realizan las mismas actividades que un *firewall* por software.

Los *firewalls* por software generalmente se instalan en la misma computadora que se utilizar para navegar en la Internet. Sin embargo se puede utilizar una computadora únicamente para contener un programa *firewall* y funcionar como tal.

Para proteger al *cluster* de riesgos de intrusión desde la Internet, se utiliza una PC en desuso para que actúe como un *firewall*. Sus características son las siguientes:

- Procesador Pentium a 233 MHz
- 64 MB de RAM
- 1 disco duro de 2 GB
- 2 tarjetas de red 10/100 Mbs
- 1 unidad floppy 3 ½

El sistema operativo que se emplea es *OpenBSD* 3.7, elegido porque demanda pocos recursos además de ofrecer un alto nivel de estabilidad y seguridad.

2.10.1 Obtención del software para la instalación

Hay diferentes métodos de instalación, aunque para este caso el proceso se realiza desde red a partir de un floppy, porque la máquina no puede iniciar desde la unidad de CDROM. La imagen del disco de arranque que se almacena en el floppy se obtiene del siguiente sitio ftp:

```
ftp://ftp.openbsd.org/pub/OpenBSD/3.7/i386/floppy37.fs
```

Y para lograr utilizar la imagen *floppy37.fs* disponible del sitio anterior es necesario que se formatee un floppy y se grabe la imagen, desde un sistema tipo UNIX empleando los siguientes comandos:

```
>$ fdformat /dev/fd0H1440
```

que realiza el proceso de formateo, y:

```
>$ dd if=floppy37.fs of=/dev/fd0 bs=32k
```

que copia la imagen en el floppy.

Con este disco se arranca la máquina, previamente conectada a la red.

2.10.2 Inicio de la instalación

Durante el proceso de arranque las líneas que se despliegan en pantalla es la salida de dmesg, el núcleo del sistema, que muestra información sobre los dispositivos

encontrados y su ubicación.

Al terminar de cargar el instalador se ve el siguiente mensaje:

```
rootdev=0x1100 rrootdev=0x2f00 rawdev=0x2f02
erase ^?, werase ^W, kill ^U, intr ^C, status ^T
(I)nstall, (U)pgrade or (S)hell? i
```

Como es un sistema nuevo el que se requiere, se elige la opción *(I)nstall* para iniciar el proceso de instalación de *OpenBSD* en la máquina, sobreescribiendo cualquier información contenida en el disco duro. Después muestra otra pantalla con más datos, desplegando al final un par de preguntas necesarias para continuar con el proceso, que son las siguientes:

```
Specify terminal type: [vt220] Enter
Do you wish to select a keyboard encoding table? [n] Enter
```

Presionando la tecla *Enter* para que tome las opciones genéricas, las cuales resultan válidas y funcionan adecuadamente para el resto del proceso de instalación. Como no se selecciona una tabla de codificación de caracteres, se asume que se tiene un teclado en inglés.

Después muestra algunas advertencias, indicando el riesgo de perder la información contenida en el disco duro; para al final solicitar una confirmación de que se desea continuar con la instalación del sistema operativo.

```
Proceed with install? [no] y
```

Se afirma que se desea proceder con la instalación con la tecla “y”.

2.10.3 Configuración de los discos

La configuración del disco para procesadores del tipo *i386* se hace en dos partes. Primero se define una partición del disco duro de *OpenBSD* utilizando el comando *fdisk*, y luego se subdivide ésta en varias particiones del tipo *OpenBSD* utilizando *disklabel*.

```
Cool!  Let's get to it...

You will now initialize the disk(s) that OpenBSD will use. To enable all
available security features you should configure the disk(s) to allow the
creation of separate filesystems for /, /tmp, /var, /usr, and /home.

Available disks are: wd0.
Which one is the root disk? (or done) [wd0] Enter
```

El disco raíz es el disco desde el que se arrancara el sistema, y suele ser en el que reside además el espacio de la memoria de intercambio llamado *swap*[8]. Los discos IDE se muestran como *wd0* para el primero detectado, *wd1* para el segundo disco, y así se continua con el resto, cuando éstos existen.

El programa de instalación muestra una lista con todos los discos que encuentre en el sistema. Para la máquina en cuestión sólo aparece el único disco instalado. Como no se pretende usar otro sistema operativo, se utiliza todo el espacio disponible del disco duro.

```
Do you want to use *all* of wd0 for OpenBSD? [no] y
```

De este modo se grabaran en el disco un “Registro de Arranque Maestro” (*MBR*, Master Boot Record) y una tabla de particiones mas simples: una partición del tamaño de todo el disco duro configurada con el tipo de partición de *OpenBSD*, y marcada como de arranque.

8 Es el nombre que habitualmente se utiliza en los entornos tipo UNIX.

Se invoca al programa *fdisk* para realizar la primera parte correspondiente a la partición del disco duro.

Con el comando *fdisk* se aprecian diferentes propiedades del disco duro como: su geometría, el dispositivo asociado en el directorio */etc*, el espacio ocupado y el disponible, donde comienza y termina cada partición creada, así como un *prompt* para la modificación de las particiones.

Al invocar el comando se obtiene la siguiente pantalla:

```
Enter 'help' for information
fdisk: 1>
```

Como se ve al principio, existe la opción *help*, que despliega la ayuda del comando *fdisk*, mostrando las opciones disponibles:

- *help*: lista las opciones disponibles
- *manual*: muestra la página completa del manual de fdisk de OpenBSD
- *reinit*: reinicaliza el MBR previamente cargado
- *setpid*: coloca el identificador de una tabla dada
- *disk*: edita los datos de los discos existentes
- *edit*: edita la tabla de particiones dada
- *flag*: marca una partición como “booteable”
- *update*: actualiza el código máquina en el *MBR*
- *select*: selecciona una tabla de particiones extendidas
- *print*: imprime la tabla de particiones
- *write*: escribe las particiones en el disco
- *exit*: sale sin salvar los cambios
- *quit*: Sale del programa grabando los cambios realizados

- *abort*: Aborta el programa sin salvar los cambios recientes

Se emplea la opción *reinit*, que elimina la tabla de particiones existente y crea una nueva partición grande *OpenBSD*, que se marca como activa e instala en el *MBR* el código de inicio de *OpenBSD*. Solamente es necesario emplear la primer letra de cada opción:

```
fdisk: 1> r
```

Para guardar los cambios y continuar con la instalación del sistema, en el *prompt* utilizamos la opción *write* para guardar las modificaciones y *quit* para salir de *fdisk*.

```
fdisk:*1> w
Writing MBR at offset 0.
wd0: no disk label
fdisk: 1> q
```

Ahora se utiliza *disklabel* para subdividir la partición de *OpenBSD*, comando que primero muestra información relacionada al disco además de las particiones que contiene y posteriormente habilita un *prompt* desde donde se pueden realizar las subdivisiones requeridas.

Como se aprecia a continuación, se utiliza el caracter '?' para mostrar la ayuda del comando:

```
Initial label editor (enter '?' for help at any prompt)
> ?
Available commands:
        p [unit]  - print label.
        M         - show entire OpenBSD man page for disklabel.
        e         - edit drive parameters.
        a [part]  - add new partition.
        b         - set OpenBSD disk boundaries.
```

```
          c [part]  - change partition size.
          d [part]  - delete partition.
          D         - set label to default.
          g [d|b]   - Use [d]isk or [b]ios geometry.
          m [part]  - modify existing partition.
          n [part]  - set the mount point for a partition.
          r         - recalculate free space.
          u         - undo last change.
          s [path]  - save label to file.
          w         - write label to disk.
          q         - quit and save changes.
          x         - exit without saving changes.
          X         - toggle expert mode.
          z         - zero out partition table.
          ? [cmnd]  - this message or command specific help.
  Numeric parameters may use suffixes to indicate units:
          'b' for bytes, 'c' for cylinders, 'k' for kilobytes, 'm' for
megabytes,
          'g' for gigabytes or no suffix for sectors (usually 512 bytes).
          Non-sector units will be rounded to the nearest cylinder.
  Entering '?' at most prompts will give you (simple) context sensitive help.
  >
```

En el disco raíz se deben crear las dos particiones 'a' y 'b', dado que el proceso de instalación no seguirá adelante hasta que éstas estén disponibles.

La partición 'a' se usa para el sistema de archivos raíz[9] ('/'), y 'b' como espacio para la memoria de intercambio (*swap*).

En el *firewall* se crean únicamente tres particiones; las particiones '*a' y 'b'* ya mencionadas, y *d* para que contenga el subdirectorio */var*, tal como se muestra a continuación:

9 Comúnmente llamada root y que se presenta con el carácter /

```
> a a
offset: [3069360] Enter
size: [36030960] 1000M
Rounding to nearest cylinder: 307440
FS type: [4.2BSD] Enter
mount point: [none] /
> a b
offset: [3376800] Enter
size: [35723520] 128M
Rounding to nearest cylinder: 614880
FS type: [swap] Enter
> a d
offset: [3991680] Enter
size: [35108640] 500M

Rounding to nearest cylinder: 245952
FS type: [4.2BSD] Enter
mount point: [none] /var
```

Ya creadas las particiones se sale de *disklabel* y se guardan los cambios con la opción q

```
> q
Write new label?: [y] Enter
```

Hay una partición *c* que parece haber sido ignorada. Esta partición representa al disco duro completo. En general, a las particiones no se les asigna ninguna letra en concreto, a excepción de *a*, que es la partición raíz, *b* la partición para la memoria de intercambio, y *c* que es una representación de todo el disco duro; el resto de las letras hasta la *p*, pueden ser asignadas a cualquier partición según se desee.

Por último para terminar con la configuración de los puntos de montaje, dado que previamente se ha configurado éstos con *disklabel* sólo consiste en confirmar las selecciones realizadas.

```
The root filesystem will be mounted on wd0a.
wd0b will be used for swap space.
Mount point for wd0e (size=423582k), none or done? [/var] Enter
Done - no available disks found.

You have configured the following partitions and mount points:

wd0a /
wd0d /var

The next step creates a filesystem on each partition, ERASING existing data.
Are you really sure that you're ready to proceed? [no] y
```

A partir de este punto el programa dará formato a todos los sistemas de archivo.

2.10.4 Configuración del nombre de anfitrión del sistema y de la red

A continuación se configura el nombre de anfitrión que utilizará el sistema, el cual será el que se emplee cuando se generen las claves criptográficas durante el primer arranque después de la instalación. La generación de estas claves se realiza tanto si la red está configurada como si no.

```
Enter system hostname (short form, e.g. 'foo'): gnsyf
```

Es conveniente que la red esté configurada si se va a realizar una instalación por *FTP*, ya que estas instalaciones de los paquetes se basarán en la información que introduzca en este punto.

Se puede usar *DHCP* para IP*'s* dinámicas en lugar de estáticas. En el caso de *DHCP*, la mayor parte de la información de la red se obtiene del servidor de *DHCP*; el proceso de

instalación da la oportunidad de confirmar los datos obtenidos a través del servidor *DHCP*, que se despliegan en un pantalla como la siguiente:

```
Configure the network? [yes] Enter
Available interfaces are: tl0.
Which one do you wish to initialize? (or 'done') [tl0] Enter
Symbolic (host) name for tl0? [puffy] Enter
The default media for tl0 is
        media: Ethernet autoselect (100baseTX full-duplex)
Do you want to change the default media? [no] Enter
IP address for tl0? (or 'dhcp') dhcp
Issuing hostname-associated DHCP request for tl0.
Internet Software Consortium DHCP Client 2.0pl5-OpenBSD
Listening on BPF/tl0/XX:XX:XX:ec:f5:df
Sending on   BPF/tl0/XX:XX:XX:ec:f5:df
Sending on   Socket/fallback/fallback-net
DHCPDISCOVER on tl0 to 255.255.255.255 port 67 interval 1
DHCPOFFER from xxx.xxx.xxx.128
DHCPREQUEST on tl0 to 255.255.255.255 port 67
DHCPACK from xxx.xxx.xxx.128
New Network Number: xxx.xxx.xxx.214
New Broadcast Address: xxx.xxx.xxx..255
bound to xxx.xxx.xxx.55 -- renewal in XXXXX seconds.
Done - no available interfaces found.
DNS domain name? (e.g. 'bar.com') [torre_ingenieria.mx] Enter
DNS nameserver? (IP address or 'none') [xxx.xxx.xxx.1] Enter
Use the nameserver now? [yes] Enter
Default route? (IP address, 'dhcp' or 'none') [xxx.xxx.xxx.254] Enter
add net default: gateway  xxx.xxx.xxx.254
Edit hosts with ed? [no] Enter
Do you want to do any manual network configuration? [n] y
```

Al terminar de configurar la red externa mediante el cliente *DHCP*, podemos configurar otra interfaz de red que se conecta al *cluster*, y que éste usará para conectarse a la Internet. Los datos son de una red privada en el segmento 192.168.2.x.

Lo siguiente que se solicita es la contraseña para la cuenta del superusuario que también se le llama *root*:

```
Password for root account? (will not echo) ********
Password for root account? (again) ********
```

Posteriormente al arrancar el sistema es posible crear cuentas para otros usuarios, con el propósito de darle mantenimiento al *firewall*, sin conectarse con la cuenta de *root* a través del servicio *ssh*.

2.10.5 Elección del medio de instalación

Después de configurar la red, el *script* de instalación da la oportunidad de realizar cambios manuales a la configuración y se montarán los sistemas de archivos creados. Finalmente los discos ya estarán listos para instalar en ellos *OpenBSD*.

Lo siguiente es escoger un medio de instalación:

```
You will now specify the location and names of the install sets you want to
load. You will be able to repeat this step until all of your sets have been
successfully loaded. If you are not sure what sets to install, refer to the
installation notes for details on the contents of each.

Sets can be located on a (m)ounted filesystem; a (c)drom, (d)isk or (t)ape
device; or a (f)tp, (n)fs or (h)ttp server.
Where are the install sets? f
```

La URL empleada desde donde se pueden obtener los paquetes para la instalación es:

```
ftp://ftp.openbsd.org/pub/OpenBSD/
```

2.10.6 Elección de los archivos de instalación

Ahora se escogen los archivos a instalar. El instalador mostrará en pantalla aquellos archivos que encuentre y lo único que se hace es especificar cuáles se instalarán. Todos los archivos excepto los del entorno gráfico X están preseleccionados, y como no es necesario un ambiente gráfico, sólo se deja en el mínimo indispensable requerido para el funcionamiento de *OpenBSD*, que considera a *base35.tar.gz*, *etc35.tar.gz* y *bsd*

```
The following sets are available. Enter a filename, 'all' to select
all the sets, or 'done'. You may de-select a set by prepending a '-'
to its name.

        [X] bsd
        [X] bsd.rd
        [X] base35.tgz
        [X] etc35.tgz
        [X] misc35.tgz
        [X] comp35.tgz
        [X] man35.tgz
        [ ] game35.tgz
        [ ] xbase35.tgz
        [ ] xshare35.tgz
        [ ] xfont35.tgz
        [ ] xserv35.tgz

File Name? (or 'done') [bsd.rd] done
```

Una vez que se escogen los archivos, el programa pregunta si realmente se quiere extraer estos archivos que al contestar afirmativamente procede a instalarlos.

Muestra una barra de progreso mediante la cual mantiene informado del tiempo que

tarda en instalar cada uno de los paquetes escogidos. El tiempo dependerá en gran medida de la máquina en la que se está instalando OpenBSD, de los archivos que se instalen y de la velocidad del medio de instalación.

```
Ready to install sets? [yes] Enter

100% |***************************************************|  4472 KB    00:03
Getting bsd.rd ...
100% |***************************************************|  4190 KB    00:02
Getting base35.tgz ...
100% |***************************************************| 30255 KB    00:21
Getting etc35.tgz ...
100% |***************************************************|  1469 KB    00:01
Getting misc35.tgz ...
100% |***************************************************|  1828 KB    00:01
Getting comp35.tgz ...
100% |***************************************************| 16207 KB    00:13
Getting man35.tgz ...
100% |***************************************************|  5921 KB    00:04

Sets can be located on a (m)ounted filesystem; a (c)drom, (d)isk or (t)ape
device; or a (f)tp, (n)fs or (h)ttp server.
Where are the install sets? (or 'done') Enter
```

2.10.7 Últimos pasos

En seguida se hacen un par de preguntas sobre la configuración del sistema instalado. La primera es si *sshd* será iniciado al arranque a lo cual se contesta afirmativamente.

```
Do you wish sshd(8) to be started by default? [yes] y
```

Ya sólo faltará introducir la franja horaria. Dependiendo de la región en la que se ubica el *firewall*, existen varias respuestas que pueden ser igualmente válidas.

```
Saving configuration files......done.
Generating initial host.random file ......done.
What timezone are you in? ('?' for list) [US/Pacific] Mexico
Setting local timezone to 'Mexico'...done.
```

Los últimos pasos del programa serán los de crear el directorio */dev*, que tardar un rato porque se dispone de poca memoria de RAM. Lo siguiente es la instalación de los bloques de arranque.

Finalmente lo último que se despliega en la pantalla es un mensaje informando que el sistema se ha instalado satisfactoriamente y que lo único necesario, es reiniciar la computadora.

```
CONGRATULATIONS! Your OpenBSD install has been successfully completed!
To boot the new system, enter halt at the command prompt. Once the
system has halted, reset the machine and boot from the disk.
# halt
syncing disks... done

The operating system has halted.
Please press any key to reboot.
```

En este momento el sistema ya está instalado y preparado para ser reiniciado y configurado para entrar en servicio.

Capítulo 3 Sistemas de ecuaciones lineales y métodos de solución

Una de las herramientas matemáticas que se utilizan en diferentes problemas de ingeniería son los sistemas de ecuaciones lineales, por que permiten representar las relaciones que pueden darse en procesos de producción, así como una forma de relacionar las restricciones inherentes.

Dado su utilidad, es muy conveniente entender que son los sistemas de ecuaciones lineales, y como se puede trabajar con éstos; para poder plantear soluciones a problemas que se pueden representar o reducir a sistemas de ecuaciones lineales.

3.1 Ecuaciones lineales

Una ecuación es una expresión matemática que consta de datos conocidos, y datos que resultan variantes o desconocidos, todos ellos relacionados mediante operaciones aritméticas y el cumplimiento de una igualdad y en donde las incógnitas se encuentran elevadas únicamente a la primera potencia.

Los símbolos empleados para representar los valores desconocidos son letras minúsculas del final del abecedario (x, y, z, w) generalmente y representan cualquier valor para el conjunto de números en donde se da la ecuación.

Formalmente una ecuación lineal sobre $\mathbb{R}$ es una expresión de la forma:

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = b$$

donde:

$$a_1, a_2, \cdots, a_n, b \in \mathbb{R}$$

Los símbolos $x_1, x_2, \cdots, x_n$ se les conoce como incógnitas de la ecuación; a los números a_i como coeficientes de las incógnitas y al número b como el término independiente.

Al tener una ecuación lineal, es posible determinar un valor para cada una de las incógnitas, de forma que se pueda respetar la igualdad, por ejemplo dada la siguiente ecuación:

$$-3.2\,x_1 + 2\,x_2 - 5.2\,x_3 = 20$$

al sustituir las incógnitas por los siguientes valores $x_1 = 1, x_2 = 2.5, x_3 = -3.5$ en la ecuación se tiene lo siguiente:

$$\begin{aligned} -3.2(1) + 2(2.5) - 5.2(-3.5) &= 20 \\ -3.2 + 5 + 18.20 &= 20 \\ -3.2 + 23.2 &= 20 \\ 20 &= 20 \end{aligned}$$

donde se aprecia que se cumple la igualdad. A estos valores con los cuales se han sustituido las incógnitas se le conoce como una solución de la ecuación, al lograr determinar un valor único para cada una de las incógnitas que respeta la igualdad.

Formalmente una solución de una ecuación lineal es el conjunto de n valores

$$k_1, k_2, \cdots, k_n \in \mathbb{R}$$

tales que al sustituir las incógnitas con los valores k_i satisfacen la igualdad:

$$a_1 k_1 + a_2 k_2 + \cdots + a_n k_n = b$$

Al determinar la solución para una ecuación lineal se pueden dar tres casos.

El primer caso se presenta cuando al menos uno de los coeficientes es diferente de cero. En este caso se da que $a_k \neq 0$, por esta razón la ecuación se puede reescribir como:

$$a_k x_k = b - a_1 x_1 - \cdots - a_{k-1} x_{k-1} - a_{k+1} x_{k+1} - \cdots - a_n x_n$$

despejando a x_k tenemos:

$$x_k = \frac{1}{a_k}(b - a_1 x_1 - \cdots - a_{k-1} x_{k-1} - a_{k+1} x_{k+1} - \cdots - a_n x_n)$$

En esta última expresión se pueden asignar valores arbitrarios a cada una de las siguientes incógnitas $x_1, \cdots, x_{k-1}, x_{k+1}, \cdots, x_n$ que al evaluar se puede determinar el valor de la última incógnita que es x_k , este conjunto de valores, tanto los asignados arbitrariamente como el último calculado, corresponden a una solución de la ecuación lineal.

Dado el ejemplo:

$$4x_1 - 6x_2 + 2x_3 = 10$$

Se puede realizar lo siguiente para definir una solución de la ecuación lineal.

Se asigna valores arbitrarios a las incógnitas $x_1 = a, x_2 = b$ y al sustituir estos valores

en la ecuación lineal y al despejar la incógnita x_3 tenemos:

$$x_3=\frac{1}{2}(10-4\mathrm{a}+6\mathrm{b})$$

que es equivalente a $x_3=5-2\mathrm{a}+3\mathrm{b}$ con el cual se determina un conjunto solución para el sistema de ecuaciones dado por $(a,b,5-2\mathrm{a}+3\mathrm{b})$, donde *a* y *b* pueden ser cualquier valor.

El segundo caso se da cuando todos los coeficientes y el término independiente es cero; en tal caso, cualquier valor del conjunto de números considerado puede satisfacer la relación.

Ejemplo:

$$0\mathrm{x}_1+0\mathrm{x}_2+\cdots+0\mathrm{x}_n=0$$

en donde se aprecia que cualquier valor $k_i\in\mathbb{R}$ puede satisfacer la igualdad.

El último caso se da cuando todos los coeficientes sean cero pero el término independiente es diferente de cero, en tal caso la ecuación es inconsistente y no tienen solución.

$$0\,x_1+0\,x_2+\cdots+0\,x_n=b$$

Y esto es debido a que no se cumple la igualdad, independientemente de los valores que tomen las incógnitas.

3.2 Sistemas de ecuaciones lineales

Un sistema es un grupo de ecuaciones lineales que tienen en común un número semejante de incógnitas además de existir alguna relación entre ellas que en algunos casos puedan representar las condiciones de un sistema físico.

Formalmente de forma general un sistema de ecuaciones lineales se define como:

$$\begin{array}{c} a_{11}x_1+a_{12}x_2+\cdots+a_{1n}x_n=b_1 \\ a_{21}x_1+a_{22}x_2+\cdots+a_{2n}x_n=b_2 \\ \vdots \quad \ddots \quad \vdots \\ a_{m1}x_1+a_{m2}x_2+\cdots+a_{mn}x_n=b_m \end{array}$$

en donde:

$$a_{11,}a_{12,}\cdots,a_{mn},b_1,\cdots,b_m \in \mathbb{R}$$

Dado que un sistema de ecuaciones lineales contiene las mismas incógnitas, una solución es un grupo de valores que al sustituirlos por las incógnitas satisfacen simultáneamente a todas las ecuaciones del sistema.

Por lo tanto una solución para el sistema:

$$\begin{array}{c} a_{11}x_1+a_{12}x_2+\cdots+a_{1n}x_n=b_1 \\ a_{21}x_1+a_{22}x_2+\cdots+a_{2n}x_n=b_2 \\ \vdots \quad \vdots \quad \ddots \quad \vdots \quad \vdots \\ a_{m1}x_1+a_{m2}x_2+\cdots+a_{mn}x_n=b_m \end{array}$$

es un conjunto ordenado de n valores $k_{1,}k_{2,}\cdots,k_n$ con $k_i \in \mathbb{R}$; tales que:

$$\begin{array}{c} a_{11}k_1+a_{12}k_2+\cdots+a_{1n}k_n=b_1 \\ a_{21}k_1+a_{22}k_2+\cdots+a_{2n}k_n=b_2 \\ \vdots \quad \vdots \quad \ddots \quad \vdots \quad \vdots \\ a_{m1}k_1+a_{m2}k_2+\cdots+a_{mn}k_n=b_m \end{array}$$

Como se aprecia en la definición anterior para una solución, un valor k_i determinado para una incógnita x_i dada, debe reemplazarse en todas las ocurrencias de la variable x_i , por ejemplo si para un sistema dado se determina que el valor de la incógnita x_3 de la primera ecuación es 4.5, debe sustituirse ese valor en todas las ecuaciones donde aparezca x_3 .

Trabajando con un ejemplo sencillo como el siguiente:

$$\begin{array}{c} 3x_1+2x_2-2x_3=-2 \\ 4x_1+3x_2-x_3=-2 \\ 2x_1+x_2+2x_3=3 \end{array}$$

El conjunto solución es $x_1=2, x_2=-3, x_3=1$, que al sustituir los valores por las incógnitas en el sistema anterior, se tiene:

$$\begin{array}{c} 3(2)+2(-3)-2(1)=-2 \\ 4(2)+3(-3)-1=-2 \\ 2(2)+(-3)+2(1)=3 \end{array}$$

reduciendo operaciones:

$$\begin{array}{c} 6-6-2=-2 \\ 8-9-1=-2 \\ 4-3+2=3 \end{array}$$

comprobando que se cumplen con las igualdades en todas las ecuaciones.

Lo anterior define aquello que cumple con ser una solución para un sistema de

ecuaciones lineales, pero no garantiza que exista siempre una única solución.

Puede presentarse varios casos generales de soluciones de acuerdo a la naturaleza del sistema de ecuaciones lineales.

Hay sistemas para los cuales no existe solución, como puede ser el caso del siguiente sistema:

$$\begin{aligned} -x_1 - 3x_2 &= 4 \\ 2x_1 + 6x_2 &= 5 \end{aligned}$$

para el cual no existe un par de números que satisfagan simultáneamente al sistema.

Un sistema que resulta más evidente el hecho de no tener solución es el siguiente:

$$\begin{aligned} x_1 - x_2 &= 5 \\ x_1 - x_2 &= 3 \end{aligned}$$

en donde se observa que no existe un par de números que al restarse dé 5 y 3 a la vez.

A estos sistemas que no tienen solución se les llama incompatibles o inconsistentes.

Como contraparte, a los sistemas lineales que sí tienen solución, se les llama compatibles ó consistentes.

Éstos a su vez pueden tener una única solución, a los que se les llama determinados, o pueden tener varias soluciones en cuyo caso se les llama indeterminados.

Ya se ha mostrado tanto sistemas compatibles determinados como sistemas incompatibles, por lo que solo queda mostrar un sistema compatible indeterminado:

$$x_1 + 2x_2 - x_3 = -3$$
$$2x_1 - x_2 + 3x_3 = 4$$

donde se observa que el conjunto solución es:

$$x_1 = k$$
$$x_2 = -k - 1 \quad \text{con} \quad k \in \mathbb{R} \ .$$
$$x_3 = -k + 1$$

En este ejemplo, se puede dar un valor arbitrario a k , con lo que x_2 y x_3 quedan determinados en función de este valor; como k puede ser un valor en $\mathbb{R}$ cualquiera, existen varias soluciones para el sistema de ecuaciones lineales anterior.

Cuando existen dos sistemas de ecuaciones lineales que tienen la misma solución, se les llama sistemas equivalentes

Generalmente para que un sistema de ecuaciones lineales sea compatible determinado, debe tener el mismo número de ecuaciones que de incógnitas y que las ecuaciones que conforman al sistema, sean linealmente independientes.

Como se mencionó, de acuerdo a su posible solución existen diferentes tipos de sistemas de ecuaciones lineales, pero para el presente trabajo, los sistemas que interesan son los sistema compatibles determinados; aquellos a los que se les puede determinar una solución única.

3.2.1 Transformaciones elementales

Las transformaciones elementales son aquellas que al aplicarse a un sistema de ecuaciones lineales, da como resultado un sistema equivalente.

Las transformaciones elementales consisten en:

1. El intercambio de dos ecuaciones
2. El multiplicar una ecuación por un número diferente de cero
3. Multiplicar una ecuación por un número y sumarla a otra ecuación, reemplazándola por el resultado obtenido

Para ejemplificar el primer caso, dada el siguiente sistema:

$$\begin{aligned} x_1+x_2+x_3&=-2 \\ -x_1+x_2-2x_3&=4 \\ 2x_1-2x_2+4x_3&=6 \end{aligned}$$

es equivalente al siguiente sistema

$$\begin{aligned} -x_1+x_2-2x_3&=4 \\ x_1+x_2+x_3&=-2 \\ 2x_1-2x_2+4x_3&=6 \end{aligned}$$

al que se le han intercambiado el renglón uno con el dos y viceversa.

Para el segundo caso, se multiplica el tercer renglón del sistema anterior por $\alpha=\frac{1}{2}$:

$$\begin{aligned} -x_1+x_2-2x_3&=4 \\ x_1+x_2+x_3&=-2 \\ \alpha(2x_1-2x_2+4x_3&=6) \end{aligned}$$

obteniendo el siguiente sistema equivalente:

$$\begin{aligned} -x_1 + x_2 - 2x_3 &= 4 \\ x_1 + x_2 + x_3 &= -2 \\ x_1 - x_2 + 2x_3 &= 3 \end{aligned}$$

Para el caso tres, dado el sistema:

$$\begin{aligned} 3x_1 + 2x_2 - 2x_3 &= -2 \\ 4x_1 + 3x_2 - x_3 &= -2 \\ 2x_1 + x_2 + 2x_3 &= 3 \end{aligned}$$

se multiplica el tercer renglón por -2

$$\begin{aligned} 3x_1 + 2x_2 - 2x_3 &= -2 \\ 4x_1 + 3x_2 - x_3 &= -2 \\ -4x_1 - 2x_2 - 4x_3 &= -6 \end{aligned} \quad ,$$

y se suma al segundo renglón dando como resultado el siguiente sistema equivalente:

$$\begin{aligned} 3x_1 + 2x_2 - 2x_3 &= -2 \\ x_2 - 5x_3 &= -8 \\ 2x_1 + x_2 + 2x_3 &= 3 \end{aligned}$$

3.3 Matrices

Una matriz es un conjunto ordenado de elementos que tienen algún rasgo o característica en común entre ellos. En matemáticas, los elementos generalmente pertenecen a un conjunto de números como puede ser el de los $\mathbb{R}$, y están dispuestas de forma idéntica a una tabla:

$$\begin{bmatrix} 4.8 & 3.5 & -7.2 & -34.6 & 17.3 \\ -7.5 & 9.83 & 4.8 & -3.5 & 13.2 \\ -34.6 & 27.3 & 21.7 & 16.8 & -3.4 \\ 7.26 & -9.3 & 12.52 & -1.8 & 6.11 \\ 5.63 & 2.13 & -7.1 & 6.18 & 1.74 \end{bmatrix}$$

De forma general se puede decir que una matriz de $m \times n$ con elementos en $\mathbb{R}$ es un arreglo de la forma:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

donde $a_{11}, a_{12}, \cdots, a_{mn} \in \mathbb{R}$, $m, n \in \mathbb{N}$

Se le denomina orden al número de renglones y columnas que conforman a la matriz; así $m \times n$ significa que se tiene una matriz de m renglones y n columnas.

De forma abreviada una matriz se puede representar como:

$[a_{ij}]$, donde $i = 1, 2, \cdots, m$; y $j = 1, 2, \cdots, n$

3.3.1 Multiplicación de un escalar por una matriz

La multiplicación de un escalar representado por letras minúsculas, por una matriz simplemente indica que el escalar se multiplica por cada una de los elementos que conforma la matriz sin alterar la dimensión. Así se tiene lo siguiente:

Dada una matriz $A = [a_{ij}] de m \times n$ con elementos en $\mathbb{R}$ y un escalar α también en

$\mathbb{R}$, la multiplicación de la matriz por el escalar entonces queda dada como $E=\left[e_{ij}\right] de\ m\times n$ donde $e_{ij}=\alpha\cdot a_{ij}, i=1,\cdots,m\ y\ j=1,\cdots,n$

Por ejemplo teniendo la siguiente matriz y el siguiente escalar:

$$A=\begin{bmatrix}3 & -2 & 4\\ 5 & 7 & -8\end{bmatrix}\ ,\quad \alpha=-4$$

El resultado de multiplicar el escalar por la matriz es:

$$\alpha A=\begin{bmatrix}-12 & 8 & 16\\ -20 & -28 & 32\end{bmatrix}$$

3.3.2 Multiplicación entre matrices

Para que las matrices puedan multiplicarse estas deben ser conformables para la multiplicación, es decir que el número de columnas de la primera debe ser igual que el número de renglones de la segunda; el orden de la matriz resultante será igual al número de renglones de la primera e igual al número de columnas de la segunda.

Básicamente lo que se realiza, es una multiplicación de los elementos renglón de la primera matriz, por los elementos columna de la segunda matriz; si nos encontramos en el renglón i-ésimo y en la columna j-ésima, se está conformado el elemento c_{ij} de la matriz resultante de la multiplicación.

Para obtener el valor c_{ij} se suma el resultado de multiplicar elemento a elemento del i-ésimo renglón por la j-ésima columna. Por ejemplo, dadas las siguientes matrices:

$$A=\begin{bmatrix} 2 & 4 \\ -6 & -1 \\ 7 & 3 \end{bmatrix},\ B=\begin{bmatrix} 4 & -5 \\ -9 & 8 \end{bmatrix}$$

el resultado de multiplicar A por B es:

$$\begin{aligned} c_{11}&=2\cdot 4+(4\cdot -9)=-28 \\ c_{21}&=(-6\cdot 4)+(-1\cdot -9)=-15 \\ c_{31}&=7\cdot 4+(3\cdot -9)=1 \\ c_{12}&=(2\cdot -5)+4\cdot 8=22 \\ c_{22}&=(-6\cdot -5)+(-1\cdot 8)=22 \\ c_{32}&=(7\cdot -5)+3\cdot 8=-11 \end{aligned}$$

, es decir: $C=\begin{bmatrix} -28 & 22 \\ -15 & 22 \\ 1 & -11 \end{bmatrix}$

De manera general, si la matriz A es $m\times n$ y la matriz B es de $n\times q$, la matriz C resultado de multiplicar A por B será de orden $m\times q$.

La definición formal para la multiplicación de matrices se dado de la siguiente forma:

Sean $A=[a_{ij}]$ y $B=[b_{ij}]$ dos matrices con elementos en $\mathbb{R}$, de $m\times n$ y $n\times q$ respectivamente. El producto AB es una matriz , de $m\times q$ definida por :

$$P_{ij}=\sum_{k=1}^{n} a_{ik}b_{kj}$$

para $i=1,\cdots,m$ y $j=1,\cdots,q$

3.3.3 Matriz identidad

Se define como matriz identidad a una matriz cuadrada de orden n constituida de 1's en su diagonal principal y ceros en el resto de ella, es decir una matriz de la forma:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Formalmente de manera general se llama matriz identidad de orden n a la matriz cuadrada:

$$I_n = [\delta_{ij}] \text{ , tal que: } \delta_{ij} = 1, si \;\; i = j \quad \text{y} \quad \delta_{ij} = 0, si \;\; i \neq j$$

3.3.4 Transposición

Es una operación que se realiza sobre una matriz para transformarla en otra matriz llamada transpuesta, que consiste en intercambiar renglones por columnas de la matriz original, consecuentemente las columnas de la matriz terminan siendo los renglones de la matriz transpuesta. Por ejemplo, dada la siguiente matriz:

$$A = \begin{bmatrix} 1 & 2 & -5 \\ -3 & 9 & -4 \end{bmatrix} \quad \text{la transpuesta de A es:} \quad A^T = \begin{bmatrix} 1 & -3 \\ 2 & 9 \\ -5 & -4 \end{bmatrix}$$

Formalizando, se define la matriz transpuesta a partir de:

Sea $A = [a_{ij}]$ una matriz de $m \times n$ con elementos en $\mathbb{R}$. Se llama transpuesta de A a la matriz de orden $n \times m$ $A^T = [c_{ij}]$ tal que $c_{ij} = a_{ji}$.

3.3.5 Matriz dispersa

Se le denomina a una matriz A como dispersa, cuando una gran cantidad de los elementos que la conforman son ceros.

Esto es porque en una matriz, los elementos ceros simplifican mucho las operaciones matriciales. Dado que resulta más sencillo realizar cualquier operación aritmética cuando ésta involucra ceros.

Una matriz diagonal es una matriz dispersa, aunque no todas las matrices dispersas son matriz diagonal, porque los elementos diferentes de cero pueden estar ubicados en cualquier posición dentro de la matriz.

Ejemplos de matrices dispersas:

$$\begin{bmatrix} 1.4 & 1.9 & 0 & 0 & 0 & 0 \\ 1.5 & 1.1 & 0.9 & 0 & 0 & 0 \\ 0 & 0.8 & 1.9 & 1.7 & 0 & 0 \\ 0 & 0 & 1.2 & 1.4 & 1.1 & 0 \\ 0 & 0 & 0 & 0.9 & 1.7 & 1.3 \\ 0 & 0 & 0 & 0 & 0.7 & 1.6 \end{bmatrix}, \quad \begin{bmatrix} 5 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 7 & 4 & 0 & 0 \\ 0 & 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 \\ 0 & 9 & 0 & 0 & 8 & 0 \end{bmatrix}$$

Una matriz cuadrada puede ser dispersa como se aprecia en los ejemplos anteriores, pero también puede ser dispersa una matriz rectangular.

3.4 Representación matricial de los sistemas de ecuaciones lineales

Las matrices son entes matemáticos que resultan útiles para representar sistemas de ecuaciones lineales.

Si tenemos un sistema de ecuaciones lineales como el siguiente:

$$x_1 - 2x_2 + x_3 = 7$$
$$2x_1 + 3x_2 - x_3 = -1$$
$$x_1 + 3x_2 - 2x_3 = -8$$

Se puede representar mediante matrices como $Ax = b$, es decir:

$$\begin{bmatrix} 1 & -2 & 1 \\ 2 & 3 & -1 \\ 1 & 3 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ -1 \\ 8 \end{bmatrix}$$

Se observar que si la matriz de coeficientes logra convertirse en la matriz identidad, al multiplicarla por la matriz de incógnitas, se puede determinar la solución del sistema. Es claro además, que los cambios realizados a la matriz de coeficientes deben realizarse también en el vector de términos independientes.

3.5 Métodos de solución

Dado la importancia de lo que puede representar un sistema de ecuaciones lineales, se han creado diferentes métodos para calcular su solución, cuando ésta existe y es única.

Típicamente se puede hablar de existen dos grupos de métodos de solución: los métodos directos y los métodos iterativos.

Un método directo, es aquel que a partir de realizar transformaciones elementales u operaciones matriciales logra despejar cada una de las incógnitas para determinar su valor dentro del sistema. El más representativo es la eliminación Gaussiana.

Un método iterativo es aquel que genera soluciones aproximadas de manera iterativa a partir de una estimación inicial, valiéndose de los pasos anteriores para mejorar la solución siguiente, es decir que la solución x aproximada, depende de la solución x-1.

Los métodos directos garantizan que se calculará la solución exacta del sistema, el inconveniente que presentan es que son muy costosos tanto en tiempo como en recursos computacionales.

Cuando se tienen sistemas muy grandes, el uso de un método directo resulta inviable, por ello es conveniente buscar otra forma de obtener la solución.
Los métodos iterativos, tienen la ventaja de que pueden obtener una solución aproximada en un tiempo mucho menor al que requiere un método directo. Solución que muchas veces es aceptable para ser usado en el sistema de ecuaciones lineales.

El inconveniente que puede presentarse, es que al ser métodos iterativos, necesitan condiciones de convergencia, las cuales pueden o no darse en casos muy particulares.

De entre los métodos iterativos se encuentran los métodos de Krylov.

3.5.1 Métodos de Krylov

Debido a la dimensión de los sistemas a resolver, el proceso de eliminación gaussiana tiene costo computacional muy elevado aún en aquellas variantes específicas de sistemas dispersos. Ante esta situación es recomendable el uso de métodos iterativos para obtener la solución del sistema; sin embargo las técnicas usuales como los métodos de tipo SOR[10], resultan poco efectivos debido a las limitaciones teóricas para la convergencia de dichos métodos.

Entre los algoritmos iterativos más eficientes se encuentran los métodos de Krylov, los cuales han sido objetos de estudio en diversas áreas de la matemática aplicada.

10 SOR: el método Successive Overrelaxation o de relajación.

La idea básica éstos métodos consiste en buscar una aproximación a la solución del sistema de ecuaciones, como una combinación lineal de los vectores $v, Av, A^2v, \cdots, A^{k-1}v$, donde k, es mucho menor que la dimensión del sistema.

El subespacio generado por dichos vectores es llamado el subespacio de Krylov de dimensión k, asociado al vector v y a la matriz A y se denota por $K_k(A, v)$ o simplemente K_k .

Entre los métodos de Krylov, los más representativos son el gradiente conjugado y GMRES.

3.5.1.1 Gradiente Conjugado

Uno de los métodos mas utilizados debido a su simplicidad, eficiencia y bajo costo computacional, es el método de Gradiente Conjugado (CG), dado que en cada iteración de CG se resuelve el problema:

$$\min_{x x_0 + K_k} \|x - x_E\|_A$$

Donde x_E es la solución exacta, x_0 es una aproximación inicial y $\|V\|_A := \sqrt{\langle v, Av \rangle}$ con $\langle , \rangle$ como el producto escalar usual.

El algoritmo es el siguiente:

Dados A, b, x_0

$r_0 = b - Ax_0, v_0 = r_0$

Para $k = 1, 2, \cdots$

Determinar x_k solución de $\min_{x x_0+K_k} \|x-x_E\|_A$

Si ($\|b-Ax_k\|_2<tol$) Salir

Construir $v_k K_{k+1}$ tal que $\langle v_i, v_j\rangle=0$ para $j=0,\cdots,k-1$

FIN

Desafortunadamente este algoritmo requiere que la matriz de coeficientes sea simétrica, es decir que $A=A^t$ y la convergencia solo es garantizada si A es definida positiva $xAx^t>0\, x\in\mathbb{R}^n$

3.5.1.2 GMRES

El método de Residuo Mínimo Generalizado (GMRES), es uno de los algoritmos más representativos de los métodos de Krylov y uno de los más eficientes.

En cada aproximación de GMRES, al igual que en gradiente conjugado, se resuelve un problema de minimización, sin embargo, en este caso se busca reducir la norma del residual asociado a cada aproximación, así, en cada paso, GMRES resuelve el problema:

$$\min_{x x_0+K_k} \|b-Ax\|_2$$

Donde x_0 es la aproximación inicial y $\|v\|_2:=\sqrt{\langle v,v\rangle}$ con $\langle,\rangle$ el producto escalar usual.

Es importante notar que si $n=k$ entonces $x_k=k_E$ es decir, la solución exacta será alcanzada en a lo mas n iteraciones.

3.6 Algoritmo de descomposición de dominios de Schwarz para sistemas lineales

Aunque la descomposición de dominios es un método matemático complejo, utilizado para resolver problemas que involucran ecuaciones diferenciales parciales y una notación mucho más formal, se explica el proceso reduciendo lo más posible los elementos complejos del problema.

Básicamente la idea parte de que existen dos dominios limitados por alguna forma simple y entre los dominios existe un traslape, con lo que en su conjunto el dominio principal se considera de la unión de ambos dominios.

Grosso modo, si es posible resolver el problema en cada uno de los subdominios, y se encuentra una forma de intercambiar información entre éstos, entonces se considera que la unión de la solución de ambos subdominios, corresponde a la solución del dominio completo.

La idea antes descrita se puede extender a la solución de sistemas de ecuaciones lineales, dado que el algoritmo de descomposición de dominios de *Schwarz* es un proceso que permite calcular una aproximación de la solución del sistema $Ax=b$ a partir de las soluciones de los subsistemas de menor dimensión $A_1 x_1 = b_1$ y $A_2 x_2 = b_2$, en donde cada subsistema se le denomina como un dominio del problema original, considerando que A_1 y A_2 son submatrices de A y los sistemas $A_1 x_1 = b_1$ y $A_2 x_2 = b_2$ incluyen a todas las incógnitas del sistema original.

El algoritmo es el siguiente:

Dados A, b, x_0

Construir A_i, r_i

Hasta cumplir convergencia

Calcular $r = b - Ax_0$

Para i=1, hasta NumeroDominios

Calcular $e_i = A_i^{-1} r_i$

Construir e_0 *a partir de* e_i

$x_0 = x_0 + e_0$

Fin

Fin

3.7 Consideraciones

El conjunto numérico de interés para este trabajo es el de los números en $\mathbb{R}$, dado la naturaleza y origen del problema que se desea resolver, pero es importante notar que tanto las ecuaciones como las matrices pueden estar conformados por números pertenecientes al conjunto de los $\mathbb{C}$.

Por otro lado, dado que la representación de números en una computadora es limitada, agrega errores de cálculo a toda operación realizada con números reales; en este sentido aunque teóricamente un método directo da la solución exacta de un sistema de ecuaciones, en la práctica puede no ser así.

Esta consideración también hay que tomarse en cuenta cuando se usan métodos iterativos, dado que es un error que es inherente a la herramienta de trabajo, es por ello que se consideran criterios de paro, y una tolerancia mínima para considera a los resultados obtenidos como válidos.

Capítulo 4 Programación en paralelo

Existen varias formas de programación en paralelo de acuerdo al tipo de computadora de la que se disponga, desde instrucciones en ensamblador, hasta complejos compiladores que realizan el proceso de paralelización de manera automática.

Dado el tipo de máquina que se tiene, el *cluster Beowulf,* el método empleado en este trabajo es el esquema de paso de mensajes basado en MPI (Message Pasing Interface).

4.1 Esquema de paso de mensajes

Considerando que un *cluster* es una máquina paralela de memoria distribuida, los datos solo son accesibles por el nodo en donde se encuentran almacenados, a menos que éstos sean extraídos de archivos compartidos.

Entonces la manera en como se comparte información entre los nodos es mediante mensajes, en donde cada uno de éstos es un paquete que contiene generalmente además de la intención del mensaje, información para compartir, la cual puede ser instrucciones o datos.

Las instrucciones pueden indicar que tipo de acción se desea hacer entre los nodos, como la coordinación de tareas, o la distribución de datos, de forma masiva o solo entre algunos de los nodos, de manera coordinada o desincronizada.

Entre las herramientas de paso de mensajes, existen principalmente dos: PVM y MPI. PVM fue el primero en aparecer y se empleó durante mucho tiempo cuando se tenían clusters heterogéneos, aunque actualmente ya esta entrando en desuso; MPI por su

parte es más reciente y se buscó en un principio, que permitiera trabajar en clusters homogéneos, aunque su aceptación y desarrollo ha logrado que se convierta en una herramienta ampliamente utilizada en clusters de alto rendimiento.

4.2 Empleando mpich para la paralelización

MPI (Message Passing Interface) es un estándar desarrollado por el MPIF (Message Passing Interface Forum). Especifica una interfaz portable para la escritura de programas basados en paso de mensajes, que además resulte práctica, eficiente y flexible al mismo tiempo.

La biblioteca empleada en el desarrollo del *solver* paralelo es *mpich*, una implementación libre del estándar MPI, que provee de toda una serie de herramientas para realizar la programación en paralelo de una forma más simple. Esto lo logra mpich al proveer una biblioteca con funciones que permiten manipular envíos y recepciones de paquetes a alto nivel, dejando de lado elementos de manipulación de hardware; además de poner a disposición interfaces para el llamado de cada uno de los compiladores que soporta, permitiendo una invocación sencilla y eliminando lo tedioso de un llamado directo.

También mpich proporciona un ambiente de ejecución donde se corren los programas paralelos hechos, que además permite sincronizar las comunicaciones entre las máquinas involucradas. Todo ello permite concentrarse más en la programación del *solver*, y desentenderse lo más posible de detalles de programación de bajo nivel, como es el empaquetado de los datos para ser enviados a través de la tarjeta de red o la programación de sockets por ejemplo.

La biblioteca de mpich provee definiciones propias de los tipos de datos base de los lenguajes soportados, para homogeneizar la manera en como se intercambian los datos

entre los diferentes nodos.

Aunque hay un tipo mpi para cada uno de los tipos nativos de cada uno de los lenguajes soportados, como el *solver* fue desarrollado en Fortran 90, los tipos de datos MPI usados se reducen a: MPI_INTEGER y MPI_DOUBLE_PRECISION

Las interfaces de mpich para los compiladores Fortran 77, 90, 95, C y C++, son los nombres con los que habitualmente se invocan precedidos por mpi, teniendo así: mpif90, mpif77, mpicc, mpicpp.

Esto resulta muy práctico al reducir en un comando, todos los modificadores y banderas que de otra forma habría que pasarle a cada uno de los compiladores como parámetros en la línea de comandos.

En mpich existen varios grupos de instrucciones, según la función que desempeñan:

- Las funciones de inicialización y terminación del ambiente.
- Las funciones de sincronización.
- Las funciones de envió y recepción de mensajes
- Las funciones de reducción de datos

Aunque mpich proporciona una gran cantidad de instrucciones (en su versión 1 más de 100), se puede hacer programas paralelos usando muy pocas.

Aunque ya existe las especificaciones para MPI 2 así como de implementaciones de esta nueva versión, se decidió trabajar con la anterior, la 1.2.

4.2.1 Sobre el uso de las subrutinas de mpich

En Fortran, el código regresado para todas las subrutinas MPI se da en el último argumento. Se devuelve cero si la subrutina termina exitosamente.

4.2.2 Subrutinas para la administración del ambiente

Este grupo está conformado por aquellas subrutinas que permiten iniciar el ambiente MPI para que funcione adecuadamente el paso de mensajes. Las subrutinas son:

- MPI_INIT(ierr): Inicializa el ambiente MPI. Debe llamarse solamente una vez y antes de llamar a cualquier otra función.
- MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr): Regresa el número de los procesos que pertenecen al comunicador especificado en el primer argumento. Como se usa MPI_COMM_WORLD, nprocs tendrá el total de procesos.
- MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr) : Regresa un identificador de proceso, asignado dentro del comunicador que es dado como primer argumento.
- MPI_FINALIZE(ierr): Termina el proceso MPI, por lo que después ya no se pueden hacer más llamadas a subrutinas MPI.

Un comunicador es un identificador asociado a un grupo de procesos. En el caso de MPI_COMM_WORLD, está definido en el módulo MPI y representa al grupo constituido por todos los procesos participantes en el trabajo paralelo.

Las subrutinas anteriores deben estar en todo programa hecho para trabajar con mpi. En el caso de MPI_INIT y de MPI_FINALIZE, siempre deben aparecer y únicamente se llaman una sola vez en todo el programa.

4.2.3 Subrutinas de comunicaciones punto a punto

Son subrutinas que permiten la comunicación entre un par de nodos específicos.

En general cuando se hace uso de estas subrutinas suceden varios pasos en la comunicación cuando se manda un mensaje de un proceso 0 a un proceso 1.

Cuando se manda un mensaje los pasos son:

- El dato se copia a un *buffer* por el usuario
- El usuario llama a una de las rutinas MPI de envío.
- El sistema copia el dato desde el buffer de usuario al buffer del sistema
- El sistema envía el dato desde el buffer del sistema al proceso destino

Cuando se recibe un mensaje sucede lo siguiente:

- El usuario llama una de las subrutinas MPI de recepción
- El sistema recibe el dato desde el proceso origen y lo copia en el buffer del sistema
- El sistema copia el dato desde el buffer del sistema al buffer del usuario
- El usuario puede emplear el dato recibido

De las subrutinas punto a punto, existen dos grupos de subrutinas, las que son bloqueantes y las que no lo son.

Cuando se usan subrutinas bloqueantes, el programa no regresa de la subrutina llamada hasta que las copias respectivas de los datos se han efectuado completamente.

En el caso de subrutinas no bloqueantes, inmediatamente se regresa de la rutina después de la llamada. Aquí solo se ha realizado la copia del o desde el buffer del sistema, y no se garantiza que la copia se efectué completamente.

Se utilizan las subrutinas bloqueantes porque se requiere la garantía de que los datos se copian satisfactoriamente.

Las subrutinas más representativas son:

- MPI_Send: subrutina bloqueante empleada para enviar datos
- MPI_Recv: subrutina bloqueante empleada para recibir datos

4.2.4 Subrutinas para comunicaciones colectivas

La biblioteca de MPI provee una serie de subrutinas para realizar comunicaciones que involucren a todos los nodos o al menos a un grupo de ellos. Estas subrutinas se utilizan para agrupar datos o distribuir datos entre todos los nodos.

Las subrutinas son:

- MPI_BCAST: un dato se distribuye a todos los nodos
- MPI_SCATTER: grupos de datos se distribuyen exclusivamente a nodos establecidos
- MPI_GATHER: se recolecta segmentos de datos, todos diferente de cada nodo
- MPI_ALLGATHER: con grupos de datos exclusivos de cada nodo, se distribuyen éstos en todos los nodos
- MPI_REDUCE se toman los datos de todos los nodos, y se simplifican mediante la realización de una operación
- MPI_ALLREDUCE igual que en la subrutina anterior, pero el resultado se distribuye en todos los nodos

Existen además otra subrutinas que no envía ni recibe datos como la subrutina MPI_BARRIER

4.2.5 Tipos de datos mpich para Fortran

Se muestra en la siguiente tabla, los tipos de datos MPI que son soportados en Fortran, y su descripción para identificarlos dentro de los tipos nativos.

Tipo MPI	**Descripción**
MPI_CHARACTER	caracter
MPI_REAL	Real simple precisión
MPI_INTEGER	Entero
MPI_LOGICAL	Booleano
MPI_DOUBLE_PRECISION	Real de doble precisión
MPI_COMPLEX	Complejo
MPI_DOUBLE_COMPLEX	Equivale a complex*16 (o complex*32) si está soportado

Tabla 1: Tipos de datos MPI

4.2.6 Consideraciones para el uso de mpich

Como se mencionó al principio, existe una gran cantidad de subrutinas MPI, sin embargo, es viable realizar un programa paralelo empleando un reducido número de ellas. Como no se pretende ser exhaustivo en todo lo referente a MPI, solo se han mencionado aquellas subrutinas y datos que se emplean en el desarrollo del *solver* paralelo.

- **Subrutinas para la administración del ambiente**

  Necesarias en cualquier programa que use MPI, se utilizan:
  MPI_INIT(ierr)

MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)
MPI_FINALIZE(ierr)

- **Comunicación punto a punto**

Para enviar datos entre nodos, se utilizan:
MPI_SEND(tmp_b_n, 1, MPI_INTEGER, i, 0, MPI_COMM_WORLD, ierr)
MPI_RECV(tmp_a_n, 1, MPI_INTEGER, MPI_A

- **Comunicaciones colectivas**

Empleadas para enviar y recibir información desde y hacia todos los nodos, y sincronizar los nodos:
MPI_BARRIER(MPI_COMM_WORLD, ierr)
MPI_BCAST(res, aa_n, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
MPI_GATHER(et_g, block_size, MPI_DOUBLE_PRECISION, et_buffer, block_size, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)

Estas son todas las funciones empleadas en el desarrollo del *solver* paralelo.

4.3 El solver en paralelo

Se emplea el algoritmo GMRES como el *solver* iterativo principal.

El algoritmo por si mismo es muy eficiente y logra converger en muy poco tiempo, pero depende de un precondicionador, para poder solucionar el sistema lineal que se le entregue. Por lo tanto la velocidad a la que se entregue la solución del sistema, depende de la capacidad de convergencia del *solver* así como de la rapidez con la que

se trabaje el precondicionador.

Otra limitante del *solver*, además de la velocidad de convergencia, está limitada por la cantidad de memoria de la computadora, por lo que se espera con la paralelización, atacar dos inconvenientes, acelerar la velocidad de entrega de solución y permitir resolver sistemas mucho más grandes de lo que es posible resolver en una sola computadora.

Como el tipo de sistemas lineales que se pretende resolver son muy grandes y dispersos, es posible emplear un formato que reduzca al máximo el uso del espacio en memoria, almacenando exclusivamente aquellos valores diferentes de cero. Sin embargo, como el sistema lineal no es el único componente que se encuentra en memoria, se tiene un límite en el tamaño de sistemas que se pueden almacenar en una sola computadora.

De manera implícita, se emplea la descomposición de dominios de *Schwarz*, para distribuir los datos en los nodos y resolver los sistemas.

4.4 Representando la matriz de coeficientes

Es necesario de alguna forma representar el sistema de ecuaciones lineales que se desea resolver, lo que implica emplear alguna estructura válida para la computadora que a su vez se pueda manipularse como una matriz.

4.4.1 Como matriz densa

Representar una matriz en una computadora puede realizarse de manera directa como matriz densa, empleando un arreglo (array) de datos, como por ejemplo la siguiente

matriz:

$$\begin{bmatrix} 4.8 & 3.5 & -7.2 & -34.6 & 17.3 \\ -7.5 & 9.83 & 4.8 & -3.5 & 13.2 \\ -34.6 & 27.3 & 21.7 & 16.8 & -3.4 \\ 7.26 & -9.3 & 12.52 & -1.8 & 6.11 \\ 5.63 & 2.13 & -7.1 & 6.18 & 1.74 \end{bmatrix}$$

empleando el lenguaje C se puede representar como:

```
float matriz[5][5]={{4.8,   3.5,  -7.2, -34.6, 17.3},
                    {-7.5,   9.83,  4.8,  -3.5, 13.2},
                   {-34.6,  27.3,  21.7,  16.8, -3.4},
                     {7.26, -9.3,  12.52, -1.8,  6.11},
                     {5.63,  2.13, -7.1,   6.18, 1.74}};
```

y en Fortran se puede hacer:

```
REAL(KIND=DP), DIMENSION(5:5) :: matriz
matriz = RESHAPE ((/4.8,   3.5,  -7.2, -34.6, 17.3,
                   -7.5,   9.83,  4.8,  -3.5, 13.2,
                  -34.6,  27.3,  21.7,  16.8, -3.4,
                    7.26, -9.3,  12.52, -1.8,  6.11,
                    5.63,  2.13, -7.1,   6.18, 1.74/),(/5,5/))
```

lo cual ocupa $25 \times tamaño-de-la-variable-en-RAM$ de casillas, que en el caso de lenguaje C, esa representación ocupa 100 bytes, mientras que la de Fortran 90 ocupa 200 bytes.

Para una matriz pequeña, esto no tiene mayor problema porque cualquier computadora moderna tiene suficiente memoria para almacenar estos datos. Pero es claro el rápido crecimiento del espacio consumido en cuanto el tamaño de la matriz se incrementa,

básicamente es $(TamañoDePalabra)\times n^2$, donde *n* es el orden del sistema de ecuaciones lineales.

Así para un sistema lineal de 70,000 variables, su representación como matriz densa ocupa:

$4\times(70{,}000)^2=1.96\times10^{10}$ bytes

empleando simple precisión, recordando que generalmente 4 es el número de bytes que usa un float. Es decir que sería necesario contar con 19.6 GB de RAM para poder almacenarla, lo que resulta inviable, aún con lo relativamente sencillo que es conseguir una gran cantidad de memoria de este tipo.

Esto no tiene sentido cuando menos del 1% de los elementos que conforman la matriz son diferentes de cero, por lo que se requiere una representación diferente de la matriz, que solo almacene estos elementos ya sean negativos o positivos.

Por ejemplo para un sistema de orden 35,084 la cantidad de elementos diferentes de cero es de 739,812, que representa aproximadamente el 0.06 % de los elementos totales de la matriz.

4.4.2 Formato para matrices dispersas

Existen más de veinte diferentes formatos de almacenamiento para representar matrices dispersas, en donde cada uno de éstos “explota” características particulares de las matrices que se presentan en varias áreas para ganar eficacia tanto en el uso de memoria como en el tiempo de procesamiento.

Erisman Duff y Reid en 1986 y posteriormente Saad en 1994 propusieron varios

esquemas de almacenamiento para este tipo de matrices.

Algunos de los esquemas más difundidos son: CSR, CSC, MSR, BCRS, CDS, JDS y SKS.

Se usa el formato CSR para representar la matriz de coeficientes porque es relativamente fácil de entender, de implementar y de manipular para realizar las diferentes operaciones matriciales necesarias.

4.4.3 Formato CSR para matrices dispersas

CSR es un formato estándar de almacenamiento por filas (CSR es Compressed Row Storage), que utiliza al menos un escalar y tres vectores para el almacenado de la matriz. Los nombres de los vectores habitualmente son: IA, JA, y AA, aunque durante el proceso de implantación, los nombres pueden variar.

El escalar se denomina n y es el número de renglones de la matriz. Se puede emplear además un escalar denominado nnz que indica el total de elementos de la matriz que son diferentes de cero, aunque no es indispensable dado que se puede calcular a partir del elemento IA[n+1]. Entonces $nnz = IA[n+1]-1$

El vector JA contiene los índices de las columnas de los elementos a_{ij} almacenados en AA, cuya longitud es *nnz*.

El vector AA contiene a todos los elementos a_{ij} diferentes de cero guardados renglón a renglón, desde el renglón 1 al renglón *n* y su longitud es *nnz*.

Existe una correspondencia uno a uno entre los elementos de JA y AA. Todos los elementos del vector JA cumplen con : $JA(i) > 0 \in \mathbb{N}; i \in \mathbb{N}$.

El vector IA contiene los índices que indican el principio de cada renglón en los vectores JA y AA. El elemento IA(i) es la posición en JA y AA donde el i-ésimo renglón comienza. Su longitud es $n+1$. El elemento $IA(n+1)$ es igual a $nnz+1$. Es importante notar que esté último elemento corresponde al comienzo de un renglón ficticio $n+1$.

A partir de los elementos del vector IA, se pueden determinar cuantos elementos diferentes de cero contiene cada uno de los renglones. El primer elemento de IA siempre es 1.

Este tipo de representación de matrices tiene una gran importancia en el computo científico, porque los sistemas de ecuaciones pueden ser representados por matrices, y muchos de estos sistemas que se utilizan contienen una gran cantidad de incógnitas, que generan matrices con miles o incluso cientos de miles de elementos; y en muchas ocasiones las matrices son dispersas, así que no tiene sentido almacenar de manera densa a estas matriz.

A partir de la siguiente matriz dispersa:

$$\begin{bmatrix} 1.4 & 1.9 & 0 & 0 & 0 & 0 & 0 \\ 1.5 & 1.1 & 0.9 & 0 & 0 & 0 & 0 \\ 0 & 0.8 & 1.9 & 1.7 & 0 & 0 & 0 \\ 0 & 0 & 1.2 & 1.4 & 1.1 & 0 & 0 \\ 0 & 0 & 0 & 0.9 & 1.7 & 1.3 & 0 \\ 0 & 0 & 0 & 0 & 0.7 & 1.6 & 1.4 \end{bmatrix}$$

Su representación en formato CSR es:

n=6
IA=[1, 3, 6, 9, 12, 15, 18]
JA=[1, 2, 1, 2, 3, 2, 3, 4, 3, 4, 5, 4, 5, 6, 5, 6, 7]

A=[1.4, 1.9, 1.5, 1.1, 0.9, 0.8, 1.9, 1.7, 1.2, 1.4, 1.1, 0.9, 1.7, 1.3, 0.7, 1.6, 1.4]

Que como se puede observar, no es indispensable que la matriz sea una matriz cuadrada, y que el tamaño del vector IA, corresponde al número de renglones que contiene la matriz.

- Se aprecia que el tamaño de IA es igual a n + 1 = 7
- Se aprecia que tanto el tamaño de JA como el de A es:

$$nnz = IA[n+1]-1 = IA[7]-1 = 18-1 = 17$$

- Se ve que los elementos que conforman a JA corresponde a la columna donde se encuentra almacenado un número diferente de cero

4.5 El programa

En Fortran originalmente las subrutinas son la forma habitual de escribir y agrupar bloques de código independiente y que se pueden invocar por otras subrutinas o por el programa principal, por ello en los parámetros se definen tanto los argumentos de entrada como los argumentos de salida además de que éstos ingresan a la subrutina por referencia y no por valor, por lo que en Fortran 90/95 se debe especificar la naturaleza de los argumentos de entrada, de que tipo son y cual es su propósito, si son de entrada o de salida.

Aunque el estándar empleado para definir el lenguaje a partir de la versión 90 especifica declaraciones para definir funciones, es un poco más extenso definir bloques de código de esa manera en contra posición al uso de subrutinas.

Caso contrario a lo que sucede en el lenguaje C, en donde las funciones son los bloques principales de construcción y para poder implementar el equivalente a una subrutina de Fortran 90 en C, se puede realizar mediante la declaración de una función

del tipo *void*, cuyos argumentos son todos declarados como punteros, permitiendo el paso de argumentos por referencia.

4.5.1 Módulos y funciones de apoyo

El algoritmo de solución aunque es relativamente sencillo, para su funcionamiento hace uso de funciones complejas, que es preferible programar aparte y contenerlos en archivos diferentes para lograr una mejor organización, además de que el código sea mas legible y mantenible.

En Fortran 90 a los archivos que conforman un proyecto y que contienen funciones auxiliares se les nombra módulos, en ellos se integran tanto las declaraciones como definiciones de las funciones, además de declarar su alcance. También Fortran 90 permite que en los módulos se pueda realizar sobrecarga de operadores, lo que permite mediante interfaces, ocultar el llamado de dos o más funciones que sean semejantes.

Apoyados en estas características del lenguaje, se escriben varios módulos que contienen funciones que tienen características comunes.

Por ello, además de la programación del *solver* paralelo, se crearon algunos módulos más a modo de bibliotecas, donde residen funciones para la manipulación de las matrices, que básicamente son operaciones sobre matrices como son: la multiplicación matriz-vector, el cálculo de la transpuesta, extracción de renglones, extracción de diagonales (transformaciones) y otras.

También existen otras funciones que realizan transformaciones en la matriz, para generar sistemas equivalentes pero más simples de resolver. Además de otros elementos auxiliares necesarios, como precondicionadores.

4.5.2 Los parámetros MPI

Se construyó un módulo donde se definen de manera global las variables que requiere el ambiente MPI, para que cada uno de los nodos participantes puedan disponer de la información en cualquier momento durante todo el proceso de computo.

Como se menciona sobre las características de MPI, durante ejecución del programa debe existir en cada uno de los nodos participantes información del estado global del proceso, por ello en este módulo se declaran todas las variables que indican el número de procesadores participantes, identificador para cada uno de ellos, una variable para recuperar el estatus de error de cada función MPI en caso que suceda y un vector de estado que emplea la función MPI_Send().

Aunque es recomendable no emplear variables globales en el proceso de desarrollo de software, esta fue la mejor forma de compartir las variables mientras se realiza el computo; por ello se buscó ser tan descriptivo como sea posible en el nombramiento de los elementos del módulo.

El código es el siguiente:

```
MODULE Env_mpi
   USE mpi

   SAVE

   INTEGER, PARAMETER :: dp_s = KIND(1.0D0)
   !---------- Data for mpi enviroment
   INTEGER :: myid     ! For the identifier of each processor (begin in 0)
   INTEGER :: numprocs ! Number of processors work
   INTEGER :: ierr     ! For save error value if mpi fail
   INTEGER :: estat (MPI_STATUS_SIZE) !Vector to subroutine mpi_send
END MODULE Env_mpi
```

Como se aprecia, es necesario incluir el uso del modulo mpi, dado que en éste se encuentran todas las definiciones concernientes a MPI. También notar que todas las variables definidas son para el manejo de las subrutinas MPI, excepto por la variable dp_s.

4.5.3 Las operaciones matriciales

Como el proceso de solución emplea operaciones matriciales, se decidió crear un módulo que contenga únicamente las funciones para realizar operaciones matriciales.

Las operaciones que se utilizan para realizar transformaciones sobre el sistema lineal son:

- Transpuesta
- Multiplicación matriz-vector
- Multiplicación matriz-vector en paralelo
- Extracción de renglones
- Extracción de diagonales

Es cierto que en Fortran 90 ya existen funciones definidas para operar con matrices, pero solo funcionan sobre matrices en formato denso, es decir aquellas que se representan con arreglos; y como en el problema a resolver se usa un formato comprimido, se requiere hacer nuevas funciones para trabajar con el formato CSR.

Esto es porque una matriz en formato CSR está constituida por 4 elementos, donde tres de ellos son vectores, en lugar de uno solo como en el caso de matriz densa.

Las primeras funciones hacen exactamente lo que su nombre expresa, las restantes se

emplean para reducir el sistema lineal, es decir la obtención de submatrices que se consideran subdominios.

En todas las funciones se adopta una notación para los parámetros de entrada, con el propósito de representar lo mejor posible la realización de las operaciones con las matrices que se encuentran en formato CSR, por lo que los argumentos representantes de la matriz de entrada van juntos y antes que los argumentos representantes de la matriz de salida.

El orden en que se pasan las matrices a las funciones siempre es el mismo y corresponde a la siguiente secuencia:

- n: orden de la matriz
- ia: vector para el manejo de los renglones
- ja: vector que contiene las columnas de los elementos diferentes de cero
- a: los elementos diferentes de cero

4.5.3.1 La función Transpuesta

Tiene como argumentos de entrada la matriz de entrada, a la que se le calcula su transpuesta, y la matriz de salida, en donde se coloca la traspuesta obtenida.

4.5.3.2 La función extracción de renglones

Como se comentó en el primer capítulo, se pueden tener diferentes efectos sobre una matriz dada, al realizar operaciones con algunas matrices características. Como se desea eliminar renglones de la matriz cuadrada, matemáticamente esto se logra de la siguiente manera.

Si se tienen una matriz cuadrada $A=[a_{ij}];i,j\in\mathbb{N};i=0,\cdots,n\,j=0,\cdots,n$ de orden $n\times n$ y si se premultiplica por una matriz rectangular de orden $m\times n$ $B=[\gamma_{ij}]$ tal que $\gamma=1\,si\,i=j$ y $\gamma=0\,si\,i\neq j$, donde además $m<n$, el producto es una matriz de orden $m\times n$ que está constituida por los primeros m renglones de la matriz cuadrada original. Es decir:

$$BA=A'\ ,$$

así A' es una matriz rectangular con los primeros m renglones de la matriz A

Ejemplificando, dada la siguiente matriz cuadrada

$$A=\begin{bmatrix}4.3 & 7.5 & 0.0 & 0.0\\ 6.9 & 5.7 & 8.6 & 0.0\\ 0.0 & 9.3 & 6.6 & 4.5\\ 0.0 & 0.0 & 5.5 & 7.3\end{bmatrix}$$

y si se desea obtener los primeros dos renglones de la matriz, se debe premultiplicar por una matriz que tenga la siguiente forma:

$$B=\begin{bmatrix}1.0 & 0.0 & 0.0 & 0.0\\ 0.0 & 1.0 & 0.0 & 0.0\end{bmatrix}$$

Es decir, al realizar la multiplicación entre matrices tendremos lo siguiente:

$$\begin{bmatrix}1.0 & 0.0 & 0.0 & 0.0\\ 0.0 & 1.0 & 0.0 & 0.0\end{bmatrix}\begin{bmatrix}4.3 & 7.5 & 0.0 & 0.0\\ 6.9 & 5.7 & 8.6 & 0.0\\ 0.0 & 9.3 & 6.6 & 4.5\\ 0.0 & 0.0 & 5.5 & 7.3\end{bmatrix}=\begin{bmatrix}4.3 & 7.5 & 0.0 & 0.0\\ 6.9 & 5.7 & 8.6 & 0.0\end{bmatrix}$$

que como se puede observar, da el resultado deseado que es el tener una matriz

rectangular constituida por los primeros dos renglones.

La matriz que premultiplica puede construirse en función de los renglones contiguos que se desean obtener; como se observa del primer análisis, se tendría una matriz identidad si ésta fuese cuadrada, teniendo únicamente 1.0's en su diagonal principal. Al ser rectangular y siempre tener más columnas que renglones, existen columnas completas conformadas exclusivamente de ceros.

Entonces, la matriz que permite obtener los renglones contiguos que no empiezan en el primer renglón, es aquella que tiene desplazada la diagonal principal tantas columnas como sea la posición del renglón inicial de la matriz trunca.

Para una matriz cuadrada de orden $n \times n$ de la que se desea los renglones que van de la posición k a la posición l, la matriz que premultiplica será $B = [\gamma_{ij}]$, de orden $(l-k+1) \times n$ conformada por $\gamma = 1 \; para \; i = j+l-1$, y $\gamma = 0$ en cualquier otro caso; $i = 1, \cdots, (l-k+1) \quad j = 1, \cdots, n$

En forma desarrollada queda:

$$B = \begin{bmatrix} \overbrace{0 \cdots 1}^{l} & 0 & 0 & \cdots & 0 \\ 0 \cdots 0 & 1 & 0 & \cdots & 0 \\ 0 \cdots 0 & 0 & 1 & \cdots & 0 \end{bmatrix}$$

Aunque existe un método matemático que permite realizar la acción de obtener una matriz rectangular con un número inferior de renglones al de la matriz original mediante operaciones matriciales, es conveniente remarcar el hecho de que la representación de matrices en la computadora se realiza mediante arreglos (arrays), que como se sabe, son localidades contiguas en la memoria RAM. Por ello es más práctico simplemente eliminar los elementos que no se desean emplear. La operación matemática antes

descrita en su implementación se reduce a un simple truncamiento de elementos de la matriz.

En el formato CSR el vector más pequeño que conforman a la matriz es aquel que describe a los renglones, por ello es más sencillo realizar una extracción de renglones, que hacer una extracción de columnas. Es por esta razón, que resulta más simple el manejo de renglones que de columnas, por lo cual solo se implementa la función extraer renglones y no existe una función que permita extraer columnas.

Esta función a pesar de ser simplemente un truncamiento sobre la matriz original, se ha mostrado que se puede obtener mediante operaciones matemáticas, razón por la cual se incluyó en el módulo mat_op.f90

El extraer renglones de la matriz da como resultado una matriz rectangular, que da un sistema lineal compatible indeterminado, dado que tenemos más incógnitas que ecuaciones. Como se requieren matrices cuadradas para que sea un sistema determinado, se necesita quitar renglones y columnas.

Pero como es más costoso el proceso de extraer columnas, se extraen renglones de la matriz original y de la transpuesta resultante. Por ello no se crea la función para la extracción de renglones.

4.5.3.3 Extracción de diagonales

Subrutina que su propósito es obtener una matriz que contiene únicamente los elementos que conforman la diagonal principal y los elementos adyacentes a esta. Entre sus argumentos se indica cuantos diagonales además de la principal se desean conservar de una matriz dada.

4.5.4 El módulo que contiene los elementos para el precondicionamiento

Como GMRES es un solver iterativo que requiere del apoyo de un precondicionador, estas subrutinas necesarias están almacenadas en el modulo llamado Module_LU.f90.

4.5.5 El módulo que contiene al solver

Como se mencionó con anterioridad, cada archivo independiente de código fuente se le llama módulo y como el *solver* es parte de un programa más grande, también se encuentra en un archivo independiente llamado solver.f90.

Este módulo contiene básicamente tres funciones, una función para obtener los subdominios, una función para resolverlos y la función que corresponde al *solver* siendo ésta la función principal, organizada de tal forma que permite incluirse en cualquier programa escrito en Fortran 90 en donde se requiera un *solver* que cumpla con las restricciones antes mencionadas, además de que corra en paralelo.

4.5.5.1 La extracción de subdominios

La declaración de la función es la siguiente:

```
SUBROUTINE Subdoms_sub(aa_n, aa_ia, aa_ja, aa_a, Num_dom, overlap, iord, &
                       riord, nts, niter, ju, jlu, alu, iperm)
```

Los argumentos de entrada para esta subrutina son la matriz original, el número de dominios en que se divide la matriz, el valor de uno de los traslapes, un par de vectores de reordenamiento, los valores absolutos de número de iteraciones totales realizadas y

los vectores para la factorización LU[11] incompleta.

La construcción de los subdominios consiste en crear a partir del sistema de ecuaciones lineales completo, sistemas más pequeños y que están conformados con la información del sistema original.

Como se está usando el algoritmo de descomposición de dominios de Schwarz, es necesario la existencia de traslape entre cada uno de los subdominos creados, para que se pueda compartir información entre ellos garantizando que la solución global se pueda obtener.

El proceso consiste en obtener bloques cuadrados y traslapados de la matriz original, a partir de los elementos que conforman la diagonal principal. El número de bloques creados será igual al número de procesadores invocados para el proceso paralelo. El valor del traslape es variable, de acuerdo al tamaño de la matriz, en general se considera un valor de un 5 % sobre el tamaño de los subdominios.

Por ejemplo, si se tiene una matriz de 70,000 variables, y se emplean 10 procesadores esclavos, con un traslape de 600 elementos tanto a un lado como al otro, las matrices que conforman los subdominios serán del orden de 8200×8200 , donde el orden de la matriz se divide entre el número de procesadores esclavos participantes, en este caso 70,000/10 y se le suma el doble del valor del traslape, siendo éste de 1200. Una restricción al proceso es el hecho que el orden de la matriz debe ser una división entera exacta entre el número de dominios deseados.

En realidad solo se requiere el sistema original completo durante el proceso de corrección de la solución, así que en el cálculo de las soluciones locales a cada subdominio se trabaja con subsistemas transformados, que permiten obtener la solución en un tiempo menor.

11 LU Lower Upper, es el resultado de factorizar a una matriz. A = L U

Es por ello que las submatrices obtenidas a partir de la matriz original, se les elimina aquellos valores que son inferiores a un cierto límite, por considerarlos insignificantes o de poca repercusión en la solución del sistema.

Como el sistema de ecuaciones lineales es el resultado de un proceso anterior, y se repite muchas veces durante la ejecución de todo el programa, el esquema del sistema completo se construye una única vez al principio de todo el proceso y se conserva, aún cuando en el esquema se consideran posiciones de datos que aparecerán mucho después en el proceso. Así que estos lugares se ocupan con ceros que para solucionar el sistema, es mucho más conveniente eliminar; de esta forma se reduce el número de elementos de la matriz y por tanto, el tiempo de solución.

Por supuesto, es necesario realizar una evaluación previa sobre los sistemas lineales que se quieran resolver, si se desea emplear este *solver* en otro problema para determinar que procesos es deseable omitir durante el proceso de solución, porque por ejemplo en otro problema, no existen elementos cero ocupando lugares de próximos valores, o no se pueden omitir elementos por no lograr ser insignificantes.

Después de la eliminación, se reordenan a partir de un proceso llamado AMD, cuyos elementos que son vectores de reordenamiento se calcula una única vez; y para determinar el momento adecuado se usan los contadores globales que permiten determinar cuando se da la primera iteración de todo el proceso. Los vectores del proceso AMD se emplean durante todas las llamadas al *solver* para reordenar las submatrices.

Por último se transforma las submatrices reordenadas en una variante de matrices LU.

Parte del código correspondiente a la construcción de los subdominios, se puede consultar en el apéndice A.

4.5.5.2 Resolviendo los subdominios

Aunque formalmente el algoritmo de Schwarz se aplica en todo el proceso de solución, se creó un módulo que básicamente cumple con lo más evidente del algoritmo que se declaró de la siguiente manera:

```
SUBROUTINE Schwarz_sub(aa_n, aa_ia, aa_ja, aa_a, b, x0, Num_dom, overlap, &
                       ju, jlu, alu, iperm, tol_sch, riord, iord)
```

Donde los argumentos de entrada son el sistema lineal completo, el número de dominios, el traslape, las submatrices ya transformadas, la tolerancia, y los vectores de reordenamiento.

Se emplea el sistema original completo solo para calcular el residual y determinar si se ha cumplido con la tolerancia, y en función de ello dejar de iterar.

En general, los subdominios medios se traslapan tanto con sus vecinos anteriores, como con sus vecinos posteriores, excepto el primero y el último, casos particulares, por no tener vecinos el uno anterior y el otro posterior. Por ello se calculan el valor general para el traslape de los sudominios medios, y los traslapes particulares para los subdominios primero y último. Posteriormente se calculan los índices para inicio y fin de bloque.

Este proceso se realiza para obtener la parte del residual que corresponde a cada subdominio.

Posteriormente se reordena el residual local y con la submatriz se construyen sistemas locales de la forma $A_i x_i = e_i$ y se resuelven

Con las soluciones locales calculadas, se crea un vector completo para el sistema global que es una corrección a la solución inicial que se le agrega posteriormente para mejorar la aproximación. Este proceso iterativo termina cuando se alcance un número determinado de iteraciones ó antes si se cumple primero con el criterio de convergencia.

4.5.5.3 La función principal. El solver en paralelo

La declaración de la función es la siguiente:

```
SUBROUTINE GMRES_P(m, a_n, a_ia, a_ja, a_a, b, X, toler, nts, niter, &
                   riord, iord, nz)
```

En donde los argumentos de entrada son los parámetros del *solver*, la matriz que representa los coeficientes del sistema lineal en formato CSR, el vector de términos independientes, una primera aproximación, un par de variables globales sobre el número total de iteraciones realizadas y un par de vectores para la reorganización de la matriz.

El proceso inicia al recibir el sistema de ecuaciones lineales completo, obtener el residual de éste, extraer a la matriz inicial las diagonales principales y después mandar a construir los subdominios, en donde como se mencionó se obtienen los elementos necesarios para trabajar con el precondicionador.

En general, el *solver* es una implementación del algoritmo GMRES que como se explicó previamente, requiere de un precondicionador para acelerar su proceso de convergencia.

En este caso el precondicionador está apoyado por el algoritmo de descomposición de

dominios de *Schwarz* que se explicó previamente.

Posteriormente se realizan una serie de pasos correspondientes al calculo de la solución y en el momento que se requiere invocar al precondicionador, se invoca a la función Schwarz_sub que se explicó previamente.

4.6 Resultados y conclusiones

Durante el desarrollo del solver, se logró tener una versión inicial, que sirvió para evaluar la posibilidad de su paralelización, al que se le denomina Paralelo 1. Posteriormente se hicieron adecuaciones al solver en su versión secuencial para acelerarlo y mejorarlo, modificaciones que fueron migradas a la versión paralela y a la cual se le denominó Paralelo 2. Es importante hacer notar que del simulador solamente el solver es el único que está programado en paralelo.

Para probar el solver se realizaron una serie de simulaciones de yacimientos de petróleo, de las cuales solo se muestran los resultados más representativos. La primer simulación con las siguientes condiciones:

- Tamaño del yacimiento 100 x 100 x 10, 4 pozos, yacimiento fracturado
- Tamaño de las celdas: Lx = 10,000 m, Ly = 10,000 m, Lz = 300 m
- Tiempo de simulación = 2,000 días
- Valor Pwf de referencia = 3891.85
- 11 procesadores para el caso en paralelo

Obteniendo los siguientes resultados:

Algoritmo	**NTS**	**t simulador (s)**	**t solver (s)**	**T restante (s)**	**Pwf**
Serial 1	146	22890	---	---	4206.3

Serial2	246	4305	2279	2026	3883.18
Paralelo 1	246	11529	9247	2282	3885.18
Paralelo 2	246	3888	1606	2282	3896.92

En el siguiente caso se tiene una simulación más compleja, al incluir más elementos que complican la naturaleza del yacimiento:

- Tamaño del yacimiento 72 x 24 x 12, 105 pozos, yacimiento fracturado
- Tamaño de las celdas: Lx = 90,000 m, Ly = 10,000 m, Lz = 100 m
- Tiempo de simulación = 3,000 días
- 11 procesadores para el caso en paralelo.

Algoritmo	**NTS**	**t simulación (s)**	**t solver (s)**	**T restante (s)**	**Pwf**
Serial 2	1039	5635	3614	2021	3758.7
Paralelo 2	1028	3357	1002	2355	3758.8

Es importante notar que en este caso, los algoritmos que se tenia previamente y que se refieren como *Serial 1* y *Paralelo 1* fueron incapaces de funcionar en la simulación, por lo que no se incluyen los datos.

Por último, se simuló un yacimiento con las siguientes características:

- Tamaño del yacimiento 120 x 120 x 15, 4 pozos, yacimiento fracturado
- Tamaño de las celdas: Lx = 10,000 m, Ly = 10,000 m, Lz = 300 m
- Tiempo de simulación = 1,000 días
- 11 procesadores para el caso en paralelo.

Obteniendo los siguientes resultados:

Algoritmo	**NTS**	**t simulador (s)**	**t solver (s)**	**T restante (s)**	**Pwf**

Paralelo 2	291	8906	3330	5576	3851.24

Para este caso, el único solver que fue capaz de resolver el problema fue el programado en paralelo con adecuaciones, al que se le llamó Paralelo 2.

También se hizo una evaluación en cuanto al número de procesadores involucrados en el proceso paralelo, para determinar si realmente al incrementar éstos, el tiempo disminuía, con lo cual para un yacimiento de dimensiones 120 x 120 x 12, y con un pozo se obtuvieron los siguientes resultados:

	ND			
Iteraciones	**3**	**4**	**5**	**6**
30		10:37	10:15	9:54
40		15:02	14:20	13:48
53	31:15	27:42	25:10	23:54
60	50:50	44:49	40:06	38:08
61	54:20	47:59	42:56	40:42
62	58:25	51:39	46:01	43:22
63	62:20	55:18	48:56	46:02
64	66:35	58:50	51:56	48:42
65	70:45	62:15	54:51	51:22
66	75:20	65:40	57:46	53:42
73	104:36	90:01	79:31	72:52
75	113:21	97:15	85:46	77:47

Que como se puede observar, efectivamente al incrementar el número de procesadores, se disminuye el tiempo que tarda una simulación.

Después de los resultados anteriores y al hacer una evaluación de las diferentes ofertas relativas a supercomputo, se concluyó que un cluster tipo Beowulf es mucho muy conveniente dado las ventajas que ofrece, principalmente el hecho de poder construirse

con equipo de computó que se ofrecen en cualquier tienda especializada.

Como consecuencia de esta decisión, fue fundamental el utilizar para la programación paralela el modelo de paso de mensajes apoyado por la biblioteca MPI.

Y para aprovechar el cluster, resultó necesario identificar que procesos se pueden paralizar, encontrado la viabilidad de hacerlo sobre el solver al emplear el algoritmo de descomposición de dominios de Schwarz.

Al usar el solver paralelo dentro de un simulador de yacimientos de petróleo con diferente número de procesadores y para las mismas condiciones, se mostró una reducción en el tiempo de procesamiento, a mayor número de procesadores, un menor tiempo de cómputo.

Además se pudo atacar sistemas de mayor tamaño en el cluster, que los posibles de resolver en una PC, por lo que paralelizar el solver resultó beneficioso, por los ahorros presentados.

Apéndice A

Secciones del código de las subrutinas de solver.f90

Sección correspondiente a la subrutina Subdoms_sub()

```
   block_size = INT(aa_n / Num_dom)
   subdomain_size = block_size + overlap * 2
!-------------------------------------------------------------------------
-
!-- Calc the subdomains and distribution
---------------------------------------
   IF(myid == 0) THEN
   ! It calculates and it distributes each subdomain
      DO i = 1, Num_dom
         top_overlap = overlap
         IF(i == 1) THEN
            top_overlap = 0
         ELSE IF (i == Num_dom) THEN
            top_overlap = overlap * 2
         END IF
         begin_sub = block_size * (i - 1) + 1 - top_overlap
         CALL Extract_r(subdomain_size, begin_sub, aa_n, aa_ia, aa_ja, aa_a, &
                        tmp_a_n, tmp_a_ia, tmp_a_ja, tmp_a_a)
         tmp_b_n = aa_n
         ALLOCATE(tmp_b_ia(aa_n + 1))
         ALLOCATE(tmp_b_ja(tmp_a_ia(tmp_a_n + 1) - 1))
         ALLOCATE(tmp_b_a(tmp_a_ia(tmp_a_n + 1) - 1))

         CALL Transpuesta(tmp_a_n, tmp_a_ia, tmp_a_ja, tmp_a_a, tmp_b_n, &
                          tmp_b_ia, tmp_b_ja, tmp_b_a)

         DEALLOCATE(tmp_a_ia, tmp_a_ja, tmp_a_a)
```

```
         CALL Extract_r(subdomain_size, begin_sub, tmp_b_n, tmp_b_ia, &
                        tmp_b_ja, tmp_b_a, tmp_a_n, tmp_a_ia, tmp_a_ja,
tmp_a_a)

         DEALLOCATE(tmp_b_ia, tmp_b_ja, tmp_b_a)

         tmp_b_n = tmp_a_n
         ALLOCATE(tmp_b_ia(tmp_a_n+1))
         ALLOCATE(tmp_b_ja(tmp_a_ia(tmp_a_n + 1) - 1))
         ALLOCATE(tmp_b_a(tmp_a_ia(tmp_a_n + 1) - 1))

         CALL Transpuesta(tmp_a_n, tmp_a_ia, tmp_a_ja, tmp_a_a, tmp_b_n, &
                          tmp_b_ia, tmp_b_ja, tmp_b_a)

         !CALL Transpuesta(tmp_a_n,tmp_b_n, tmp_a_a, tmp_a_ja, tmp_a_ia, &
         !                 tmp_b_a, tmp_b_ja,tmp_b_ia)
         DEALLOCATE(tmp_a_ia, tmp_a_ja, tmp_a_a)
   !---------- It send each subdomain ----------------------------------------
         CALL MPI_SEND(tmp_b_n, 1, MPI_INTEGER, i, 0, MPI_COMM_WORLD, ierr)
         CALL MPI_SEND(tmp_b_ia, (tmp_b_n + 1), MPI_INTEGER, i,0, &
                       MPI_COMM_WORLD, ierr)
         nnz = tmp_b_ia(tmp_b_n + 1) - 1
         CALL MPI_SEND(tmp_b_ja, nnz, MPI_INTEGER, i, 0, MPI_COMM_WORLD, &
                       ierr)
         CALL MPI_SEND(tmp_b_a, nnz, MPI_DOUBLE_PRECISION, i, 0, &
                       MPI_COMM_WORLD, ierr)
         DEALLOCATE(tmp_b_ia, tmp_b_ja, tmp_b_a)
      END DO
   ELSE
   !---------- Receive local subdomain
----------------------------------------
      top_overlap = overlap
      down_overlap = top_overlap
      IF (myid == 1) THEN
         top_overlap = 0
         down_overlap = overlap * 2
```

```
   ELSE IF (myid == Num_dom) THEN
      down_overlap = 0
      top_overlap = overlap * 2
   END IF
   begin_sub = block_size * (myid - 1) + 1 - top_overlap

   CALL MPI_RECV(tmp_a_n, 1, MPI_INTEGER, MPI_ANY_SOURCE, 0, &
                 MPI_COMM_WORLD, estat, ierr)
   ALLOCATE(tmp_a_ia(tmp_a_n + 1))
   CALL MPI_RECV(tmp_a_ia, tmp_a_n+1, MPI_INTEGER, MPI_ANY_SOURCE, 0, &
                 MPI_COMM_WORLD, estat, ierr)
   ALLOCATE(tmp_a_ja(tmp_a_ia(tmp_a_n + 1) - 1))
   ALLOCATE(tmp_a_a(tmp_a_ia(tmp_a_n + 1) - 1))
   nnz = tmp_a_ia(tmp_a_n + 1 ) - 1
   CALL MPI_RECV(tmp_a_ja, nnz, MPI_INTEGER, MPI_ANY_SOURCE, 0, &
                 MPI_COMM_WORLD, estat, ierr)
   CALL MPI_RECV(tmp_a_a, nnz, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE, 0, &
                 MPI_COMM_WORLD, estat, ierr)
END IF
```

Sección del código correspondiente al algoritmo de Schwarz

```
block_size = INT(aa_n / Num_dom)
comm_size = block_size + 2 * overlap
ALLOCATE(et_l(comm_size))
ALLOCATE(et_g(block_size))
et_g = 0.0
maxr = 0.0
flag = 0

ALLOCATE(comm_x0(aa_n))
comm_x0 = x0

ALLOCATE (res(aa_n))
subdomain_size = block_size + overlap * 2
```

```
   !----------- Begin the local solution calc
--------------------------------
   !---------- Firts residual, only the master node
---------------------------
   IF (myid == 0) THEN
      ALLOCATE (error(aa_n))
      ALLOCATE(et_buffer(block_size * numprocs))
      comm_x0 = x0
      ALLOCATE(ax(aa_n))
      ax = 0.0
      CALL amux(ax, aa_n, aa_ia, aa_ja, aa_a, comm_x0)
      !CALL amux(aa_n, comm_x0, ax, aa_a, aa_ja, aa_ia)
      res = b - ax
      DEALLOCATE(ax)
   ELSE
   !---------- The other nodes calculate sizes
      nl = SIZE(ju)
      top_overlap = overlap
      down_overlap = top_overlap
      IF (myid == 1) THEN
         top_overlap = 0
         down_overlap = overlap * 2
      ELSE IF (myid == Num_dom) THEN
         down_overlap = 0
         top_overlap = overlap * 2
      END IF
      begin_sub = block_size * (myid - 1) + 1 - top_overlap
      end_sub = myid * block_size + down_overlap
   END IF

   CALL MPI_BARRIER(MPI_COMM_WORLD, ierr)
   CALL MPI_BCAST(res, aa_n, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)

   j = 0 ! Is the counter of iterations number
   DO
      IF (myid /= 0) THEN
```

```
   !---------- The others nodes allocate local vectors
         ALLOCATE(et(subdomain_size))
         ALLOCATE(et0(subdomain_size))
         ALLOCATE(rt(subdomain_size))

         rt = res(begin_sub:end_sub) ! A portion of the residual one
         et0 = rt(riord)
         rt = et0
         et0 = 0.0

         !-------- Solve LU system
-----------------------------------------
         CALL solve(nl, rt, et0, alu, jlu, ju)

         DO k = 1, subdomain_size
            et(k) = et0(iperm(k + subdomain_size))
         END DO
         DO k = 1, subdomain_size
            et_l(k) = et(iord(k))
         END DO
         IF (myid == 1) THEN
            et_g = et_l(1:block_size)
         ELSE IF (myid == Num_dom) THEN
            et_g = et_l(2*overlap+1:)
         ELSE
            et_g = et_l(overlap+1:block_size+overlap)
         END IF
         DEALLOCATE(rt)
         DEALLOCATE(et, et0)
      END IF

      CALL MPI_GATHER(et_g, block_size, MPI_DOUBLE_PRECISION, et_buffer, &
                      block_size, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, &
                      ierr)
      !---- Recolection and distribution of all subdomain solutions
-----------
```

```
      IF (myid == 0) THEN
         error = et_buffer(block_size+1:)
         comm_x0 = comm_x0 + error
      END IF

      CALL MPI_BARRIER(MPI_COMM_WORLD, ierr)
      CALL MPI_BCAST(comm_x0, aa_n, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, &
                     ierr)

      j = j + 1
      IF (myid == 0) THEN
      !-------------------------------------------------------------------------
-
      ! Se utiliza algun criterio de paro antes de obtener el residual
         maxr = SQRT(DOT_PRODUCT(error,error))
      END IF
      IF (myid == 0) THEN
         IF (maxr < tol_sch) flag = 1
      END IF
      CALL MPI_BCAST(flag, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      IF(j > 30 .OR. flag == 1) THEN
         EXIT
      END IF
      IF (myid == 0) THEN
         ALLOCATE(ax(aa_n))
         CALL amux(ax, aa_n, aa_ia, aa_ja, aa_a, comm_x0)
         !CALL amux(aa_n, comm_x0, ax, aa_a, aa_ja, aa_ia)
         res = b - ax
         DEALLOCATE(ax)
      END IF
      CALL MPI_BCAST(res, aa_n, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
   END DO
```

Bibliografía

Libros:

Aoyama Yukiya, Nakano Jun, *RS/6000 SP: Practical MPI Programming*, agosto 1999

Buyya, Rajkumar, "*High Performance Cluster Computing: Architecture and Systems"*, volumen 1, junio 1999.

Grama Ananth , Karypis George, Kumar Vipin,Gupta Anshul*, "An Introduction to Parallel Computing: Design and Analysis of Algorithms*", Second Edition, Pearson Addison Wesley, 2003.

Jaramillo Jaramillo Juan David, Vidal Maciá Antonio M., Correa Zabala Francisco José,*"Métodos directos para la solución de sistemas de ecuaciones lineales simétricos, indefinidos, dispersos y de gran dimensión"*, Febrero de 2006.

Solar Gonzalez E, Speziale De Guzmán L. *"Apuntes de álgebra lineal."* Limusa Noriega Editores, 1996.

Sterling Thomas, Salmon John, J. Becker Donald, F. Savarese Daniel*, "How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters"*, mayo 1999.

Artículos:

Cervantes C. Guillermo, Mejía S. Carlos, *"Precondicionamiento de métodos iterativos"*, 2004

Koenker Roger, Ng Pin*, "SparseM: A Sparse Matrix Package for R *"*, Marzo 7 de 2003

Meddahi Salim, Sayas Francisco Javier, *"Introducción a los métodos de descomposición de dominio"*, Junio de 1995.

R. Rodrigo Paz*,"Métodos de Descomposición de Dominio en Mecánica de Fluidos Computacional"*, 2006

Páginas de la Internet:

Dongarra, Jack (2000). *"Sparse Matrix Storage Formats"*.
http://www.cs.utk.edu/~dongarra/etemplates/node372.html