64

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

APLICACION DE PROCESAMIENTO EN PARALELO A UN SISTEMA DE OPTIMIZACION DE RECARGAS DE COMBUSTIBLE NUCLEAR

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

MARTIN LOPEZ SANCHEZ

DIRECTOR DE TESIS: DR. JUAN LUIS FRANCOIS LACOUTURE

CIUDAD UNIVERSITARIA MEXICO, D.F. 2002

La formación de todo ingeniero mexicano es, la de transformar todos nuestros recursos naturales, renovables y no renovables, tecnológicos, ideológicos para el bien de nuestra sociedad.

La formación que he adquirido y aprendido en ingeniería me condujo a ser una persona con virtudes, principios y cualidades para poder auxiliar a mi país. Ya que al realizar un análisis profundo en los habitantes de la República Mexicana, me di cuenta que tenemos muchas carencias.

Estas carencias principalmente se deben a la educación que recibimos y al no querer ser profesionistas de excelencia, por otro lado los principales motivos a los que nos enfrentamos es la falta de oportunidades tanto en las empresas públicas y privadas así como la ausencia de capital.

La dificultad que todo ingeniero enfrenta es la inexperiencia a nivel profesional y laboral, ya que la educación o formación que recibimos no es la adecuada cuando nos enfrentamos a los problemas de la vida real, por el solo hecho de que dependemos mucho de otras regiones del mundo, tanto de nuestras tradiciones, costumbres, así a la vez del idioma y de la tecnología.

Por ello en México es necesario formar profesionales de calidad, para que nuestro país deje de pertenecer al tercer mundo y pasemos a ser un país del primer mundo y tal vez seamos autosuficientes y aprovechemos todos los recursos que la naturaleza brindó y proporcionó a nuestro territorio.

PROVERBIO MEXICANO : PRIMERO LIMPIEMOS Y MEJOREMOS NUESTRO OJO IZQUIERDO PARA PODER OBSERVAR CON EL OJO DERECHO.

Dedico esta tesis a:

Mis Padres; Francisco López Márquez y María Dolores Sánchez Olmedo, a mis consortes Emma Oliveros Arontes y María de los Ángeles Soto Rivera, a mi hijo Rhamm López Oliveros, a mi familia de apellidos López Sánchez y García Sánchez, a mis camaradas Francisco Javier Montoya Cervantes, Aurelio Sánchez Vaca, Cruz Sergio Aguilar Díaz, y algunos otros que me apoyaron en la facultad de ingeniería. Así de igual manera, al Dr. Ernesto Zedillo Ponce De León, al Lic. Jesús Olvera por haberme apoyado en cuestiones prácticas de Juicios; familiares, civiles y penales. Por otro lado a todos aquellos que me auxiliaron y confiaron en mi para ser cada día a día mejor.

ÍNDICE

INTRODUCCIÓN

Una de las variantes que me condujo a plantear este tema, es sin duda la necesidad de poder desarrollar un tema novedoso para aplicarlo a un problema de ingeniería, las posibilidades de contar con diferentes equipos de cómputo y programación me condujeron a realizar un análisis de como podría atacar el problema, utilizando herramientas nuevas y novedosas dentro del campo de la computación.

Es importante poder presentar en este trabajo un tópico de varias alternativas de solución.

El fondo y la forma de como este trabajo quedó estructurado para obtener las soluciones planteadas, las cuales describo a continuación, ya que plantear de esta manera el desarrollo del presente trabajo de investigación, me permitió obtener mejores resultados al momento de ir analizando cada uno de los temas.

Primeramente, me enfoco a la descripción de los fundamentos de la energía nuclear, ya que el objetivo planteado es, el aplicar en su conjunto el procesamiento en paralelo a un sistema de optimización de recargas de combustible nuclear. Consecuentemente dentro de la investigación, describo algunos aspectos de la forma en que se genera y produce la energía eléctrica. Ya que actualmente, la forma de controlar la mayor parte de los procesos industriales, es por medio de computadoras y programas especializados.

Posteriormente describo los aspectos generales y particulares de un diseño de sistemas de recargas de combustible nuclear utilizando algoritmos genéticos, además incorporo a éste una descripción y análisis de los algoritmos genéticos, ya que en su conjunto el sistema es aplicado en la actualidad en la nucleoléctrica de Laguna Verde, en el Estado de Veracruz.

Así mismo menciono y describo la historia del procesamiento en paralelo, ya que varios autores han investigado y propuesto varios métodos de desarrollo e implementación, pero en lo particular trato de incorporar nuevas técnicas más modernas, planteando su descripción, utilización y aplicación.

En consecuencia y con los antecedentes antes referidos me adentro a describir la aplicación del procesamiento en paralelo, ya que este será, la parte medular del presente trabajo de investigación y donde se tomará como base a la Máquina Virtual Paralela.

A la vez se hace un resumen de las funciones, rutinas, bibliotecas y estructura del sistema, que en su conjunto y con la utilización, coordinación, planeación y ejecución con otros sistemas, darán cabida a proponer tres alternativas de solución planteadas como objetivo fundamental del presente trabajo.

Partiendo del objetivo propuesto y teniendo clara la solución a donde quiero llegar, realizó las especificaciones y requerimientos del sistema, así como el desempeño y funcionabilidad del mismo.

Por otra parte propongo tres alternativas de diseño, para dar solución al objetivo planteado, llamándolas;

1.- " Sistema parcialmente distribuido ".
2.- " Sistema con balanceo de procesos ".
3.- " Sistema completamente distribuido ".

Así es como elegiré, posteriormente la mejor solución, para aplicarla al sistema de optimización de recargas utilizando algoritmos genéticos que actualmente está desarrollada en forma secuencial y al proponer la utilización de la programación paralela en este caso, auxiliándome de la Máquina Virtual Paralela (PVM), sistemas operativos y computadoras personales conectadas en red.

El uso de la programación paralela desde el punto de vista de la aplicación, y con la utilización de las computadoras, experimentan una tendencia de niveles de sofisticación como los siguientes:

* Procesamiento de datos.
* Procesamiento de información.
* Procesamiento del conocimiento.
* Procesamiento de la inteligencia.

Se inicia la era de las computadoras, con el procesamiento de datos, al desarrollar más estructuras de datos y la exigencia de más cálculos, simultáneamente aparece el procesamiento de la información, la mayor parte de las computadoras actuales están dentro de estos dos niveles. Posteriormente el procesamiento del conocimiento, creció rápidamente reuniendo bases de conocimiento acumulado, por ejemplo los diferentes sistemas expertos los cuales se utilizan para la solución de problemas en áreas especificas.

Las computadoras de hoy disponen de gran capacidad en las celdas de memoria para el procesamiento de datos – información – conocimiento. Nos encontramos en la era donde las computadoras pueden tener un alto grado de capacidad de operar conocimientos y se proporciona su uso no sólo para el procesamiento convencional de datos-información, sino también para la construcción de sistemas factibles de conocimiento–inteligencia.

En el año de 1966 Michael J. Flyn [H3] se basa en la multiplicidad de los flujos de instrucciones y datos de un sistema de computadora para clasificarlos.

El proceso fundamental de una computadora, es la ejecución de una secuencia de instrucciones sobre un conjunto de datos.

Una forma de procesamiento que se realiza actualmente, es la de procesamiento secuencial, que consiste en la ejecución completa de una instrucción o tarea antes de ejecutar la siguiente instrucción tarea o proceso.

Desde el punto de vista de sistema operativo las computadoras han evolucionado en cuatro formas.

* Procesamiento por lotes.
* Multiprogramación.
* Tiempo compartido.
* Multiprocesamiento.

En estas cuatro formas de operación anteriores el grado de paralelismo se va incrementando rápidamente de fase en fase. La tendencia general es reforzar el procesamiento en paralelo de la información. Entonces podemos decir que el procesamiento en paralelo, es la forma de procesamiento que facilita la aplicación de los sucesos concurrentes en un proceso de computación. A la vez la concurrencia implica paralelismo, simultaneidad, y solapamiento.

Los sucesos paralelos son aquellos que pueden producirse en diferentes recursos durante el mismo intervalo de tiempo, los sucesos simultáneos se producen en el mismo instante de tiempo, los sucesos solapados se producen en intervalos de tiempo superpuestos. Los sucesos concurrentes pueden originarse en un sistema de computación en varios niveles de procesamiento. El procesamiento paralelo exige la ejecución concurrente en el computador de muchos programas.

El nivel más alto de procesamiento en paralelo se aplica a trabajos y programas múltiples a través de la multiprogramación, el tiempo compartido y el multiprocesamiento. Este nivel exige que se desarrollen algoritmos que se procesen en paralelo. La implementación de algoritmos paralelos depende de la asignación eficaz de limitados recursos de *software* y *hardware* a los múltiples programas que están siendo utilizados para resolver un problema extenso. El siguiente nivel de procesamiento en paralelo se aplica a procedimientos o tareas (segmentos de programa) dentro del mismo programa. Esto significa la descomposición de un programa en múltiples tareas.

El tercer nivel trata de explotar la concurrencia entre múltiples instrucciones.

De acuerdo con los conocimientos aportados por Flyn: El flujo se emplea para mostrar una secuencia de elementos " instrucciones o datos " que ejecuta u opera un único procesador. Las instrucciones y datos se definen con respecto a una máquina referida. Un flujo de instrucciones es una secuencia de instrucciones ejecutadas por una máquina. Un flujo de datos es una secuencia de datos que incluye los datos de entrada y los resultados parciales o totales solicitados o producidos por el flujo de instrucciones.

Por lo tanto en el presente trabajo se planteará la manera de utilizar la capacidad de la Máquina Virtual Paralela (por sus siglas en inglés, llamada PVM[B1]) para el procesamiento en paralelo y estudiar la factibilidad de utilizarla en una aplicación de ingeniería nuclear en una red heterogénea de cómputo. A la vez, con el aprovechamiento de sistemas de recargas de combustible nuclear que utiliza como base un modelo de Sistemas de Optimización de Patrones de Recarga utilizando Algoritmos Genéticos (SOPRAG)[F1].

La cantidad de cálculos que efectúa este sistema, por ser tan complicado utiliza varias horas de tiempo de cómputo dependiendo de la computadora que utilice, por lo que la aplicación de procesamiento en paralelo para la reducción del tiempo empleado en estos problemas de optimización, se presenta como una alternativa sumamente interesante para que el ingeniero en computación y el ingeniero industrial, puedan implementar y dar solución a tal sistema y factiblemente mejorarlo, para reducir costos y tiempos de procesamiento.

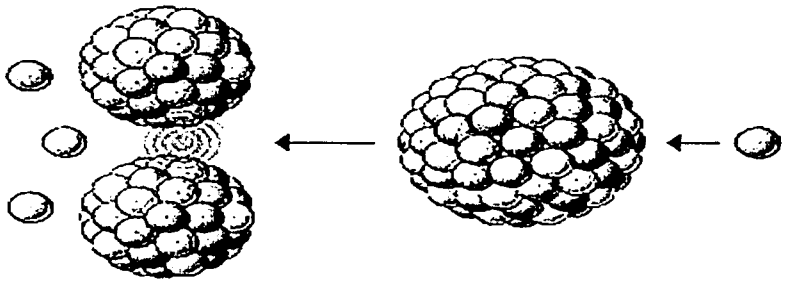
CAPITULO 1

FUNDAMENTOS DE LA ENERGÍA NUCLEAR.

1.1. DESCUBRIMIENTO DE LA ENERGÍA NUCLEAR.

Los experimentos sobre la radiactividad de ciertos elementos como el uranio, el polonio y el radio llevados a cabo a finales del siglo pasado por Henri Becquerel y por Pierre y Marie Curie, condujeron en 1902 al descubrimiento del fenómeno de la transmutación de un átomo en otro diferente, a partir de una desintegración espontánea que ocurría con gran desprendimiento de energía [C2].

Seguramente se podrían obtener fabulosas cantidades de energía. En 1938 Otto Hahn, Fritz Strassman y Lise Maitner pudieron comprobar el fenómeno de la fisión nuclear (figura 1), bombardeando con neutrones núcleos del isótopo del uranio 235. En esta reacción cada núcleo se parte en dos núcleos de masas inferiores, emitiendo radiaciones y liberando energía que se manifiesta en forma térmica y emite dos o tres nuevos neutrones.



La fisión nuclear [T26].

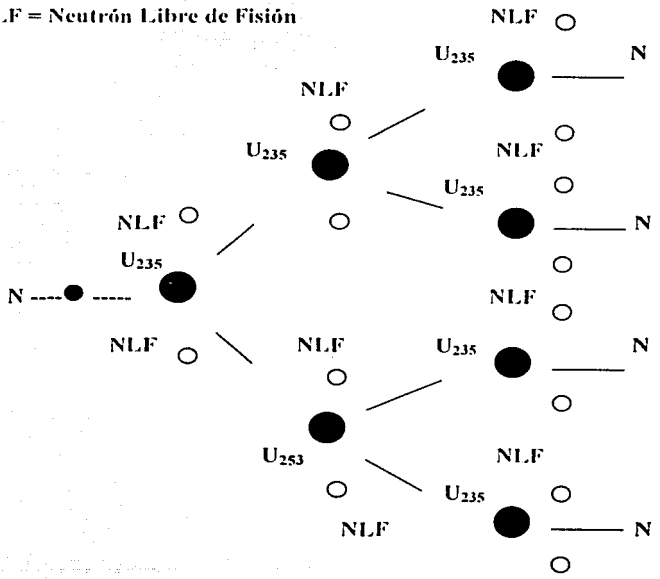
Figura 1.

Esta última circunstancia llevó al físico italiano Enrico Fermi a tratar de mantener y controlar una reacción nuclear, utilizando los neutrones producidos en la fisión de núcleos de U^{235} (Uranio 235) para fisionar otros núcleos del mismo isótopo que se denomina "reacción en cadena" (figura 2), la cual finalmente se logró producir en diciembre de 1942; el control de la reacción en cadena se obtuvo mediante la absorción de neutrones por elementos cómo boro y cadmio.

N = núcleo

N = neutrón

NLF = Neutrón Libre de Fisión



Reacción en cadena [T26].

Figura 2.

Los descubrimientos que arrojaron como consecuencia a la energía nuclear se transporta hasta la primera mitad de la década de los 50, cuando por primera vez se empleó la energía nuclear para generar electricidad.

En cada una de las fisiones se produce una pequeña cantidad de energía en forma de calor; al producirse la reacción en cadena se suman las energías producidas en cada fisión.

Se puede obtener con este proceso una cantidad de energía considerable. Siendo esté el origen de la energía nuclear, para el funcionamiento de la mayor parte de los reactores nucleares se utiliza un combustible llamado uranio enriquecido.

El proceso de obtención del mineral se somete a rigurosos procesos para lograr que llegue a contener aproximadamente 3 % de uranio 235, y mantener el control de una reacción en cadena[C1].

Para aprovechar y controlar la reacción en cadena se emplea un sistema llamado reactor nuclear.

1.2. LA ENERGÍA NUCLEAR PARA PRODUCIR ELECTRICIDAD.

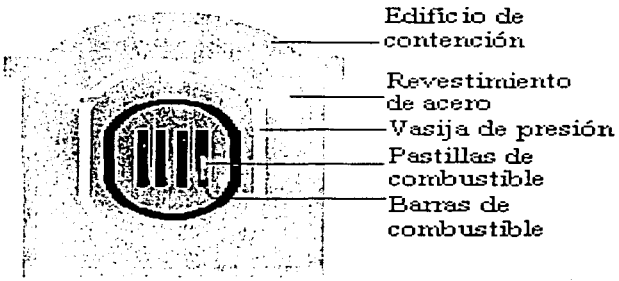
Las centrales nucleoléctricas funcionan con el mismo principio que las centrales térmicas convencionales, se utiliza el calor para producir vapor. En las térmicas convencionales el calor se obtiene de la combustión del carbón, hidrocarburos, combustoleo y gas. En las nucleoeléctricas el calor se obtiene de la fisión del uranio. En todos los casos, el "combustible" debe de ser trasladado desde las minas, o centro de elaboración hasta la central. Después de utilizarse en las centrales nucleoeléctricas el combustible se envía a lugares donde se reprocesa para extraer los productos útiles; los productos radioactivos se separan para almacenarse en forma de productos químicos insolubles. También pueden almacenarse temporalmente en albercas, en las centrales nucleoeléctricas.

La energía de las fisiones que ocurren en el interior del reactor hace que se caliente el agua en una vasija (figura 3). Esta agua lo mismo que sucede en otras centrales térmicas de carbono o de combustoleo, se convierte en vapor para mover una turbina e impulsar al generador donde se produce electricidad.

Entre los elementos del combustible se pueden introducir barras de control fabricadas de boro, material capaz de absorber a los neutrones libres. Al introducir más o menos estas barras entre los elementos de combustible, se puede controlar el número de fisiones que se producen.

Las centrales nucleoeléctricas resultan competitivas comparadas con otras fuentes de producción de energía, ya que es relativamente poca la cantidad de combustible que necesitan, debido al elevado contenido energético del uranio enriquecido. Por ejemplo, una central nucleoeléctrica necesita 27 toneladas de combustible, mientras que harían falta 3950000 toneladas de carbón, 1668 millones de metros cúbicos de gas para generar la misma cantidad de energía anualmente [C1].

El poder energético de una pastilla de combustible cuyo peso sea de 10 grs. equivale a la de 3.9 barriles de combustoleo[C1].



Esquema de vasija conteniendo combustible[C1].

Figura 3.

1.3. DESCRIPCIÓN DE UNA CENTRAL NUCLEOELÉCTRICA.

Una nucleoeléctrica es una central térmica de producción de electricidad su principio de funcionamiento es esencialmente, la conversión de calor en energía eléctrica. Esta conversión se realiza en tres etapas (figura 4).

En la primera, la energía del combustible se utiliza para producir vapor a elevada presión y temperatura.

En la segunda, la energía de vapor se transforma en movimiento de una turbina.

En la tercera, el giro del eje de la turbina se transmite a un generador, que produce energía eléctrica.

La centrales nucleoeléctricas se diferencian de las demás centrales térmicas solamente en la primera etapa de conversión, es decir en la forma de producir vapor (figura 6). En las centrales convencionales el vapor se produce en una caldera donde se quema en forma continua el carbón combustoleo o gas natural.

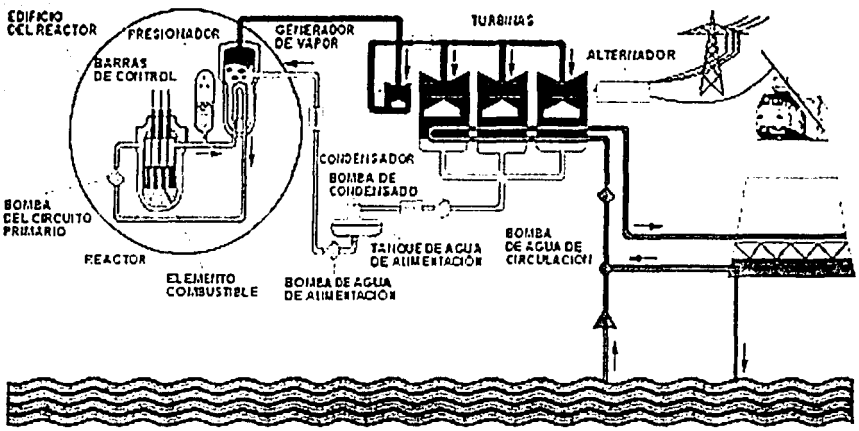
En las centrales nucleoeléctricas se tiene un reactor nuclear, que equivale a la caldera de las centrales convencionales. En las centrales nucleoeléctricas el calor se obtiene de la fisión del uranio que se le denomina combustible nuclear.

Una central nucleoeléctrica como la de Laguna Verde es constituida básicamente por seis edificios principales y otros secundarios [C1] .

Los seis edificios principales son:

a.-Edificio del reactor: alberga en su interior al reactor nuclear, sus sistemas auxiliares y los dispositivos de seguridad, la plataforma de recambio de combustible y la alberca de almacenamiento de combustible irradiado.

b.-Edificio del turbogenerador: aloja a las turbinas de alta y baja presión, el generador eléctrico y su exitador, el condensador, los precalentadores de agua de alimentación y los recalentadores de vapor.



Esquema de una central nuclear [C1].

Figura 4.

e.-Edificio de control: en su interior están el cuarto de control y la computadora de procesos, el cuarto de cables, los sistemas de aire acondicionado, el banco de baterías, los laboratorios radio-químicos.

d.-Edificio de generación de diesel: aloja los tres generadores de diesel que se utilizan para el suministro de energía eléctrica a los sistemas de refrigeración de emergencia.

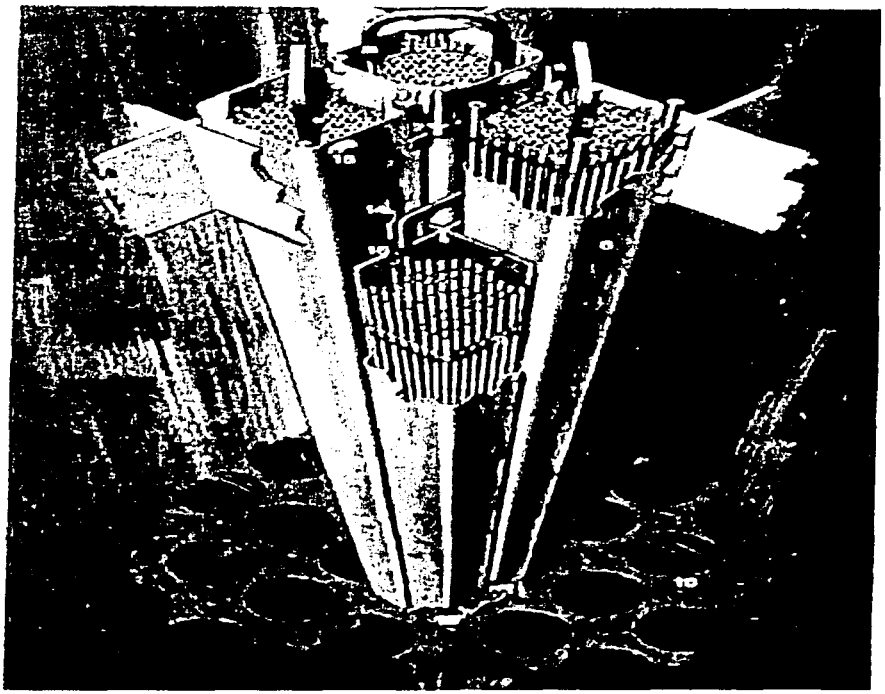
e.-Edificio de tratamiento de residuos radiactivos: aloja los sistemas de tratamientos de residuos sólidos, líquidos y gaseosos de mediano y bajo nivel de radiactividad.

f.-Edificio de planta de tratamiento de agua y de taller mecánico: contiene la planta de producción de agua desmineralizada de alta pureza para su uso en el ciclo de vapor. También contiene el taller mecánico para reparación de equipos y mantenimiento.

Los edificios secundarios son: toma de agua de enfriamiento para el condensador y los componentes nucleares; la subestación eléctrica, el edificio administrativo, el edificio de almacenamiento de partes de repuesto, el edificio de acceso, el edificio de almacenamiento temporal de residuos, el edificio de entrenamiento y el centro de información pública.

EL REACTOR NUCLEAR:

Un reactor nuclear consta de tres elementos esenciales: el combustible (figura 5), el moderador y el fluido refrigerante.



Ensambles de combustible nuclear [C1]

Figura 5.

El combustible.

El uranio se utiliza en forma natural que contiene 0.711% de uranio 235 o bien en forma de uranio enriquecido, al que artificialmente se eleva la concentración de uranio 235, hasta un 3 o 4%.

El uranio se coloca de forma de uranio metálico o de oxido de uranio (UO_2), normalmente con él se fabrican pequeñas pastillas cilíndricas de un poco más de 1 cm de diámetro y longitud.

El moderador.

El moderador no mitiga la reacción de fisión sino que la hace posible. Para que el choque de un neutrón con un núcleo de uranio 235 pueda producir una fisión, es preciso que la velocidad del neutrón sea del orden de 2 Km/ seg. Cuando el neutrón sale de un núcleo fisionado, lleva una velocidad de 20,000 Km / seg. y es necesario frenarlo. Esta es la función del moderador frenar neutrones sin absorberlos. El moderador debe de reunir ciertas condiciones: que tenga un peso atómico ligero, que no absorba neutrones y que tenga una elevada densidad atómica. Los moderadores más utilizados son el grafito, el agua ordinaria, el agua pesada y algunos líquidos orgánicos [C1].

El refrigerante.

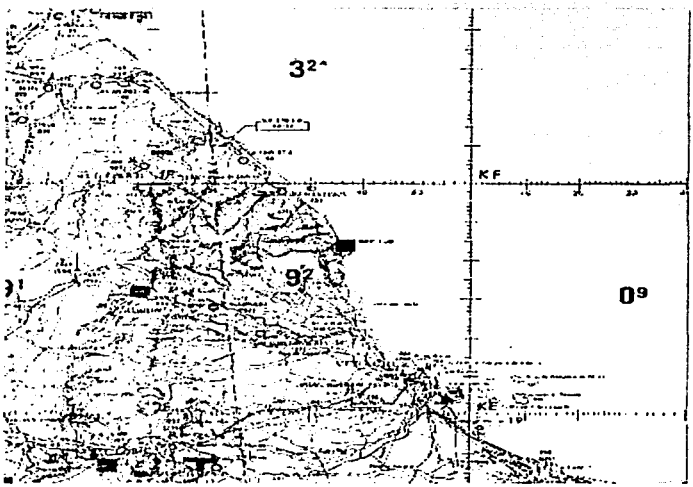
El fluido refrigerante circula entre las barras de combustible impulsado por una bomba. Debe de reunir una serie de condiciones para que pueda cumplir su función en forma satisfactoria: no capturar neutrones, tener un elevado calor especifico para remover la energía producida en el combustible nuclear y no ser corrosivo para los tubos y demás elementos del reactor.

CAPITULO 2

DISEÑO Y OPTIMIZACIÓN DE RECARGAS DE COMBUSTIBLE NUCLEAR.

2.1. DISEÑO DE RECARGAS DE COMBUSTIBLE NUCLEAR.

En el núcleo de algunos reactores (ejemplo; Central de Laguna Verde localizada en el estado de Veracruz, figura 6.) se tienen 444 ensambles de combustible que contienen 81 toneladas de Uranio cuya forma química es dióxido de uranio enriquecido al 1.87% de uranio 235 en promedio.



Localización de Laguna Verde[C1].

Figura 6.

También hay 109 barras cruciformes de control hechas de carburo de boro encapsulado en tubos y placas de acero inoxidable [C1].

El ciclo de cada reactor se diseña para operar entre 12 y 18 meses a plena potencia y al término de los cuales, la reactividad del núcleo llega a cero. Así que debe de realizar un recambio parcial de combustible que restituya la necesaria reactividad positiva en exceso. Generalmente los ensambles que se reemplazan son los que han agotado más su contenido de uranio 235.

Los ensambles nuevos de combustible que se introducen en el reactor dependen de la energía que se quiera generar durante los siguientes ciclos. Así como la frecuencia con la que se quieran realizar los recambios. En términos generales el objetivo fundamental en esté trabajo es encontrar un conjunto de valores de las variables independientes, que producirán la respuesta óptima.

En nuestro caso el modelo matemático que se plantea en la optimización de recargas contiene funciones matemáticas que van a ser utilizadas para representar el modelo real. Así mismo el método elegido de Algoritmos Genéticos, será utilizado para obtener la solución óptima.

Para diseñar una recarga se debe de tener definida la longitud del ciclo de operación, el quemado de descarga y los márgenes de diseño. La longitud del ciclo y el quemado de descarga se seleccionan buscando minimizar el costo del ciclo de combustible dentro de las restricciones impuestas por los requerimientos de operación del sistema.

Al diseñar el plan de recarga para un reactor se establece cuántos ensambles se necesitan, qué enriquecimiento deben de poseer, la colocación que tendrán en el núcleo y cómo se controlará el exceso de reactividad. Se debe respetar el límite de quemado de descarga para evitar fallas del combustible.

Determinar el arreglo que tendrán los ensambles combustible dentro del núcleo y la estrategia de control para satisfacer de la mejor forma los requerimientos y restricciones de la generación de la energía y de la distribución de la potencia es un problema complejo. Es teóricamente posible tener un arreglo que produzca una densidad de potencia uniforme. Para ello se propone que las propiedades del reactor sean uniformes excepto en el material fisil; realizar esto es impráctico, sin embargo se intenta.

En la región del núcleo del reactor se colocan los ensambles con un factor de multiplicación infinita promedio de uno; cerca de la periferia se necesita que la densidad de material fisil aumente. Esto es, se colocan cerca de la frontera los ensambles frescos y el resto de los ensambles se distribuyen en el centro de tal manera que se minimizan los efectos del pico de potencia local. En la frontera se colocarán los ensambles más quemados.

Un principio utilizado para diseñar las recargas es el de Haling, que establece que el factor de pico de potencia mínimo para un arreglo dado es el alcanzado operando el reactor de tal forma que el perfil de la distribución de la potencia no cambia apreciablemente durante el ciclo. Con este principio se define la estrategia de operación con la que se mantiene una óptima distribución de la potencia a lo largo del ciclo, se conserva un factor de pico de potencia mínimo y se soluciona el compromiso entre obtener el máximo de energía y la operación del reactor con un margen suficiente frente al limite térmico.

OPTIMIZACIÓN DE LAS RECARGAS.

Las compañías productoras de electricidad le conceden poca confianza a los resultados obtenidos hasta hoy y prefieren seguir empleando metodologías de diseño basadas principalmente en la experiencia, se han producido patrones adecuados pero no óptimos. Gracias al avance en la capacidad de las computadoras y en los códigos de física de reactores se están desarrollando nuevas herramientas de apoyo para los diseñadores de patrones de recargas.

Para lograr la optimización del diseño de las recargas que se instalan en el núcleo de un reactor se están desarrollando paquetes que aplican la técnica de programación lineal y no lineal, se producen códigos o sistemas que utilizan métodos basados en el conocimiento que emplean métodos de búsqueda heurística, se ha recurrido a redes neuronales, un método de ingeniería química conocido como recocido y la técnica de los algoritmos genéticos [L3].

El objetivo general de la optimización es encontrar un conjunto de valores de las variables independientes, sujetas a varias restricciones, que producirán la respuesta óptima deseada.

Un enfoque o procedimiento general de la optimización, que no se aplica necesariamente a todos los casos pero que se llevará a cabo con los patrones de recargas, puede ser el listado siguiente:

Elegir un sistema o sistemas para estudiar.

Definir un objetivo conveniente para el problema que se examina.

Examinar las restricciones impuestas al problema por causas externas.

Examinar la estructura del sistema o sistemas y sus interrelaciones.

Construir un modelo para el sistema. Esta es la etapa que permite que el objetivo sea definido desde el punto de vista de las variables del sistema.

Examinar y definir las restricciones internas impuestas sobre las variables del sistema.

Realizar la simulación expresando el objetivo desde el punto de vista de las variables del sistema, usando el modelo del sistema. Esta es la función objetivo.

Analizar el problema y reducirlo a sus características esenciales. En muchos casos esta reducción es necesaria para poder realizar la optimización.

Verificar que el modelo propuesto represente en los hechos el sistema que está siendo estudiado.

Determinar la solución óptima para el sistema y discutir la naturaleza de las condiciones óptimas.

Usar la información así obtenida, repetir este procedimiento hasta encontrar un resultado satisfactorio.

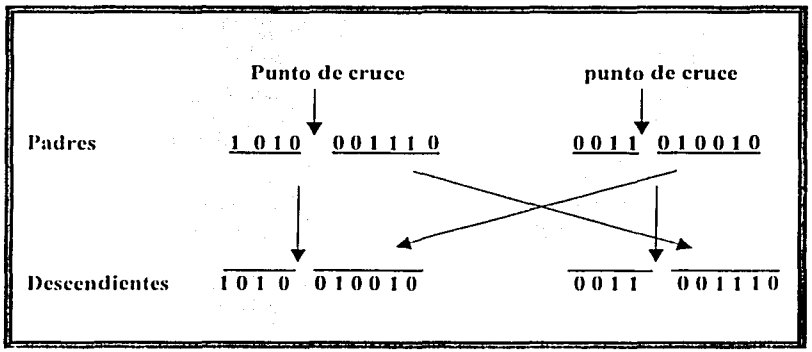
Dependiendo de la forma en que se expresa la función objetivo desde el punto de vista de las variables independientes y del conjunto de restricciones asociadas, se aplica la técnica de optimización adecuada.

Una técnica de optimización diferente a las que se manejan regularmente es la de los algoritmos genéticos [L3].

2.2. CARACTERÍSTICAS Y APLICACIONES DE LOS ALGORITMOS GENÉTICOS.

Los algoritmos genéticos son algoritmos de búsqueda estocástica basados sobre los mecanismos de selección y genética natural, practican en forma iterativa tres acciones principales: primera, las probables soluciones intercambian información de forma aleatoria. Segunda, introducen de forma aleatoria nueva información. Tercera, forman una nueva solución. Una de las ventajas de los algoritmos genéticos es que con ellos se pueden construir códigos computacionales utilizando tres operaciones fundamentales (cruzamiento, mutación y selección) tomando los individuos mejor adaptados (con mejores soluciones temporales), utilizando cruzamiento (figura 7) generamos nuevos individuos (nuevas soluciones) que contendrán parte del código genético (información) de sus dos antecesores, y aunque el nuevo individuo no tenga que estar forzosamente mejor adaptado.

Para representar los algoritmos genéticos, es necesario representar a los individuos en forma conveniente; cadenas binarias de ceros (0) y unos (1).



Operador de cruce basado en un punto.
Figura 7.

La aparición del digito "1" en la cadena binaria representa la existencia de una determinada característica y la aparición de un "0" representa ausencia. La cadena será tan grande como características o parámetros necesitemos para representar el individuo.

Para representar un algoritmo genético tomaremos como punto de referencia un modelo biológico. La idea básica es la siguiente: sí generamos un conjunto con algunas soluciones posibles y a cada una de estas soluciones les llamamos individuos (cada individuo tiene una información asociada a él) y a todo el conjunto le llamaremos población.

En un problema de optimización, a los algoritmos con variables libres se les asocia un valor para que la función sea mínima o máxima para esos valores. La función en el modelo biológico se denominará función de adaptación y determinará el grado de adaptación del individuo y a la información se le denominará código genético.

Las características de los individuos se van a denominar fenotipos y la información asociada al fenotipo se compone de dos partes indivisibles denominadas cromosomas.

Un fenotipo puede estar en más de un cromosoma, en cuyo caso puede ser que el hijo herede un fenotipo que no tenía ni el padre ni la madre, sino una combinación de ambos.

En caso de que el hijo tenga partes de los genes del padre y parte de los genes de la madre que intervienen en un fenotipo, se va a crear una característica nueva a este fenotipo [T22].

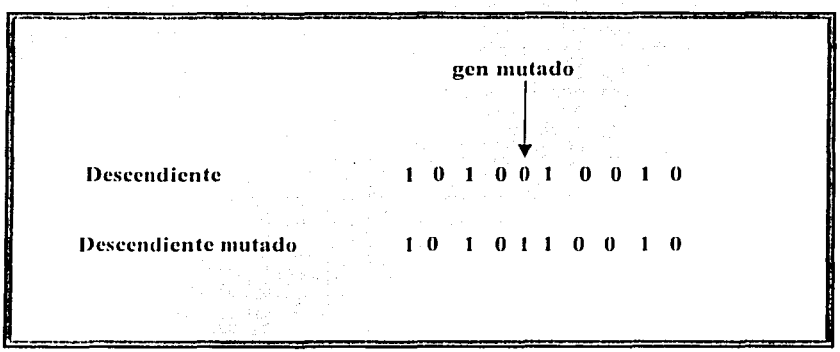
El promedio de la adaptación de la generación se mejorará ya que tiende a perpetuarse y a extenderse las mejores características, y a extinguirse las poco benéficas.

La mutación es definida como una variación de las informaciones contenidas en el código genético, habitualmente un cambio de un gen a otro producido por algún factor externo al algoritmo genético. En términos biológicos se definen dos tipos de mutaciones: las generativas que se heredan y las somáticas que no se heredan.

En los algoritmos genéticos creamos una población cuya función se mejorará globalmente y sólo aplicaremos las mutaciones (figura 8) que se heredan.

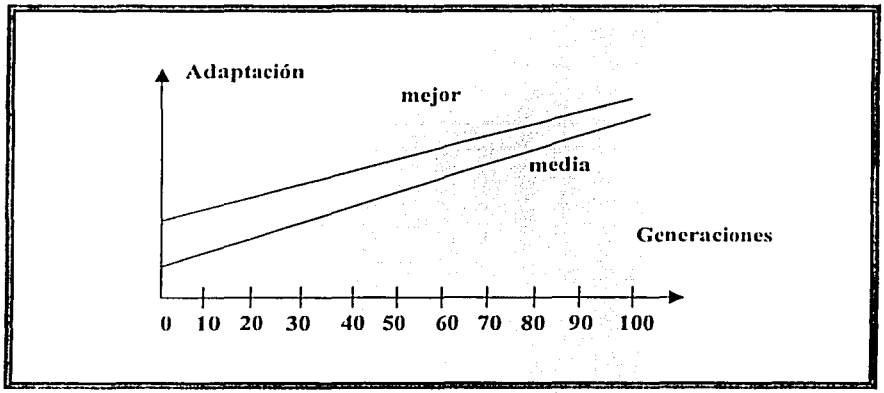
El elegir equivocadamente la forma de almacenar la información puede ocasionar que la convergencia sea más lenta, es decir que tardaremos más en encontrar la solución o que no converja en ninguna forma (figura 9), decir que la población esté errando aleatoriamente por efecto de los cruzamientos y mutaciones sin llegar nunca a un punto estable, esto es el fenómeno que se denomina deriva genética.

Los algoritmos genéticos presentan ventajas para determinado tipo de aplicaciones entre las cuales puede considerarse la optimización de planes de recarga de combustible.



Operador de mutación

Figura 8.



Adaptación media y mejor adaptación en un algoritmo simple.

Figura 9.

Algunas de las características de los algoritmos genéticos son [L4]:

a) -Son algoritmos de búsqueda estocásticos basados en mecanismos de selección y genética natural: dos ejecuciones distintas pueden dar dos soluciones distintas. Esto es útil por el hecho de que hay gran cantidad de soluciones válidas.

b) -Son algoritmos de búsqueda múltiple; dan soluciones múltiples, por ello nos podemos quedar con la solución que más nos convenga según la naturaleza de nuestro problema. En nuestro caso analizar los planes de recargas de combustible nuclear generados por el algoritmo para decidir cuales son los óptimos.

c) -Los Algoritmos Genéticos, hacen una barrida al subespacio de posibles soluciones válidas y son de los más exploratorios posibles.

d) -La convergencia del algoritmo genético es poco sensible a la población inicial si ésta se elige de forma aleatoria y es lo suficientemente grande.

e) -Por su grado de penetración casi nulo, la curva de convergencia asociada al algoritmo presenta una convergencia excepcionalmente rápida al principio, esto se debe a que el algoritmo genético es excelente descartando subespacios malos. Cada cierto tiempo la población vuelve a dar el salto evolutivo y se produce un incremento en la velocidad de convergencia excepcional. La razón de esto es que algunas veces aparece una mutación altamente benéfica, o un individuo excepcional que propaga algún conjunto de cromosomas excepcional al resto de la población.

f) -La optimización es función de la representación de datos. Este es el concepto clave dentro de los algoritmos genéticos, ya que una buena codificación puede hacer la programación y la resolución más sencillas.

g) -Los Algoritmos genéticos son intrínsecamente paralelos; esto significa que independientemente de que los hayamos implementado de forma paralela o no, buscan en distintos puntos del espacio de soluciones de forma paralela. Ese paralelismo intrínseco permite que sean fácilmente paralelizables, es decir, fácilmente de modificar el código para que se ejecute simultáneamente en varios procesadores.

h) -Al analizar simultáneamente varias regiones de todo el espectro de probables soluciones puede escapar de los óptimos locales en los problemas multinodales.

Las aplicaciones de los algoritmos genéticos pueden ser diversas ya que se pueden utilizar en diferentes áreas de investigación y desarrollo:

.En problemas de combinatoria.
.En casos particulares como la optimización estructural de agregados del silicio.
.Para la optimización de patrones de recarga para los reactores de agua hirviente (BWR), etc.

2.3. COMPONENTES DEL SISTEMA SOPRAG.

EL SIGNIFICADO DE SOPRAG ES [F1]:
Sistema de Optimización de Patrones de Recarga utilizando Algoritmos Genéticos.

El Sistema SOPRAG cuenta con diversas rutinas o programas donde se ejecutan las operaciones de los algoritmos genéticos (generación de poblaciones, cruzamiento, mutación y selección) además, cuenta con el simulador PRESTO (código de simulación del comportamiento del reactor nuclear en estado estable) así mismo cuenta con un evaluador denominado la función objetivo donde obtenemos los méritos o la calificación de los candidatos a ser elegidos.

Para ejecutar el sistema SOPRAG debemos de contar con archivos específicos que contengan historia de identificadores utilizados en ciclos anteriores y que podrían ser utilizados en el nuevo ciclo.

Al ejecutar en su totalidad el sistema SOPRAG encontraremos la optimización de los patrones de recarga para los reactores nucleares en este caso el reactor nuclear de agua hirviente " BWR " y en particular para el reactor de Laguna Verde.

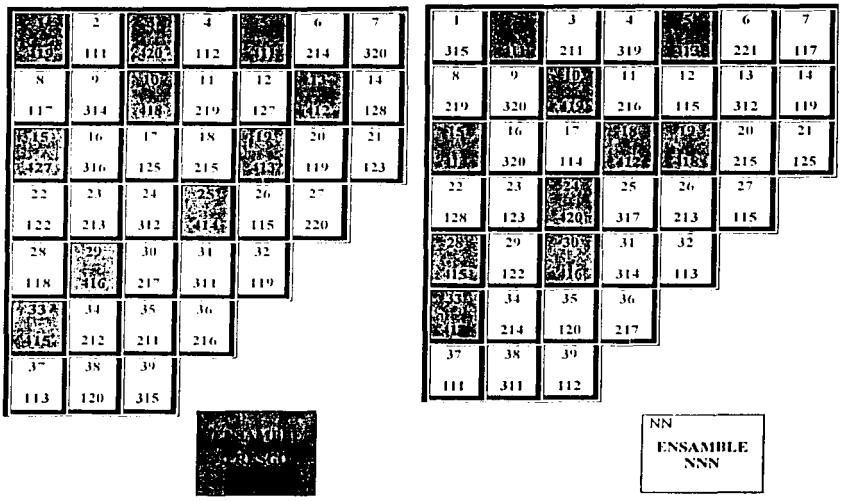
A continuación se describen los diferentes componentes del SOPRAG:

2.3.1. ALGORITMOS GENÉTICOS.

a) – POBLACIÓN INICIAL.

El tamaño de la población se debe de determinar como primer paso en SOPRAG; el sistema es capaz de manejar uno o dos tipos de combustible, y es necesario indicar el número total de ensambles de la recarga.

Para la primera generación del patrón de recarga (LP, por sus siglas en inglés) el programa inicia la distribución de ensambles frescos en una cuarta parte del núcleo. En forma aleatoria se selecciona una posición el núcleo y en ese lugar se instala el combustible nuevo, este proceso se repite hasta completar la distribución de 111 ensambles en el cuarto del núcleo, la figura 10. sé ilustran dos patrones de recarga.



NN = posición, NNN = identificador de ensamble.

Núcleos producidos colocando ensambles frescos y quemados de forma aleatorios.

Figura 10.

Dos restricciones se consideran en el sistema, una es que no se asigne combustible fresco en la periferia del núcleo (nunca debe de aparecer en ese lugar) y en la primera población los ensambles frescos no se pueden colocar en posiciones vecinas, excepto en la diagonal principal, conforme avance el proceso esas vecindades se pueden presentar. No se colocan en la periferia por su nivel de reactividad y por la forma del perfil de potencia que se presenta en esa región. Se opta también por no permitir la vecindad entre los ensambles al inicio para evitar la formación de picos de potencia altos.

Para la creación de los LP de la primera población primeramente se instalan aleatoriamente todos los ensambles nuevos en el núcleo; después se procede a seleccionar aleatoriamente los ensambles quemados que se instalarán en cada una de las posiciones vacías del núcleo, el proceso se repite para cada una de los patrones que formarán la población inicial.

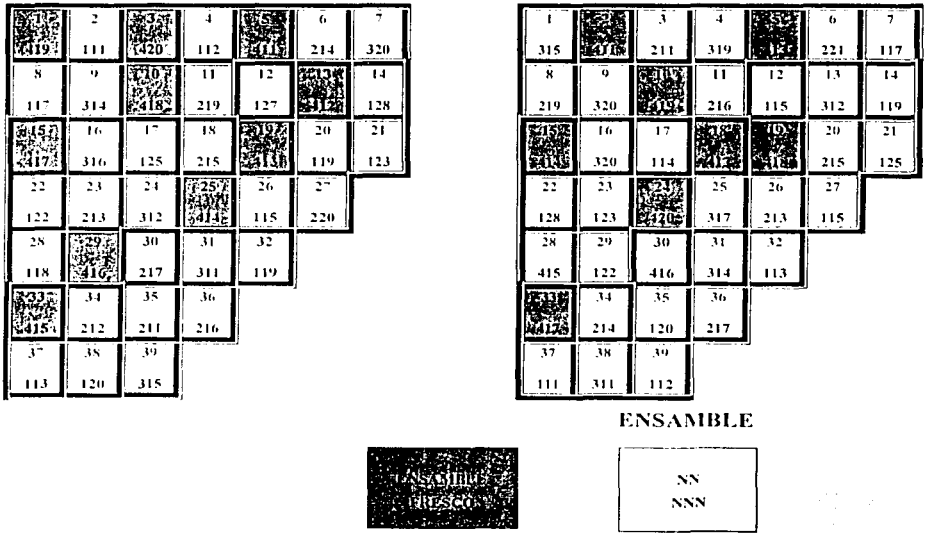
Cada elemento de la población inicial es evaluado y recibe una calificación dependiendo de su comportamiento. Los individuos de la siguiente población son elegidos de la población anterior por un procedimiento de selección. Un método menos complicado consiste en eliminar de la población aquellos individuos que no muestran un comportamiento satisfactorio mínimo y con los que quedan se procede a formar nuevos individuos aplicando operaciones genéticas.

b) - CRUZAMIENTO.

El cruzamiento es el operador genético de recombinación más importante. Bajo este operador dos individuos de la población intercambian porciones de su estructura mediante el siguiente procedimiento:

El proceso de cruzamiento puede ser de dos formas, utilizando un solo punto de cruce o dos puntos. Para ello se seleccionan de forma aleatoria, dos núcleos de la población y enseguida se obtiene también, en forma aleatoria los puntos de cruce.

Se determina al azar una posición de los individuos en donde se cortarán para realizar el intercambio como se ilustra en la figura 11, si se eligen dos puntos de cruce el procedimiento es similar al anterior, se eligen aleatoriamente dos puntos en donde los individuos se cortan.

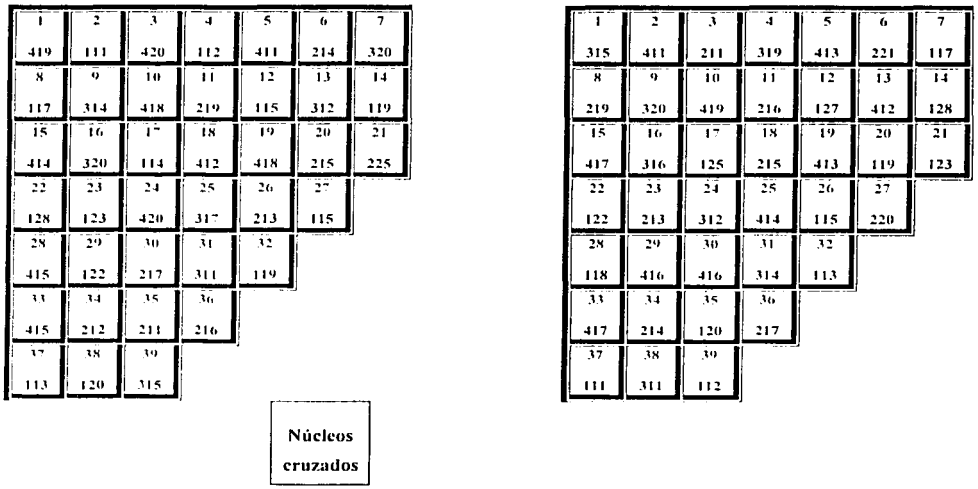


NN: posición, NNN: identificador de ensambles

Núcleos seleccionados aleatoriamente para cruzarse

Figura 11.

Al realizarse el cruzamiento, secciones de los núcleos se intercambian. Aleatoriamente se selecciona el primero y el segundo punto de corte; los ensambles que están antes del punto de corte y los que están después del segundo punto se intercambian entre ambos núcleos por ejemplo como se ilustra en la figura 12.



Núcleos cruzados seleccionando dos posiciones de corte (11,30)

Figura 12.

Al hacer los cruzamientos entre los núcleos el problema que se puede presentar es la repetición de ensambles quemados. La razón es que al instalarse aleatoriamente, cada ensamble queda en diferente posición en los núcleos y al cruzarse llega aparecer el mismo ensamble dos veces.

Para corregir este problema se realiza lo siguiente, en cada uno de los patrones de recarga se revisan las cadenas de identificación de los ensambles que aparecen, aun los repetidos y se construye una lista con los ensambles no utilizados. Teniendo la lista se buscan las parejas de los ensambles quemados no utilizados, se buscan las parejas de ensambles repetidos y uno de los dos se sustituye por el ensamble que aun no se instala y que más se le parece en reactividad.

c) – MUTACIONES.

El proceso de mutación se realiza después de haberse llevado a cabo los cruzamientos entre los distintos núcleos.

La mutación es un proceso aleatorio de los algoritmos genéticos. Todas las cadenas formadas mediante el cruzamiento tienen la posibilidad de ser mutadas en alguna parte de la cadena, sin embargo, no todas llegan a sufrir este cambio, se pude definir cuantas estructuras de la población completa serán mutadas.

Se tendrá una mutación cuando a una cadena seleccionada aleatoriamente se le cambia el carácter en una posición elegida al azar. Por ejemplo si se tiene una estructura de 8 caracteres enumerados del 0 a 7 y se localiza la posición cinco la cual será mutada, en esa posición aparece un digito uno "1" con la mutación, se le cambia el digito a cero "0" en la mutación se realizan pequeños cambios y se experimenta con otras regiones en la búsqueda de una solución más adecuada (figura 13).

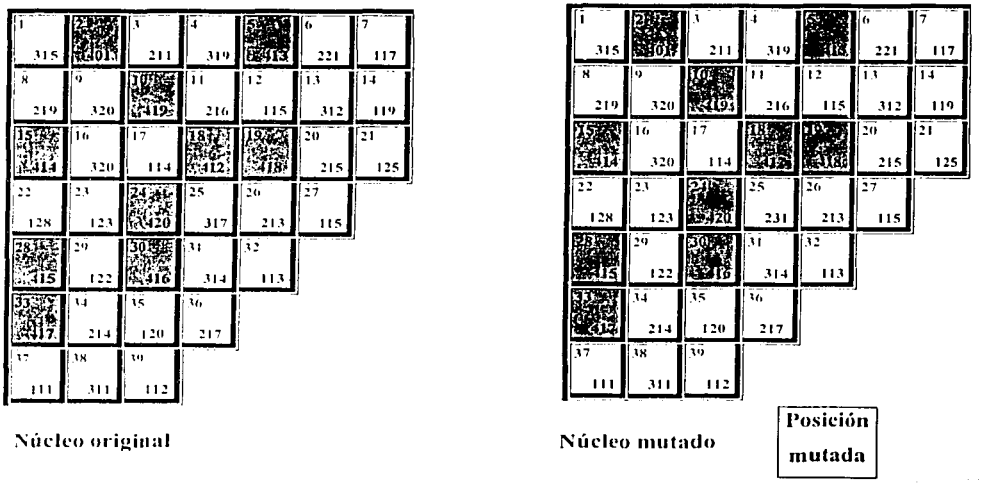
En esta operación se selecciona aleatoriamente un núcleo y se elige de la misma manera, una posición donde exista un ensamble con exposición, no se incluyen ensambles frescos.

En el lugar seleccionado se quita el ensamble quemado que existe y se le sustituye por un ensamble no utilizado; si hay disponibles, si no existieran ensambles sin instalar, el proceso inmediato será el de intercambiar dos posiciones con ensambles quemados.

El porcentaje aproximado de núcleos que se debe mutarse como se ilustra en la figura 15, estará alrededor de un 75 % y el proceso de mutación es esencial para evitar que

se presente la homogenización de los distintos núcleos durante los distintos ciclos de reproducción.

Se debe de evitar que en los ensambles frescos sean mutados porque si alguno se sustituye por un ensamble quemado se disminuye la posibilidad de supervivencia del diseño. Algún ensamble fresco puede ser cambiado de posición, pero no ocurrirá la mutación intercambiando dos ensambles nuevos.



Proceso de mutación en núcleo.

Figura 13.

Cómo se puede observar en la figura anterior se eligió un núcleo aleatoriamente para la mutación en este ejemplo la posición 25 en el lugar seleccionado se quita un ensamble quemado y se sustituye por un ensamble no utilizado o disponible.

2.3.2. EL SIMULADOR.

a)-El código de simulación PRESTO [F1] es un programa que simula el comportamiento del núcleo del reactor en estado estable. El reactor nuclear del tipo BWR opera en forma tridimensional con modelos neutrónicos y termohidráulicos integrados. El modelo neutrónico esta basado en una aproximación de la teoría de difusión de dos grupos de energía. El núcleo del reactor está modelado como un arreglo tridimensional de nodos cúbicos próximos, cada uno con propiedades internas homogéneas.

Las propiedades neutrónicas de cada uno de los nodos son descritas por un conjunto de secciones eficaces macroscópicas homogéneas para dos grupos de energía, representadas como tablas dependientes de la exposición del combustible, de los vacíos pesados por exposición o vacíos históricos y de vacíos instantáneos.

La operación del reactor puede ser simulada en las siguientes condiciones:

- En frío, crítico y subcrítico.
- En caliente, crítico a cero potencia.
- En caliente, operando en estado estacionario.
- En caliente, operando con transitorio por xenón.
- En caliente, con cálculo del paso de quemado.
- Cálculo del quemado de Haling.
- Cálculo para búsqueda de criticidad en flujo o en potencia.
- Búsqueda de patrones de barras de control para criticidad.
- Cálculo del margen de apagado.
- Generación de los albedos del reflector.

El código puede simular el comportamiento del núcleo del reactor con tres diferentes opciones:

Núcleo completo sin restricciones de simetría; un cuarto, medio o un octavo del núcleo con simetría de reflexión o rotacional.

Durante la búsqueda de la mejor selección, se opta por realizar las simulaciones con simetría rotacional de un cuarto del núcleo y utilizando la técnica de Haling para estimar la longitud del ciclo de operación en un solo cálculo de quemado, con el fin de tener resultados en un tiempo adecuado.

Como entrada el sistema necesita conocer la fracción del patrón de recarga, es decir, el número de ensambles nuevos que se van a insertar en el núcleo. Como se describió anteriormente el núcleo del reactor tiene 444 ensambles y el sistema trabaja únicamente con 111 ensambles para así reducir el orden del problema.

2.3.3. - FUNCIÓN OBJETIVO.

Para la asignación de las calificaciones a cada uno de los patrones simulados, se recurre a la función objetivo que relaciona la longitud del ciclo y el factor de pico de potencia, se busca maximizar la longitud del ciclo o quemado de combustible (Q) y disminuir el factor de pico de potencia (p) [F1].

La función objetivo es la siguiente:

$$f(Q, p) = (Q - w_1) * w_2 + (w_3 - p) * w_4$$

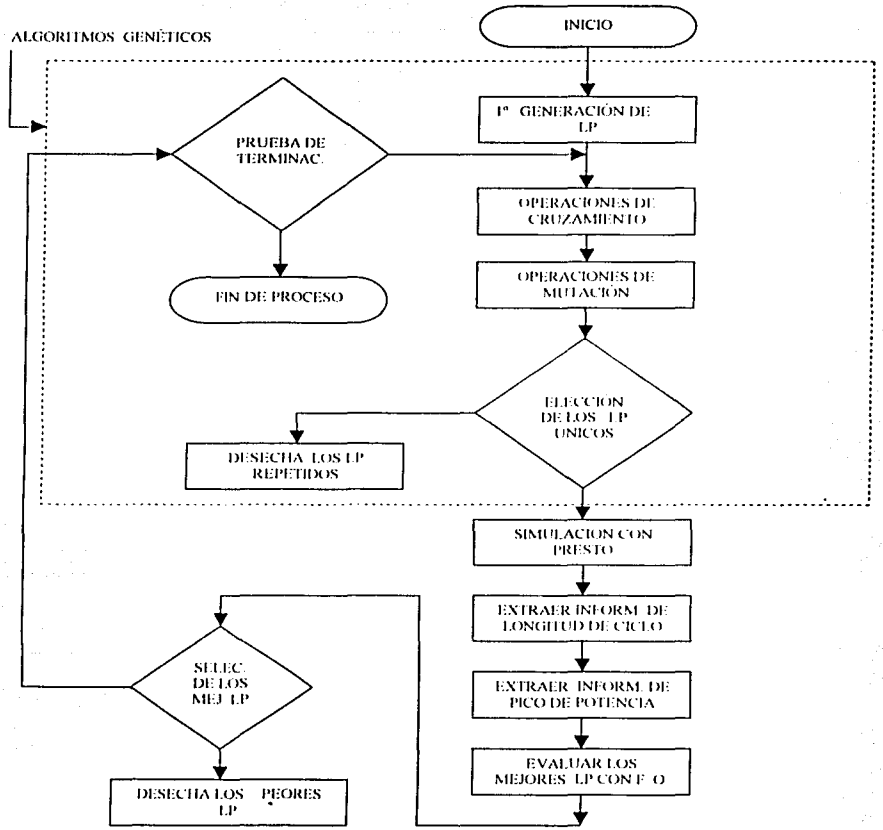
Dónde w_1, w_2, w_3, y w_4 son factores de peso que permiten dar mayor o menor importancia a alguna de las variables que se consideran en la búsqueda de los patrones de recarga óptimos.

Los factores w_1 y w_3 están relacionados con restricciones impuestas en el programa, si el diseño producido en la primera generación o en cualquier otra etapa de la búsqueda no alcanza un quemado mínimo de 5000 MWD/TMU * o su factor de pico de potencia rebasa el valor de 2.5, automáticamente el patrón diseñado se elimina de las opciones. De ahí que w_1 y w_3 tengan los valores de 5000 y de 2.5 respectivamente. Estos dos factores de peso, al igual que los otros, pueden cambiarse a voluntad dependiendo de los resultados obtenidos, sin embargo, se recomiendan conservarlos así.

Los factores w_2 y w_4 son más flexibles y modifican sustancialmente la calificación de un diseño.

* NOTA: MWD / TMU; tiene el significado de megawatts día por tonelada métrica de uranio.

2.4. DESCRIPCIÓN DEL SISTEMA SOPRAG.



DESCRIPCIÓN DEL SISTEMA SOPRAG.

a – Inicio.

b - Se genera la primera población de LP, está primera población será de 100 LP [L2].

c - Realizar las operaciones de cruzamiento de los LP .

d - Realizar las operaciones de mutación de los LP.

e - Seleccionar a los LP únicos y desechar a los LP repetidos.

f – Simular los LP con PRESTO.

g - Extraer la información de la longitud del ciclo de los LP.

h - Extraer la información del pico de potencia de los LP.

i - Realizar la evaluación de los mejores LP .

j - Seleccionar a los mejores LP y desechar a los peores LP.

k – Realizar prueba de terminación o iniciar nuevamente el ciclo en la operación de cruzamiento.

NOTA : Esta descripción hace referencia a la figura 2.4. Los incisos b, c, d, e, describen las operaciones que son realizadas con los algoritmos genéticos. Los incisos f, g y h son realizados en el simulador PRESTO. Los incisos a, i, j, k son complementarios del sistema. Todo el proceso en el sistema se desarrolla en forma secuencial.[1]

Para ejecutar el programa de optimización, se debe preparar el archivo INICIO.SOP donde se indica la cantidad de tipos de ensambles nuevos, el número de posiciones que se instalarán en una cuarta parte del núcleo, la cantidad de ensambles disponibles del ciclo anterior, el número de patrones que se producirán en la primera generación, la cantidad de ciclos de evolución que se permitirá estudiar, la cantidad de patrones que se quiere conservar hasta el final, el porcentaje y el tipo de mutaciones, finalmente el nombre del archivo donde se guardarán los mejores diseños resultado de la optimización.

Al iniciar la ejecución del sistema, la primera operación del sistema es leer toda la información que contienen los archivos de inicio, con los datos almacenados en la memoria del procesador se inicia la creación de las cadenas de identificación de los ensambles frescos o nuevos que se instalaran en cada núcleo. Una vez creados los identificadores, se les utiliza para crear la primera generación de núcleos.

La primera generación se construye distribuyendo en un cuarto del núcleo, a los ensambles frescos, uno a uno se colocan aleatoriamente en cualquier posición del núcleo, evitando instalarlos en la parte de la periferia o con vecinos del mismo tipo (frescos). Después de instalar los ensambles frescos se distribuyen los combustibles quemados: se determina como se colocarán los ensambles disponibles, y en que posición se colocarán, este proceso se repite hasta completar la cuarta parte del núcleo.

El proceso de formación de la primera generación se concluye cuando se tiene la cantidad estipulada de individuos de la población, por ejemplo 100.

A cada patrón de la primera generación se le asigna una calificación de acuerdo con el comportamiento mostrado, para ello se utiliza la función objetivo, que toma los valores de la simulación que corresponden a la longitud del ciclo y al factor de pico de potencia. Mientras más alto es el quemado y más bajo el factor de pico de potencia mayor es la calificación.

Con los patrones ordenados se realizan los cruzamientos, aleatoriamente se eligen dos diseños para cruzarlos y se les marca, aleatoriamente se determinan los puntos donde se cruzan y se procede a realizar el intercambio de ensambles. Al terminar del intercambio se realiza una revisión para sustituir a los ensambles repetidos en un mismo diseño. Se obtiene

por comparación con la lista de ensambles quemados, la relación de los ensambles que no se han instalado, después se compara cada posición en la cuarta parte del núcleo con las demás posiciones, para verificar que el ensamble no se haya repetido, se elige al que más se le asemeje en reactividad. Este paso se ejecuta para cada posición repetida y en ambos núcleos resultado del cruzamiento.

Para evitar la repetición de indentificadores de los ensambles nuevos, se le renombra; el primer ensamble que se encuentra se le completa su cadena de identificación con los dígitos 01, al segundo con los dígitos 02 y sucesivamente hasta terminar.

La mutación se realiza seleccionando de manera aleatoria una posición en la cuarta parte del núcleo, y de la misma forma se selecciona un ensamble quemado que no esté instalado y se realiza su sustitución.

La mutación se realiza intercambiando dos paquetes de combustible en posiciones diferentes y se evita que se intercambien dos ensambles nuevos del mismo tipo, al completarse el proceso de cruzamiento y mutación de los diseños, se simula cada uno de los patrones que resultó. Se obtiene la longitud del ciclo de quemado y el factor de pico de potencia y se utilizan en la función objetivo donde se les asigna la calificación correspondiente, con la calificación definida se inicia la selección.

La selección consiste en comparar la calificación de los patrones hijos con la generación anterior, y las calificaciones más altas se conservan para crear a la siguiente generación y las calificaciones más bajas se eliminan.

La repetición de cruzamiento, mutación, simulación de diseño y selección de patrones más aptos se repite hasta completar el número de generaciones que se estipula al inicio. De la ultima selección de patrones más aptos se toman los diseños de las calificaciones más altas y se guardan en un archivo para posteriormente ser extraídos y probados con un modelo de simulación más preciso.

Cuando se hace la simulación de cada uno de los diseños propuestos por el sistema, el código simulador PRESTO crea varios archivos que después serán utilizados.

En el archivo de la salida de SOPRAG, se guardan los mejores patrones, se anota también la distribución de los ensambles en la forma que los requiere PRESTO para

realizar la simulación, el sistema no lo hace de forma automática, lo debe de realizar un analista.

CAPITULO 3

PROCESAMIENTO EN PARALELO.

3.1. ANTECEDENTES DEL PROCESAMIENTO EN PARALELO.

Con las siguientes referencias podemos clasificar a las diferentes organizaciones de computadoras:

Simple flujo de instrucciones- simple flujo de datos (*SISD*).

En este tipo de organización las instrucciones se ejecutan secuencialmente pero pueden estar solapadas en las etapas de ejecución (segmentación encauzada) una computadora *SISD* puede tener más de una unidad funcional, todas las unidades funciónales están ligadas bajo supervisión de una unidad de control.

Simple flujo de instrucciones- múltiples flujos de datos (*SIMD*).

Esta clase corresponde a los procesadores matriciales, existen múltiples elementos de proceso (EP) supervisados por la misma unidad de control.

Todos los EP reciben la misma instrucción emitida por la unidad de control pero operan sobre diferentes conjuntos de datos procedentes de flujos distintos. El subsistema de memoria compartida puede contener múltiples módulos.

Múltiples flujos de instrucciones- simples flujos de datos (*MISD*).

Existen "n" unidades de procesadores, cada uno recibe distintas instrucciones que operan sobre el mismo flujo de datos y sus derivados. Los resultados (la salida) de un procesador pasan a ser la entrada (los operandos) del siguiente procesador.

Múltiples flujos de instrucciones – múltiples flujos de datos (*MIMD*).

Una computadora intrínseca implica iteraciones entre los "n" procesadores, porque todos los flujos de memoria se derivan del mismo espacio de datos compartidos por todos los procesadores.

Si el grado de iteraciones entre los procesadores es elevado, es fuertemente (estrechamente) acoplado. En caso contrario, si el grado de iteraciones entre los procesadores es pequeño se considera débilmente (ligeramente) acoplado.

La caracterización depende de la multiplicidad de sucesos simultáneos que ocurren en los componentes del sistema, conceptualmente las instrucciones y los datos se toman de los módulos de memoria. Las instrucciones se decodifican en la unidad de control, que envía el flujo de instrucciones decodificadas a las unidades de los procesadores para su ejecución. Los flujos de datos circulan entre los procesadores y la memoria bidireccionalmente.

3.2 ANÁLISIS DE LA MÁQUINA PARALELA.

El punto de partida para realizar el análisis de las máquinas paralelas, será la clasificación de los flujos de datos e instrucciones.

Para ejecutar un programa, un procesador necesita una fuente de instrucciones y una fuente de valores de datos (el número de flujos de instrucciones y el número de flujo de datos que la máquina emplea).

Las máquinas secuenciales y vectoriales tienen un solo flujo de datos, por otra parte las máquinas paralelas pueden tener varios flujos de instrucciones y varios flujos de datos o ambas cosas, por ejemplo:

Las tuberías (*pipelines*) o máquinas *MISD*.

Las tuberías son ejemplo de maquina *MISD*. En este caso, cada etapa de la tubería esta implementada por un procesador. El flujo de datos pasa de un procesador al siguiente y cada procesador tiene un flujo de instrucciones diferentes.

Las tuberías representadas en una máquina segmentada tienen varias etapas fijas y por ende no requieren flujos de instrucciones explícitos para cada etapa de la tubería. En lugar de ello los flujos de instrucciones se incorporan a la lógica empleada para implantar etapas de tuberías.

Con las tuberías se pueden implementar soluciones paralelas de varios problemas, no sólo interpretar instrucciones. Para describir un programa de tubería se tiene que especificar un programa para cada etapa de la tubería. Estos programas deben especificar cómo el procesador produce un valor de salida y un valor de entrada. También se tiene que describir la estructura del flujo de datos.

Programación Paralela de datos *SIMD*.

Las máquinas *SIMD* han dado lugar a un estilo de programación llamada programación paralela de datos. En la mayoría de la programación actual, un programa determina la forma en que el contador del programa se actualizará durante la ejecución. Una forma de proporcionar la ejecución en paralelo es con varios contadores de programa. Este tipo de método se le conoce como programación paralela de control y su forma de aplicar las instrucciones es aplicando las instrucciones de un mismo programa a varios valores de datos. En la estructura básica de una máquina *SIMD* el procesador de control ejecuta un programa paralelo de datos emitiendo instrucciones a los procesadores de datos. Después de emitir una instrucción a los procesadores de datos, el procesador de control espera a que todos los procesadores de datos hayan ejecutado la instrucción antes de emitir la siguiente.

Tres puntos importantes se deben de considerar en la programación paralela de datos:

1.- Si se tienen " n " (el número de elementos de un vector) y es demasiada grande, es probable que sea mayor que el número de procesadores de una máquina *SIMD*. Para evitar este problema, los entornos de programación de las máquinas *SIMD* proporcionan procesadores virtuales. Cada procesador de datos de la máquina *SIMD* simula varios procesadores virtuales, cuando se emite una instrucción a un

procesador de datos, ejecuta la instrucción varias veces, una por cada procesador virtual.

2.- La forma en la cual se ejecutan las estructuras de control se determina con los enunciados, el lazo "*for*" se utiliza para controlar las instrucciones que se envían a los procesadores.

Los enunciados "*if*" en el cuerpo del lazo "*for*" dependen de los datos locales de los procesadores (es decir, de los *PID*); estos enunciados controlan los procesadores que reciben las instrucciones emitidas.

3.- Las máquinas paralelas se pueden clasificar como síncronas o asíncronas.

Las máquinas síncronas ejecutan los programas en pasos discretos. Al iniciar cada paso se asigna una tarea a cada procesador. El paso concluye cuando todos los procesadores han completado sus tareas. Esta forma se le conoce también como sincronización barrera, ya que ningún procesador puede cruzar esta barrera hasta que todos los procesadores estén listos.

Las máquinas asíncronas no se basan en puntos de sincronización global. Es posible que los programas escritos para las máquinas asíncronas tengan que sincronizar las actividades de dos o más procesadores.

Máquinas sistólicas (*MIND* síncronas).

Si se generaliza la estructura de tubería de dos dimensiones, se obtiene una clase de máquinas llamadas sistólicas. Una tubería es una secuencia lineal de etapas de tubería. Cada etapa acepta un valor de entrada y produce un valor de salida que sirve como entrada de la siguiente etapa. Cada nodo de una máquina sistólica acepta dos valores de entrada y produce dos valores de salida.

La descripción de un programa de máquina sistólica es similar a la descripción de un programa de tubería. Hay que especificar un programa para cada procesador de la matriz, así como especificar la estructura de los flujos de datos.

Máquinas de flujos de datos (*MIMD* asíncronas).

En un programa de flujo de datos se especifica cómo se transforman sus valores de entrada a valores de salida, es decir, cómo fluyen los datos de las entradas a las salidas. Un diagrama de flujo de datos es un grafo dirigido. Los nodos que no tienen aristas de entrada, ni aristas de salida denotan entradas y salidas del programa. Los nodos de entrada se conocen cómo nodos fuente y los nodos de salida se conocen como nodos sumidero. Los demás nodos representan operaciones. Los arcos que conectan los nodos representan las trayectorias que siguen los valores de datos durante la ejecución del programa.

El aspecto importante de los diagramas de flujos de datos es que exponen gran cantidad de paralelismo de bajo nivel. Durante la interpretación del diagrama de flujo, varios operadores pueden tener sus valores de entrada requeridos, estos nodos operadores se pueden ejecutar en paralelo, es factible que la ejecución de un diagrama de flujo de datos se base en marcas (*tokens*) y disparos de operador.

Los programas de las máquinas paralelas se dividen en granos (tamaño de la operación que realiza cada procesador antes de que tenga que sincronizar sus actividades con otros procesadores) o con otro concepto podemos decir el programa se divide en granos para su ejecución en paralelo.

Las máquinas paralelas se dividen en tres categorías de acuerdo con su granularidad, grano fino, grano mediano y grano grueso.

Los modelos de flujo de datos y *SIMD* se basan en la ejecución paralela de grano fino. En cada paso, el procesador *SIMD* ejecuta una sola instrucción (a nivel de máquina). La unidad de ejecución en el modelo de flujo de datos también es una sola operación (a nivel de máquina). En cambio las máquinas sistólicas se basan en la ejecución paralela de grano mediano. En cada paso, los procesadores de una máquina sistólica ejecutan un programa relativamente pequeño, por lo general del orden de unos cuantos enunciados. Otras máquinas paralelas se basan en la ejecución paralela de grano grueso, en ellas los procesadores ejecutan programas relativamente independientes y no sincronizan sus actividades.

3.3. USO Y APLICACIÓN DEL MULTIPROCESAMIENTO.

USO DEL MULTIPROCESAMIENTO.

En los procesos cooperativos, cada procesador recibe un programa secuencial, denominado proceso, para su ejecución. No hay sincronización implícita entre los procesos y distintos procesadores pueden recibir un programa diferente para ejecutarlo, teniendo una memoria global o una memoria compartida.

Como no hay sincronización implícita, cualquier sincronización entre los procesos debe estar explícita en el código. En este caso se utilizarán semáforos para expresar la sincronización, los semáforos serán manipulados por dos operaciones, la de espera y la de señal (conocidas como *P* y *V*) cuando un proceso ejecuta la operación de espera, queda bloqueado, esperando una señal de otro proceso.

Las señales tienen memoria, de manera que sí ya se envío la señal, el proceso que ejecutó la espera no queda bloqueado. Cada operación de señal desbloquea un proceso.

Cuando los procesadores comparten una memoria común, cada proceso puede tener acceso directo a los elementos del vector que necesiten. Así tenemos tres métodos para considerar la estructura de memoria de un multiprocesador [H3]:

Memoria físicamente compartida, memoria lógicamente compartida y físicamente distribuida y memoria distribuida.

En un multiprocesador con memoria físicamente compartida, todos los procesadores comparten una memoria común. Los procesos que se ejecutan en procesadores diferentes pueden comunicarse leyendo y escribiendo valores en la memoria compartida.

En un multiprocesador de memoria distribuida, cada procesador tiene su propia memoria. Los procesadores no pueden tener acceso directo a memoria de otros, por lo cual los procesos que se ejecutan en otros procesadores deben de intercambiar mensajes para comunicarse entre sí.

Los multiprocesadores con memoria lógicamente compartida y físicamente distribuida, es estas máquinas, cada procesador tiene una memoria local (la memoria está

físicamente distribuida) a la que otros procesadores pueden tener acceso directo (la memoria está lógicamente compartida).

Memoria físicamente compartida.

Los multiprocesadores con memoria físicamente compartida ofrecen un entorno de programación conveniente pero no escalan bien. Para aprovechar con eficiencia la multiprogramación, el programador / compilador sólo tiene que dividir la porción de control de programa en procesos independientes que manipulan las estructuras de datos compartidos. Estas estructuras de datos se almacenan en la memoria compartida, donde cualquiera de los procesadores puede tener acceso a ellas. Sin embargo, esta estructura sólo puede apoyar sistemas con decenas de procesadores. Cuando se añaden demasiados procesadores al sistema, la memoria compartida se convierte en un cuello de botella, muchos procesadores estarán inactivos a la espera de realizar una operación en el canal de memoria compartida.

Hasta cierto punto, una memoria *caché* en cada procesador puede superar la inactividad de la memoria compartida. Las memorias *caché* reducen la cantidad de operaciones de memoria efectuadas por cada procesador, reduciendo así la contienda por la memoria compartida. La memoria *caché* aumentará el número de procesadores que pueden conectarse a la memoria compartida, la contienda por la memoria sigue limitando el número de procesadores que pueden usarse con esta estructura.

Memoria lógicamente compartida (*NUMA*).

Para evitar la inactividad, se han diseñado máquinas con memoria lógicamente compartida y físicamente distribuida. En lugar de tener una memoria físicamente compartida, la memoria se divide y se distribuye entre los procesadores. Toda la memoria se sigue direccionando con un mismo conjunto de direcciones, por lo tanto cualquier programa escrito para un multiprocesador de memoria compartida podría ejecutarse en este tipo de máquina.

La memoria asignada a un procesador se denomina memoria local, el resto es remota. Hay una diferencia importante en el tiempo requerido para un acceso a la memoria local y el tiempo necesario para un acceso a la memoria remota. Por esta razón, estas máquinas se conocen como máquinas de acceso no uniforme a memoria (*NUMA*). Mientras que las máquinas con memoria físicamente compartida se denominan máquinas con acceso uniforme a memoria (*UMA*).

Memoria distribuida (*NORMA*).

En un multiprocesador de memoria distribuida, todo acceso a la memoria asociada con un procesador debe efectuarse a través del procesador. Estas máquinas no ofrecen acceso directo a la memoria remota (la memoria asociada con otro procesador) se les conoce máquinas sin acceso a memoria remota (*NORMA*). Para tener acceso a la memoria de otro procesador, el procesador que inicia la solicitud debe de interactuar con el procesador remoto.

APLICACIÓN DEL MULTIPROCESAMIENTO.

El multiprocesamiento, conduce al concepto de redes con varios procesadores localizados en un mismo edificio y se llama Red de Área Local (*LAN*) o a una Red de Área Extendida (*WAN*). Un punto importante, es la capacidad para aumentar el rendimiento del sistema gradual a medida de que crece la carga es aumentando más procesadores.

Las compañías privadas y gobiernos ofrecen servicios de redes a cualquier organización que desee suscribirse a ellas. La subred es propiedad de la compañía operadora de redes y proporciona un servicio de comunicación para los clientes y terminales. A este tipo de sistema se le llama red pública y es análoga o forma parte del sistema telefónico público.

Las redes públicas, en diferentes países, son en general muy diferentes en cuanto su estructura interna, todas ellas utilizan el modelo de referencia de Interconexión de Sistemas Abiertos (*OSI*) por sus siglas en inglés y las normas del grupo o Comité Consultivo

Internacional telegráfico y telefónico (*CCITT*) de sus siglas en francés o los protocolos de OSI para todas las capas.

La Red de la Agencia de Proyectos de Investigación Avanzada (*ARPANET*) es la creación de *ARPA* ahora conocida como *DARPA* que es la Agencia de Proyectos de Investigación Avanzada (de la Defensa) correspondiente al Departamento de los Estados Unidos.

ARPANET tiene protocolos con características que cubren el mismo campo que el cubierto por los protocolos *OSI* de red y transporte. El protocolo de red, denominado protocolo entre redes (*IP*) es un protocolo sin conexión y se diseñó para manejar la interconexión de un número grande de redes *WAN Y LAN*, incluyendo a las de *ARPA*.

El protocolo de transporte en *ARPANET*, es un protocolo orientado a conexión denominado protocolo de control de transmisión (*TCP*) que en general se parece al protocolo de transporte del modelo *OSI*.

El modelo de referencia *OSI*.

El modelo *OSI* (figura 14), se define así porque se refiere a la conexión de sistemas heterogéneos, es decir, a sistemas dispuestos a establecer comunicación con otros distintos.

El modelo *OSI* tiene siete capas que son las siguientes:

*Capa física.

La capa física se ocupa de la transmisión de los bits a lo largo de un canal de comunicación.

*Capa de Enlace.

La capa de enlace tiene la tarea de: A partir de un medio de transmisión común, transformarlo en una línea de transmisión sin errores para la capa de red.

*Capa de red.

La capa de red se ocupa del control de la operación de la subred, es la determinación sobre cómo encaminar los paquetes del origen al destino.

***Capa de transporte.**

La función principal de la capa de transporte consiste en aceptar los datos de la capa de sesión, dividirlos en unidades muy pequeñas, pasarlos a la capa de red y asegurar que todos ellos lleguen correctamente al destino.

***Capa de sesión.**

La capa de sesión permite que los usuarios de diferentes máquinas puedan establecer sesiones entre ellos. A través de una sesión se puede llevar a cabo un transporte de datos ordinario.

***Capa de presentación.**

La capa de presentación se ocupa de los aspectos de sintaxis y semántica de la información que se transmite.

***Capa de aplicación.**

La capa de aplicación contiene una variedad de protocolos que se necesitan frecuentemente para la comunicación.

Puede existir confusión al definir una red de computadoras y un sistema distribuido. Un sistema distribuido tiene múltiples computadoras que son transparentes al usuario, el usuario puede teclear un comando para correr un programa y observar que corre. El usuario del sistema distribuido no tiene conocimiento de que hay múltiples procesadores, más bien se ve el sistema como un monoprocesador virtual. La asignación de trabajos al procesador y archivos a discos, el movimiento de archivos y todas las demás funciones del sistema deben de ser automáticos.

En una red de computadoras, el usuario debe explícitamente entrar en una máquina, explícitamente enviar trabajos remotos, explícitamente mover archivos y por lo general gestionar de manera personal toda la administración de la red, y con un sistema distribuido nada se tiene que hacer de forma explicita, todo lo hace de forma automática el sistema.

Un sistema distribuido es un caso especial de una red, aquel cuyo *software* da un alto grado de cohesividad y transparencia. Por lo tanto la diferencia marcada entre el sistema distribuido y una red, es el sistema operativo.

Un multiprocesador puede emplear una red de interconexión para emitir señales de control. Por ejemplo, una máquina *SIMD* requiere una red de interconexión para transmitir instrucciones e indicar cuando todos los procesadores de datos han completado sus tareas.

Los sistemas Multiprocesadores distribuidos se pueden clasificar de acuerdo con su tamaño físico [H2].

1.-Se encuentran las máquinas de flujos de datos, que son computadoras con un alto nivel de paralelismo y muchas unidades funcionales trabajando en el mismo programa.
2.-Los multiprocesadores, que son sistemas que se comunican a través de memoria compartida.
3.-Las redes, son computadoras que se comunican por medio del intercambio de mensajes.
4.-La conexión de dos o más redes, se conoce como interconexión de redes.

Las redes de interconexión de procesadores se construyen a partir de conmutadores y enlaces. Los enlaces transportan los mensajes a través de la red, y los conmutadores dirigen los mensajes a los distintos enlaces en la red. Las diferentes estrategias de red se distinguen por su topología, o sea, la relación entre los conmutadores y los enlaces.

Para la aplicación del multiprocesamiento se tomaran como base cuatro topologías de red [H1]:

Topología de canal compartido.

Un canal compartido consiste en un solo enlace y un conmutador sencillo para cada procesador y cada módulo de memoria es conectado a la red. Es fácil disponer con está estrategia de la comunicación entre procesadores y entre el procesador y la memoria.

Topología totalmente conectada y el conmutador de barras cruzadas.

En una red totalmente conectada, cada procesador tiene un enlace directo con los demás procesadores. Cuando se usa una red totalmente conectada, no hay ninguna contienda en la red porque las trayectorias de cada procesador son independientes de las de los demás procesadores. Sin embargo, cada procesador debe de tener un conmutador que pueda enviar mensajes a otros procesadores de la red.

Topología de hipercubo.

Un hipercubo es una topología recursiva. El hipercubo más simple consiste en un solo nodo, éste es llamado hipercubo de grado cero, si se tienen dos hipercubos de grado "n", puede construirse un hipercubo de grado n + 1 conectando los nodos correspondientes en estos dos hipercubos. Entre las propiedades de los hipercubos hay varias de interés:

Cada procesador de un hipercubo de grado n (n diferente de cero) está conectado directamente a otro n - 1 procesadores.

Todo hipercubo de grado n tiene 2^n nodos.

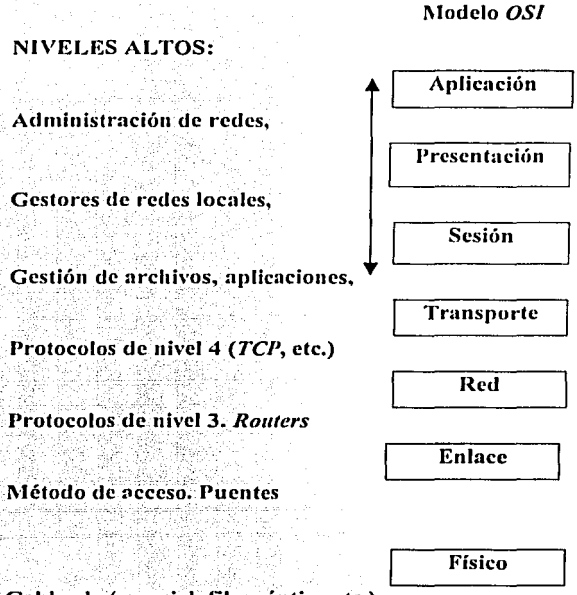
Todo hipercubo de grado n tiene n(n- 1) / 2 enlaces.

Cuando un procesador necesita enviar un mensaje a otro procesador en el hipercubo, tal vez tenga que encaminar al mensaje por otros procesadores para que llegue a su destino. Un algoritmo de encaminamiento especifica cómo se construye esta trayectoria. La longitud de la ruta o trayectoria es el número de procesadores en ella, excluyendo al procesador que originó el mensaje. En un hipercubo de grado n siempre hay una trayectoria con longitud de n o menor entre dos procesadores del hipercubo.

Topología de mariposa.

Una red de mariposa simple se construye con conmutadores que tienen dos entradas y dos salidas. En este caso la red conecta un conjunto de procesadores con igual número de

módulos de memoria, no todos los conmutadores de la red de mariposa están relacionados con los procesadores (ni con los módulos de memoria).



Modelo de transmisión de datos mediante el modelo de interconexión de sistemas abiertos (OSI).

Figura 14.

3.4. MODELOS, LENGUAJES PARALELOS Y COMPILADORES.

MODELOS.

Los modelos de programación paralelos.

Un modelo de programación es una colección de programas abstractos que proporcionan a un programador una simplificación y vista transparente del sistema de *hardware* y *software* de la computadora.

Los modelos de programación paralelos son específicamente diseñados para multiprocesadores, multicomputadoras, o para las computadoras de vector *SIMD*.

Cinco modelos caracterizan a esta variedad de computadoras que aprovechan el paralelismo con diferentes paradigmas de ejecución, los cuales son:

3.4.1 Modelo de variables compartidas.

3.4.2 Modelo de intercambio de mensajes.

3.4.3 Modelo de datos paralelos.

3.4.4 Modelo orientado a objetos.

3.4.5 Modelo de programación lógica y funcional.

3.4.1. Modelo de Variables compartidas.

En todos los sistemas de programación se considera a los procesadores recursos activos, la memoria y dispositivos de entrada y salida (*I/O*) los recursos pasivos.

Las unidades básicas de computación en un programa paralelo de procesos se denota con el funcionamiento realizado por los segmentos de código relacionado. La granularidad de procesos varia en los modelos de la programación y en sus diferentes aplicaciones.

Como un programa es una colección de procesos, el paralelismo depende de cómo se lleve a cabo la comunicación entre procesos (*IPC*). El problema fundamental en la programación paralela gira entorno de su especificación, creación, suspensión, reactivación, terminación, y sincronización de procesos concurrentes que residen en uno o en diferentes procesadores. Se puede limitar el alcance y acceso, el direccionamiento del proceso puede ser compartido o restringido.

Para garantizar el orden en la comunicación entre procesos se requiere de la propiedad de la mutua exclusión, ya que está permite un acceso exclusivo de un objeto compartido por un proceso en un instante de tiempo.

La programación en un multiprocesador de grano fino (*MIMD*) se basa en el uso de variables compartidas, utilizando memoria común entre la comunicación de los procesos. Las variables compartidas demandan el uso de memoria compartida y de mutua exclusión para accesar a través de múltiples procesos en el mismo conjunto de variables.
Una sección crítica (SC) es un segmento del código que accesa a las variables compartidas, el cual se debe de ejecutar en un proceso en un instante de tiempo y una vez iniciado, sólo puede ser detenido con una interrupción. En otros términos, el funcionamiento de una SC es indivisible y satisface los requisitos siguientes:

* Mutua exclusión. Solamente se puede ejecutar un proceso en SC en un instante de tiempo.
* No bloqueo en la espera. No hay esperas circulares, por dos o más procesos que intenten entrar en la SC; por lo menos uno tendrá éxito.
* No prioridad. No existirá interrupción, una vez entrando en la SC.

Entrada eventual. Un proceso que intente entrar en la SC tendrá éxito posteriormente.

* Acceso protegido. el principal problema asociado con el uso de una SC está en evitar las condiciones de velocidad, cuando los procesos concurrentes se ejecutan en órdenes diferentes y producen resultados diferentes. La granularidad de una SC afecta la ejecución. Si el límite de un SC es demasiado grande, puede limitar el paralelismo debido a la espera excesiva para competir por los procesos.

Cuando SC es demasiada pequeña, puede ser innecesaria y se vuelve compleja al *software*. La solución es acortar la carga de SC o el uso condicional para mantener una ejecución equilibrada.

El uso del multiprocesador con memoria compartida, utiliza las variables-compartidas para las comunicaciones entre procesos. El multiprocesador toma varias formas, dependiendo del número de usuarios y la granularidad en la división de los cálculos. Cuatro modos de operación son utilizados para programar los sistemas multiprocesador; se especifican a continuación:

Multiprogramación.

Tradicionalmente la multiprogramación se define como múltiples programas independientes que corren en un solo procesador o en un multiprocesador, compartiendo los recursos del sistema en un tiempo determinado. Un multiprocesador puede usarse para resolver un problema grande o correr múltiples programas a través de procesadores.

Un multiprograma es un multiproceso que permite a múltiples programas correr concurrentemente a través del tiempo compartido de todos los procesos en el sistema. Así múltiples programas son permitidos para entrar en la unidad central de procesamiento (*CPU*) y activar los dispositivos de entrada y salida (*I/O*). Cuando un programa entra al modo de *I / O*, el procesador cambia a otros programas. Entonces la multiprogramación, no está restringida para un multiprocesador.

Multiprocesamiento

Cuando multiprogramamos estamos definiendo, cómo los procesos son manejados por un multiprocesador. Esto es llamado multiprocesamiento.

Si la comunicación entre procesos es manejada al nivel de la instrucción, el multiprocesador opera en el modo de *MIMD*. Si la comunicación entre procesos se maneja al nivel del programa, subprograma, o a nivel procedimiento, la máquina opera en modo *MPMD* (múltiples programas - múltiples flujos de datos).

En otros términos, definimos al multiprocesamiento *MIMD* si es de grano fino y el paralelismo está al nivel de la instrucción y si el multiprocesamiento es *MPMD* es de grano grueso y el paralelismo esta en el nivel procedimiento. En ambos modos los multiprocesadores, usan variables compartidas para lograr la comunicación entre procesos.

Multitareas.

Un solo programa puede dividirse en múltiples tareas y ejecutarse en un multiprocesador concurrentemente. Esto se ha llevado a cabo en un multiprocesador *Cray*. Así las multitareas se ejecutan en forma paralela en dos o más partes de un programa. Así el trabajo se realiza eficazmente y requiere menos tiempo de ejecución.

Las multitareas se logran agregando partes de código en el programa original para proporcionar la unión apropiada y sincronización de la división de tareas. La diferencia entre las multitareas y no multitareas se debe a que a veces no todas las partes de un programa pueden ser divididas en tareas paralelas, por consiguiente, se debe de analizar el código antes de dividir el programa.

Multilecturas.

El sistema operativo *UNIX* tradicional, tiene una sola línea en el núcleo (*kernel*) y sólo un proceso puede recibirse en el *kernel* del sistema operativo y proporcionar un servicio en un instante de tiempo. En un multiprocesador podemos extenderlo para que se utilice en multilecturas. El propósito es permitir las multilecturas de procesos ligeros compartiendo el mismo espacio de dirección y ser ejecutados simultáneamente por los mismos o diferentes procesadores.

El concepto de multilecturas es una extensión de los conceptos de multitareas y multiprocesamiento, el propósito es de explotar el paralelismo de grano fino en modernos multiprocesadores para construir un múltiple contexto de procesadores o procesadores súper escalares con múltiples instrucciones, cada lectura puede usar a un contador de

programa por separado. Los conflictos de recursos son el problema mayor a ser resuelto en una arquitectura de multilecturas.

Los niveles de sofisticación dan seguridad a la coherencia de los datos y preservan el orden de los eventos incrementándose desde la programación única, para multitareas, multiprogramación, multiprocesamiento y multilecturas, en ese orden. Deben desarrollarse direcciones de memoria y mecanismos de protección especiales para asegurar exactitud e integridad de los datos para la operación de las lecturas paralelas.

Particionado y Duplicando.

La meta del procesamiento paralelo es la explotación del paralelismo tanto como sea posible. La partición del programa es una técnica para descomponer un programa grande en un conjunto de datos con partes muy pequeñas para la ejecución paralela en múltiples procesadores.

La partición del programa involucra la programación y compilación. La detección del paralelismo puede ser utilizado por diversos usuarios y se expresa explícitamente en la construcción del lenguaje paralelo, las técnicas de reestructuración del programa son usadas para transformar los programas secuenciales en una forma paralela más conveniente para los multiprocesadores. Esta transformación debe llevarse a cabo automáticamente por un compilador.

La duplicación del programa se refiere a la repetición del mismo código del programa para la ejecución paralela en los múltiples procesadores sobre diferentes conjuntos de datos.

La partición a menudo se ofrece prácticamente en un sistema multiprocesador de memoria compartida, mientras la repetición es más conveniente para memoria distribuida en multicomputadoras que utilizan el paso de mensajes.

Planeación y Sincronización.

El planear dividir en módulos un programa sobre procesadores paralelos es mucho más complicado que la planeación de programas secuenciales sobre un único procesador. La planificación estática está dirigida a un tiempo posterior de la compilación. Esta ventaja es poco utilizada principalmente porque no es muy conocida.

La planificación dinámica requiere contexto de cambio rápido, prioridad, y mucho más apoyo del SO. Las ventajas de planificación dinámica incluyen la utilización de mejores recursos. Pueden usarse mejores métodos estáticos y dinámicos conjuntamente en un sistema del multiprocesador demandando una alta eficiencia.

En un sistema *UNIX* convencional, la comunicación entre procesos (*IPC*) se dirige al nivel del proceso. Los procesos pueden ser creados por cualquier procesador. Todos los procesos accesan asincrónicamente a los datos compartidos y pueden ser protegidos solamente uno en cada ocasión, permitiendo acceder los datos compartidos en un solo instante de tiempo. Esta es la propiedad de la mutua exclusión que es utilizada con el uso de cerraduras, semáforos y monitoreos.

Al nivel de control pueden asignarse los contadores del programa virtuales a procesos o líneas diferentes. Los contadores de semáforos o contadores de barrido son utilizados para indicar la realización de actividades en las líneas paralelas.

La memoria *Caché* y la Protección.

Además de mantener la coherencia de los datos en la jerarquía de memoria, el multiprocesador debe asumir la consistencia de los datos entre los escondites privados y la memoria compartida. El problema de coherencia de la *multicaché* exige una invalidación después de cada operación de escritura. Estas coherencias de operaciones de control requieren un *bus* especial o un protocolo de red. Se dice que un sistema de memoria es coherente si el valor regresado de una instrucción leída es siempre el valor escrito por la más reciente escritura de la instrucción sobre la misma localización de memoria.

El orden de acceso a las *Cachés* y a la de memoria principal es una diferencia grande en los resultados de procesamiento. En la memoria compartida de un multiprocesador pueden utilizarse varios modelos de consistencia. La demanda de consistencia secuencial en todos los accesos a memoria es de orden global básica.

Un procesador no puede permitir un acceso mientras haya un acceso a la memoria compartida para una ejecución global. Un modelo de consistencia débil da cabida a que la memoria en ciertos momentos sea interrumpida. La consistencia en la sincronización del procesador puede ser restringida, con una espera para que la ejecución en la memoria mejore.

3.4.2. Modelo de intercambio de mensajes.

Dos procesos M y A que residen en dos nodos de procesadores diferentes pueden comunicarse entre sí, con el intercambio de mensajes a través de una red directa. Los mensajes pueden ser instrucciones, datos, sincronización, o las señales de interrupción etc. Si el retardo de comunicación causado por el paso de mensajes es demasiado grande, será causado por acceder variables compartidas en una memoria común.

Las multicomputadoras son consideradas multiprocesadores ligeramente acoplados, y el intercambio de mensajes se puede utilizar para la comunicación entre ambos procesadores o multiprocesadores.

Intercambio de mensajes síncrono.

El intercambio de mensajes no requiere de memoria compartida, no necesita mutua exclusión. La sincronización del intercambio de mensajes se efectúa al sincronizarse el envío y recepción de procesos en un tiempo y espacio determinado, parecido a una llamada de teléfono que utiliza un circuito concentrador de líneas.

En general no se usa *buffer* en la comunicación de los canales, la sincronización de la comunicación puede ser bloqueada al inicio por que los canales están ocupados, entonces los mensajes serían transmitidos uno a la vez en un canal en un tiempo.

Intercambio de mensajes asíncrono.

La comunicación asíncrona no requiere que el envío y recepción de mensajes deba de ser sincronizado en un tiempo y espacio. Los *buffers* ofrecen el uso de canales, los cuales son demasiado grandes para ser utilizados en el trafico de redes y para que estos no se saturen. De cualquier manera los retardos en la comunicación pueden experimentarse, porque al enviarse el mensaje y no encontrar la ruta, el mensaje debe de ser recibido. Entonces el utilizar y contar con el servicio postal utilizando cajas de correo (canal de *buffers*) asíncronos entre envíos y recepción de mensajes puede ser una alternativa.

El no bloqueo puede llevarse a cabo por el paso de mensajes asíncrono en el cual dos procesos no tienen que ser sincronizados en un tiempo y espacio. La comunicación asíncrona requiere el uso de *buffers* para tomar el mensaje a lo largo del camino y de la conexión de los canales. Los canales del *buffer* son finitos y el envío eventualmente puede ser bloqueado. En una multicomputadora síncrona los *buffer* no son necesarios porque solamente un mensaje es permitido pasar a través del canal en un tiempo.

Distribución del procesamiento

El programa duplicado y la distribución de datos son utilizados en el multiprocesamiento. Los procesos en una multicomputadora (máquina *NORMA*) son ligeramente acoplados y ellos no utilizan la memoria compartida sino el paso de mensajes, además, son conocidas por el manejo de subprogramas al nivel de la instrucción y conocidas por los procesos de grano fino.

3.4.3. Modelo de datos paralelos.

Los programas con datos paralelos requieren del uso de un conjunto de datos predistribuidos, estos se eligen de las estructuras de datos paralelos realizados en la programación. La estructura de datos interconectados es necesaria para facilitar las operaciones en el intercambio de datos.

La programación de datos paralelos hace énfasis en el procesamiento local, en las rutas de las operaciones de los datos con la permutación, duplicación, reducción, prefijos paralelos. Estos modelos se aplican a los procesos de grano fino utilizados en redes, mimeógrafos y en señales e imágenes multidimensionales.

El paralelismo en los datos puede ser implementado en computadoras *SIMD* o en multicomputadoras *SPMD*, dependiendo del nivel de grano y del modo de operación adoptado. El paralelismo en los datos ofrece un alto grado de paralelismo que se desarrollan en miles de operaciones concurrentemente. Esto implica, además, un alto grado de paralelismo en las instrucciones utilizadas.

Operaciones con datos paralelos síncronos.

Las operaciones son realizadas en el momento de la compilación mejorando el tiempo de corrida. La sincronización con el *hardware* es forzada por la unidad de control a realizar la ejecución de cerradura más rápidamente en los programas de las máquinas *SIMD*.

Paralelismo de los datos.

Con la introducción de los arreglos de procesadores *SIMD*, la instrucción escalar es directamente ejecutada por la unidad de control. El vector de las instrucciones es transmitido por todos los elementos de procesamiento. Los operandos del vector son cargados dentro de los elementos de procesamiento (EP) desde memorias locales simultáneamente, utilizando direcciones globales con diferentes compensaciones en el

registro de índice local. El almacenamiento en el vector puede ser ejecutado de una manera similar en datos constantes para todos los EP simultáneamente.

3.4.4. Modelo orientado a objetos.

En este modelo, los objetos son creados y manipulados dinámicamente. El proceso de ejecución para enviar y recibir mensajes es por medio de objetos. Los modelos de programación concurrente son construidos con objetos de bajo nivel. Los procesos, las colas y semáforos son construidos con objetos de alto nivel como los amonestadores y módulos del programa.

Programación orientada a objetos concurrente tiene tres características esenciales:

Primera, el incremento en la iteración de los procesos por usuarios individuales, utilizando diferentes versiones de *Xwindows (Linux)*.

Segunda, las estaciones de trabajo y redes han vuelto a ser un mecanismo para la solución de problemas con recursos compartidos y distribuidos.

Tercera, la tecnología del multiprocesador ha avanzado para proporcionar poder de superprocesamiento.

Los objetos son entidades del programa los cuales son datos y operaciones encapsuladas dentro de sencillas unidades de procesamiento. Esto gira entorno de la concurrencia y es una consecuencia natural del concepto de objeto. El desarrollo de la concurrencia en la programación orientada a objetos, provee un modelo alternativo para el procesamiento concurrente sobre multiprocesadores o sobre multicomputadoras.

Modelo actor.

La programación con concurrencia orientada a objetos (*COOP*) cuenta con modelos de reutilización y clasificación, por ejemplo, a través del uso de sucesiones, lo cual permite a todas las instancias de una clase particular, compartir una misma propiedad.

Los modelos actores están contenidos en sí mismos, interactúan con los componentes independientes de un sistema de procesamiento que se comunica con el intercambio de mensajes en forma asíncrona. El modelo actor y el intercambio de mensajes se ligan con semántica.

El actor básico incluye lo siguiente:

1.-*Create*: crea un actor con la descripción y conjunto de parámetros.

2.-*Send-to*: envía un mensaje a otro actor.

3.-*Become*: un actor es remplazado por su conducta, por otra de mejor conducta.

Los estados de cambios son especificados por el comportamiento de reemplazo. El mecanismo de reemplazo permite agregar un cambio para evitar el control de flujos dependientes. Cada mensaje puede crear un objeto (actor) para modificar su estado, creando nuevos objetos, y enviar nuevos mensajes.

La estructura de control de la concurrencia representa un modelo particular del intercambio de mensajes, el actor primario provee la descripción en el nivel bajo del sistema concurrente. La construcción del nivel alto es necesaria para la descripción de la granularidad y el encapsulamiento. El modelo del actor es particularmente conveniente para implementaciones de multicomputadoras.

El paralelismo en el *COOP* tiene tres modelos de paralelismo que lo describen, por ejemplo:

Primero, la concurrencia *pipeline* (tubería) involucra la coincidencia de numerar ocasionando soluciones sucesivas y pruebas concurrentes de soluciones.

Segundo, la concurrencia se basa en el término de divide y vencerás, el cual involucra la elaboración de diferentes subprogramas y la combinación de varias soluciones para obtener la mejor solución para el problema.

Tercero, se basa en la solución de problemas cooperativamente, un ejemplo es el modelo de evaluación dinámica (procesamiento de objetos) y cuerpos físicos (objetos) con la mutua influencia de campos gravitacionales, en este caso todos los objetos pueden

interactuar con otros, los resultados inmediatos son almacenados en objetos y compartidos entre ellos con el intercambio de mensajes.

3.4.5. Modelos de programación lógica y funcional.

Modelo de programación funcional.

El lenguaje de programación funcional hace énfasis en la funcionabilidad de un programa y no produce efectos después de la ejecución. No existe ningún concepto de almacenamiento, asignación, y bifurcación en el programa funcional. En otros términos, la historia de cualquier procesamiento realizado antes de la evaluación se considera irrelevante para el manejo de los programas funcionales. Las restricciones de anterioridad sólo ocurren como resultado de la aplicación en la función. El resultado de la función produce el mismo valor a pesar del orden en el cual los argumentos son evaluados. Esto implica que todos los argumentos sean creados en una estructura dinámica del programa funcional y pueda ser evaluado en paralelo.

Todas las asignaciones son sencillas y el lenguaje de flujo de datos son funcionales y naturales. Esto implica que los modelos de programación funcional puedan ser fácilmente utilizados para el manejo de datos en multiprocesadores. El modelo funcional es fácilmente aplicable en la máquina paralela *MIMD* de grano fino y es referentemente transparente al usuario. La mayoría de las computadoras paralelas están diseñadas para soportar el modelo funcional.

Modelo de programación lógica.

El modelo de programación lógica esta basado en la lógica de predicados. La programación lógica es utilizada para el procesamiento del conocimiento en grandes bases de datos. Este modelo adopta la estrategia de búsqueda implícita y el paralelismo soportado se basa en la inferencia lógica de procesos. Una pregunta y una respuesta y los datos son ejecutados en una base de datos. Las cláusulas (oraciones) en la programación lógica

pueden ser transformadas en un diagrama de flujo de datos. Por ejemplo, si un par de datos con sus predicados y sus argumentos asociados son los mismos, el conjunto de procesos y su unificación pueden ser paralelizados bajo ciertas condiciones. La unificación paralela puede ser implementada y probada sobre una misma computadora.

Algunos de los lenguajes de la programación lógica son *Prolog* concurrente desarrollado por Shapiro (1986) y *Prolog* introducido por Clark en (1987) ambos lenguajes son implementados con lenguajes relaciónales.

LENGUAJES PARALELOS.

El ambiente de programación, es una colección de herramientas de *software* y es un soporte para el *software* del sistema. El *software* para trabajar con computadoras paralelas no es muy conocido, ya que se encuentra en la fase de desarrollo. Todavía se obliga a los usuarios a emplear mucho tiempo de programación, el *hardware* detalla el lugar donde se concentra el paralelismo en el programa, utilizado como abstracción de alto nivel.

Para romper la barrera del *hardware / software*, se necesita un ambiente de *software* paralelo que contenga mejores herramientas para los usuarios, y sea factible poder llevar a cabo el paralelismo y la depuración de programas.

Características del lenguaje paralelo.

El lenguaje paralelo se puede clasificar en seis categorías: optimización, disponibilidad, comunicación y sincronización, control, datos paralelos y manejadores de procesos; dependiendo de su funcionabilidad.

A - Características de optimización.

Estas características son utilizadas para la reestructuración del programa y las directrices de compilación para convertir el código secuencial del programa en forma paralela. El propósito es emparejar el paralelismo del *software* con el paralelismo del *hardware* en la tarjeta madre de la máquina.

Las características de optimización son:

1. **El paralelismo será automático, sí el lenguaje es escrito con C y con el compilador de *Fortran*.**
2. **El paralelismo será semiautomático, sí el compilador necesita directrices o interacción con los programadores.**
3. **Soporte interactivo reestructurado, debe de contener un analizador estático, el tiempo de corrida es estático, el diagrama de flujo de datos y el código traductor para reestructuración de código será con *Fortran*.**

B - Características de disponibilidad.

Estas características facilitan al usuario la amigabilidad, el lenguaje debe tener portabilidad para una extensa clase de computadoras paralelas y se extienda la aplicación a bibliotecas del *software*.

Las características de disponibilidad son:

1. **Escalabilidad, el lenguaje es escalable para el número de procesos disponibles e independientes de la topología del *hardware*.**
2. **Compatibilidad, el lenguaje es compatible con lo establecido en lenguaje secuencial.**
3. **Portabilidad, el lenguaje es portable, para multiprocesadores con memoria compartida, para multicomputadoras con intercambio de mensajes, etc.**

C -Características de comunicación / sincronización.

Las características de comunicación y sincronización son:

1. **Asignación única de lenguajes.**
2. **Variables compartidas para *IPC*.**

3. Memoria compartida lógicamente.
4. Envío y recepción del intercambio de mensajes.
5. Procedimiento de llamado remoto.
6. Lenguaje de flujo de datos.
7. Semáforos, cajas de correo, comprobadores, barreras, etc.

D -Características que involucran la construcción del control para especificar el paralelismo.

Las características de control son:

1. Tipo de granularidad, grueso, mediano y fino.
2. Paralelismo implícito y explícito.
3. Paralelismo global en las entidades del programa.
4. Paralelismo de lazo en las iteraciones.
5. Paralelismo en las tareas.
6. Cola de tareas compartida.
7. Escritura de datos compartidos.
8. Dependencia en la especificación de tareas.

E -Características de los datos paralelos.

Las características de paralelismo en los datos se describe en la forma de cómo los datos son accesados y distribuidos en las máquinas computadoras. Las características de los datos paralelos son:

1. Descomposición automática del tiempo de corrida. Los datos son automáticamente distribuidos con la intervención del usuario.
2. Trazado de la especificación. Provee al usuario la especificación de modelos de comunicación, cómo los datos y procesos serán trazados sobre el *hardware*.

3. Soporta procesadores virtualmente. El mapa del compilador y los procesos virtuales son dinámicos o estáticos sobre los procesadores físicamente.
4. Accesos directos en datos compartidos. Los datos compartidos pueden ser directamente accesados con el comprobador de control.
5. Soporta un simple programa, múltiples datos (*SPMD*).

F - Características de los manejadores de procesos.

Estas características son necesarias para soportar la eficiente creación de procesos paralelos, implementación de multitareas, partición del programa, duplicación y balanceo de cargas dinámicas en el tiempo de corrida. Las características de los manejadores de procesos son:

1. Creación de procesos dinámicos en los tiempos de corrida
2. Procesos de carga ligera.
3. Duplicación de trabajos en el mismo programa sobre varios nodos con diferentes datos.
4. Partición de redes, cada nodo del procesador con más de un proceso y todos los nodos del procesador corren con diferentes procesos.
5. Carga de balanceo automática. La carga de trabajo es dinámicamente transferida a través de los nodos para la llegada de un trabajo a varios nodos procesadores.

De acuerdo con las características anteriormente mencionadas, la segmentación encauzada y el procesamiento vectorial ofrecen un modelo económico de realizar el paralelismo en los computadores digitales. El concepto de procesamiento encauzado es conocido como la división de tareas grandes a la entrada en una secuencia de subtareas, donde cada una puede ser ejecutada por una etapa de *hardware* especializado que actúe en concurrencia con otras etapas del encauzamiento. Las tareas sucesivas circulan dentro del cause y van ejecutándose de modo solapado al nivel de subtarea.

El procesamiento vectorial esta basado en un operando vectorial que contiene un conjunto ordenado de "n" elementos, donde n es la llamada longitud del vector. Cada

elemento de cada vector es una magnitud escalar, que puede ser un número de punto flotante, un entero, un carácter (*byte*) o un valor lógico.

COMPILADORES.

Un compilador es el que se encarga de traducir los módulos de un programa en el lenguaje de programación a módulos de código máquina o identificadores únicos. El montador de enlaces (*linker*) combina entonces estos módulos de identificadores únicos y la composición obtenida la traduce un cargador (*loader*) a posiciones de memoria principal. El conjunto de identificadores únicos define al espacio de nombres o espacio virtual.

El compilador con vectorización.

Una de las cualidades con las que debe contar un compilador con vectorización, es que debe de ser inteligente y que detecte la concurrencia entre las instrucciones vectoriales, que pueda realizarse con un encauzamiento o el encadenamiento de varios causes. Un compilador con vectorización deberá de regenerar el paralelismo perdido por el uso de lenguajes secuenciales. Es conveniente utilizar lenguajes de programación de alto nivel que dispongan de varias y potentes construcciones paralelas sobre procesadores vectoriales. En el desarrollo del paralelismo de la programación avanzada se conocen cuatro etapas siguientes:

a.- Algoritmo paralelo (A).
b.- Lenguaje de alto nivel (L).
c.- Código objeto eficiente (O).
d.- Código máquina destino (M).

El parámetro entre paréntesis indica el grado de paralelismo explorable en cada etapa. El grado de paralelismo se refiere al número de operaciones independientes que pueden ejecutarse simultáneamente. Si queremos hallar un algoritmo adecuado con un alto paralelismo (A) para resolver problemas matriciales de gran escala, necesitamos

desarrollar lenguajes paralelos (L) para expresar el paralelismo. En la actualidad, no existen lenguajes paralelos estándar. La mayoría de los usuarios siguen escribiendo el código fuente en lenguajes secuenciales [113].

En los lenguajes secuenciales tales como *Fortran*, *Pascal*, *Algol* aún se tiene L =1. el Paralelismo natural de una máquina lo determina el *hardware* por ejemplo; una super computadora que tiene O = M = 64 o 32 procesadores. En la situación ideal con un lenguaje paralelo bien desarrollado, se puede esperar que A ≥ L ≥ O ≥ M. Actualmente, el paralelismo que contiene un algoritmo se pierde al expresarlo en un lenguaje de alto nivel secuencial. Para promover el procesamiento paralelo en el *hardware* de una máquina, se necesitará un compilador inteligente que regenere el paralelismo mediante vectorización.

El proceso de sustituir un bloque de código secuencial por instrucciones vectoriales se le denomina vectorización. Al *software* del sistema que efectúa esta regeneración del paralelismo se le denomina compilador con vectorización.

Entre los compiladores más comerciales para vectorizar podemos encontrar los siguientes:

Fortran, Pascal, Algol, C, C++, visual C, etc.

3.5. ALTERNATIVAS Y ELECCIÓN DEL PROCESAMIENTO EN PARALELO.

ALTERNATIVAS.

Con la obtención de mejores computadoras y programas se han desarrollado varias herramientas para el procesamiento distribuido y una gran variedad de productos nuevos para el procesamiento en paralelo, por ejemplo:

El sistema p4.

El sistema p4 [B1], es una librería de macros y subrutinas desarrolladas por el laboratorio nacional de *Argonne*, para la programación en varias máquinas paralelas. El sistema p4 soporta el modelo de memoria compartida (en base de monitores) y el modelo de memoria distribuido (con el intercambio de mensajes). Para el modelo de memoria compartida el procesamiento paralelo de p4 está provisto de un conjunto de monitores.
Para el modelo de memoria distribuida, p4 está provisto de operaciones para envío, recepción y creación de procesos para el archivo de texto, el cual escribe a un grupo y una estructura de procesos.
El manejador de procesos del sistema p4 se basa sobre un archivo de configuración que especifica el tipo de *host*, el archivo objeto puede ser ejecutado en cada máquina, el número de procesos puede ser iniciado en cada *host* (en un sistema multiprocesador). El mecanismo manejador de procesos de p4, realiza primero un movimiento a los procesos "maestros" y a los procesos "esclavos". Posteriormente el sistema p4 utiliza el intercambio de mensajes para enviar y recibir procesos, este intercambio puede ser entre computadoras heterogéneas.

El sistema *Express*.

El sistema *Express* [B1], es una colección de herramientas con direcciones individuales y contiene varios aspectos del procesamiento concurrente. El *kit* de herramientas está desarrollado y comercializado por una corporación llamada *ParaSoft*.

El inicio de *express*, se basa en una versión secuencial de aplicaciones, posteriormente se desarrolló para una version paralela, fue modificado para cálculos de optimización. El programa es gráfico y los algoritmos están desarrollados en forma secuencial y con un manejador de procesos dinámico.

El núcleo del sistema *Express* es un conjunto de librerías para comunicación, de entrada / salida, graficas y paralelas, también incluye una gran variedad de operaciones globales y datos distribuidos.

El Sistema *MPI*.

El sistema MPI [B1], se basa fundamentalmente en una interfaz con intercambio de mensajes, el cual fue terminado en el año de 1994. En conjunto su núcleo se forma de rutinas con librerías, utiliza el intercambio de mensajes para comunicarse y ser utilizado con procesadores masivamente paralelos (*MPP*), la principal ventaja que se establece con el intercambio de mensajes estándar es que es muy portable.

Una de las principales metas para el desarrollo del *MPI*, es definitivamente para proveer a vendedores de los *MPP*. Así mismo es utilizado en configuraciones de máquinas virtuales, para habilitar las tareas y procesos, también permite resultados satisfactorios en el soporte de entrada y salida. Este sistema facilita a plataformas de hardware para que tengan una buena transferencia de procesos y en las operaciones de datos para ser implementadas con el *hardware*.

El sistema *Linda*.

El sistema *Linda* [B1], se basa en el modelo de programación concurrente y fue desarrollado en la Universidad de *YALE*. El primer desarrollo de *Linda* fue para comunicar procesos cooperativos, posteriormente se utilizó para dos métodos tradicionales de procesamiento paralelo, uno de ellos basado en memoria compartida y el otro basado en intercambio de mensajes. Estas aplicaciones son para usarse con el modelo de *Linda* en programas secuenciales cooperativos.

El sistema *Linda* generalmente se refiere a implementaciones específicas de *software* para soportar determinado modelo de programación, el *software* del sistema es también provisto de un conjunto de librerías y intérpretes apropiados para ejecutarse, dependiendo del medio ambiente se quiera manejar (multiprocesos con memoria compartida, computadoras paralelas que utilizan intercambio de mensajes, redes de estaciones de trabajo, etc.).

El sistema PVM.

El sistema PVM [B1], será descrito en detalle en el siguiente capítulo.

ELECCIÓN DEL PROCESAMIENTO EN PARALELO.

La programación paralela y la construcción de máquinas paralelas son temas relativamente nuevos, pero parece que cada día que pasa se anuncian nuevas máquinas y programas con gran velocidad de cambio, cualquier estudio de máquinas y programas existentes podría parecer obsoleto antes de llegar a publicarse. Pero lo que más importa en la mayoría de los casos, es que en los diferentes modelos de cada uno se subrayan los distintos aspectos de la computación paralela. En algunas aplicaciones, el potencial de ejecución en paralelo puede ser más evidente de un modelo a otro.

Por ejemplo, el procesamiento paralelo es aceptado para facilitar y trabajar con dos grandes desarrollos: los procesadores masivamente paralelos (*MPPs*) y con el uso general del procesamiento distribuido.

Los *MPPs* combinan cientos o miles de *CPU* en un pequeño gabinete conectado con cientos de *gigabytes* de memoria. Los *MPPs* ofrecen enorme poder de procesamiento y son utilizados para resolver grandes problemas en conjunto con el procesamiento en paralelo.

En el procesamiento distribuido, procesos cualesquiera están contenidos en un conjunto de computadoras conectadas por una red y estas computadoras son usadas colectivamente para obtener soluciones de problemas extensos. Con el avance de esta técnica, algunas organizaciones cuentan con redes de área local (*LAN*), ya que utilizadas con estaciones de trabajo facilitan y mejoran los recursos de procesamiento.

Comúnmente entre el proceso distribuido y los *MPP*, es utilizado el intercambio de mensajes para el procesamiento paralelo, ya que los datos son intercambiados entre tareas cooperativas. El modelo de intercambio de mensajes en este trabajo será la alternativa que se elija por sus perspectivas y la variedad de multiprocesamiento que soporta y desde el punto de vista de las aplicaciones, lenguajes y sistemas de *software* que utiliza. En particular se utilizará el sistema PVM.

CAPITULO 4

APLICACIÓN DEL PROCESAMIENTO EN PARALELO.

4.1 DESCRIPCIÓN DEL SISTEMA *PVM* [B1].

La máquina virtual paralela (*PVM*) es un sistema de *software* que habilita un conjunto de computadoras heterogéneas (compuestas de distintas características) para ser utilizadas de forma coherente, fácil y utilizando recursos de procesamiento concurrente.

El sistema *PVM* esta integrado por un conjunto de herramientas y librerías para ser utilizadas de forma general. Los principios sobre los que se basa son los siguientes:

Configuración del *host pool* (grupo de terminales) por el usuario.

Al iniciarse la aplicación del procesamiento de tareas sobre un conjunto de máquinas, el usuario determina en cuál se va ejecutar el programa *PVM*. El *CPU* de las máquinas y el *hardware* de multiprocesamiento (incluyendo memoria compartida y memoria distribuida de la computadora) pueden ser parte del *host pool*. Durante la operación del *host pool*, se pueden agregar y quitar máquinas.

Acceso transparente al *hardware*.

Los programas de aplicación pueden ser transparentes al ambiente del *hardware*, como una colección de atributos de elementos de procesamiento virtual o pueden ser elegidos para explotar la capacidad de especificaciones en el *host pool* o para fijar una posición en el procesamiento de las tareas sobre las computadoras más apropiadas.

Procesamiento de los procesos base.

La unidad de paralelismo de *PVM* es una tarea secuencial independiente, controlada por varias alternativas de comunicación y procesamiento. En un caso particular las tareas múltiples pueden ser ejecutadas sobre un solo procesador.

Modelo explícito del intercambio de mensajes.

El conjunto de procesamiento de tareas puede ser dividido y ejecutado en partes, utilizando datos, funciones o descomposición híbrida cooperativa (conjunto de elementos distintos) para el envío y recepción de mensajes. El tamaño del mensaje es limitado únicamente por la cantidad de memoria disponible.

Soporte heterogéneo.

El sistema *PVM* soporta máquinas, redes y aplicaciones, todas pueden ser heterogéneas, cuando se considera el intercambio de mensajes, *PVM* permite al mensaje contener más de un tipo de datos para ser intercambiado entre máquinas que tienen diferente presentación en los datos.

Soporte multiprocesos.

***PVM* utiliza el intercambio de mensajes, fundamentalmente para facilitar los multiprocesos sobre el *hardware*.**

El sistema *PVM* esta compuesto por dos partes:

La primera parte es el *daemon*, llamado *pvmd3* y en algunas abreviaturas anteriores llamado *pvmd*, éste reside en todo desarrollo de la máquina virtual paralela. Un ejemplo del programa *daemon* es el programa de correo, que corre como administrador, dirigiendo

todas las entradas y solicitudes de correo electrónico en la computadora. El *pvmd3* está diseñado para usuarios que tengan una cuenta de *login* validado y puedan instalar el *daemon* en la máquina. Cuando un usuario corre las aplicaciones de *PVM*, primero crea la máquina virtual para iniciar *PVM*, la aplicación puede iniciarse bajo el *prompt* (cursor) de *UNIX*, bajo el ambiente del sistema operativo *LINUX*, o en el ambiente del sistema operativo *WINDOWS* sobre cualquier *host* (máquina o terminal) y así múltiples usuarios pueden configurar sus máquinas virtuales y ejecutar simultáneamente las aplicaciones.

La segunda parte del sistema, es una librería con rutinas de interfase de *PVM*, contiene un completo repertorio funcional natural, el cual es necesario para la cooperación entre tareas de una aplicación. Esta librería contiene rutinas de llamada a usuarios utilizando intercambio de mensajes, producción de procesos, coordinación de tareas y modificación de la máquina virtual.

Los procesos frecuentemente realizan llamadas al paralelismo funcional, el método más común es llamado paralelismo de datos. En este método todas las tareas son una misma, pero cada una sólo conoce y resuelve una pequeña parte de los datos. Este caso puede ser referido a un modelo de procesamiento de la máquina *SPMD* (simple programa, múltiples datos). *PVM* soporta mezclas de estos modelos dependiendo de las funciones y tareas que deba de ejecutar en paralelo, de igual manera realiza la sincronización y/o intercambio de los datos.

PVM se ejecuta en los sistemas operativos *UNIX, Linux o Windows, NT* y soporta los lenguajes C, C++, y *Fortran*. Los lenguajes C y C++, están diseñados para ser la interfase de usuario con la librería de *PVM* y son implementadas como funciones, los argumentos de las funciones toman una combinación de valores con los parámetros y apuntadores apropiadamente.

Los programas de aplicaciones escritos en C y en C++ accesan las funciones de librería de *PVM*, para ser ligados por el archivo de librería (*libpvm3.a*) que es parte de la distribución estándar.

La diferencia en las implementaciones de las funciones de *Fortran* y C es un prefijo, por ejemplo, en lenguaje C la función de agregar una o más terminales es definida por la función *pvm_addhosts()* y en el lenguaje *Fortran* es definida como *pvmfaddhost()*.

En *PVM* todas las tareas son reconocidas por un entero identificador de tarea (*TID*). El servidor será conocido como el maestro, las terminales o nodos como los esclavos y, además, una máquina también se definirá como un *host*.

PROGRAMACIÓN DE LA *PVM*.

En principio la codificación de los sistemas o programas que se vayan a utilizar con *PVM* es algo muy personal del programador, ya que la estructura básica de un programa puede ser organizada en un conjunto de bloques estructurales que realizarán determinadas funciones básicas, por ejemplo:

Esquema de aplicación de la *PVM* para el maestro.

Con base en un programa común de *PVM*, el maestro se encarga tanto del arranque como del cierre para que la aplicación se ejecute ordenada y correctamente. El maestro se encarga de los datos de entrada y guarda los resultados finales de la aplicación en un archivo, para poder ser utilizados en otras terminales (nodos) o ser leídos por una estructura determinada. Se debe mencionar que si una aplicación sale de la *PVM*, se seguirá ejecutando, sólo que perderá su vínculo con la *PVM*, tanto el servidor como los clientes.

El esquema de inicio del maestro puede ser:

Solicita su *TID*.
Se inscribe en un grupo.
Engendra sus esclavos a los nodos que él considere pertinentes.
Hace un bloqueo de barrera esperando a que todos los esclavos se den de alta en un grupo.
Después de esto puede iniciarse el bucle principal del maestro.

Esquema de la aplicación de la *PVM* para los esclavos.

El inicio de los esclavos es más sencillo, las funciones básicas que un esclavo va ejecutar pueden ser las siguientes.

Solicita su *TID*.
Se da de alta en el grupo de su Maestro, y realiza un bloqueo de barrera, esperando que todos los esclavos se den de alta. Una observación importante, es que la modificación que se realiza al código secuencial es mínima.

Solicitud del *TID*.

Antes de ejecutar cualquier instrucción relacionada con la *PVM*, necesitamos solicitar un *TID*. La operación de solicitud del *TID* da de alta automática el proceso para el *daemon* de *PVM* local. En la *PVM* para *Linux* no es obligatorio solicitar el *TID*, ya que toda la llamada a la *PVM*, si el proceso no está dado de alta, lo da automáticamente; sin embargo es recomendable, que en determinadas implementaciones de la *PVM*, se solicite el *TID* antes de comenzar a trabajar.

La sentencia es: *int tid = pvm_mtyd ()*

Inscribirse en un grupo de tareas.

El inscribirse en un grupo de tareas es importante, si se emplea la sincronización, para asegurarnos que ponemos a todos en marcha y en caso contrario pueden producirse efectos de carrera si todos los esclavos entran en acción antes que el servidor continúe con la ejecución.

Para inscribirnos en un grupo de tareas, basta con ejecutar desde la tarea que desea inscribirse la instrucción *pvm_joingroup*. Esta instrucción tiene la sintaxis:

int inum = pvm_joingroup (char grupo)

inum será el identificador de la tarea dentro de un grupo. Siempre podremos preguntar por el *TID* para un identificador de grupo, o podemos preguntar por el identificador para un *TID* y un grupo. A su vez, siempre podemos abandonar un grupo con:

*int info = pvm_lvgroup (char *group)*

Engendrar varias tareas esclavas.

Para engendrar varias tareas esclavas, basta con hacer un *spawn* de las tareas. Si se tiene el caso de varias tareas distintas:

int ntareas = *pvm_spawn (char* *tarea, *char* **argumentos,
*int flag, char *host, int* ntareas, *int* * *TID*);

Repitiendo la línea tantas veces como tareas distintas queramos mandar, o tareas en máquinas distintas, donde:

char *tarea, será el nombre del archivo que vamos a ejecutar.
char **argumentos, será una matriz con los argumentos.
int flag, opciones para engendrar una tarea.
*char *host*, será el nombre de la máquina en la que se va a ejecutar la tarea, si procede.
int ntareas será el número de tareas iguales que serán engendradas en el nodo indicado.
*int *TIDs* devolverá una matriz con los *TIDs* de las tareas creadas.

El valor devuelto por la función será el número de tareas engendradas con éxito.

Es importante comprobar el número de tareas engendradas y devueltas por la función de *spawn*.

Envío de datos.

El envío y recepción de datos son la parte más importante del modelo de la *PVM*. Para enviar datos, se emplean tres etapas.

Iniciación del *buffer*.
Empaquetado de datos.
Envío de datos.

Es preciso tener en cuenta que los datos son codificados con la representación de datos externos *(XDR)* por *PVM*, para convertir el dato al formato que permite la comunicación entre terminales, y por ello es preciso empaquetarlos antes de ser enviados en forma correcta.

Cuando comienza a ejecutarse un programa en *PVM*, se tiene un *buffer* activo de emisión y otro de recepción de datos que serán empleados para comunicarse con otros nodos. Si queremos crear un *buffer* adicional, tenemos que emplear la función:

int bufid = pvm_mkbuf (int decod)

Donde *bufid* será el identificador del *buffer* y *decod* la forma de codificación. Generalmente no es preciso crear *buffers* adicionales, por lo que muy raramente se utilizará esta instrucción.
Los *buffer* pueden ser eliminados con la instrucción:

int info = pvm frebuff (int bufid)

Donde *bufid* es el identificador del *buffer* que se va a eliminar.

Se debe de recordar activar el *buffer* creado como *buffer* de envío.

Otra forma para enviar datos es con la instrucción *pvm_psend ()*, que permite mandar una matriz contigua de datos del mismo tipo.

Iniciación del *buffer.*

Siempre que se vaya a empaquetar un mensaje, se tiene que ejecutar la instrucción *pvm_initsend( )*, que inicializa al *buffer* active. Para evitar que se manden datos que estén almacenados anteriormente.

La sintaxis es:
int bufid = pvm_initsend (int encod)

El identificador del *buffer* devuelto por la función será el del *buffer* inicializado activo. El parámetro que se pasa a la función es la codificación.

Empaquetado de datos.

En la *PVM* los datos no se pueden mandar tal y como están. Es preciso empaquetar los datos en un *buffer* y después mandar el contenido del *buffer*. La gran diferencia entre C y *Fortran*, es que *Fortran* tiene una instrucción para empaquetar, mientras que C tiene catorce distintas. El formato de once de ellas, es:

int info = pvm_pkXXX (YYY *que, *int* cuantos, *int* desde_donde)
Donde XXX es la forma (tipo) en que la *PVM* va a mandar (*byte, cplx, long*), YYY la forma en C relacionado. Para empaquetar una cadena no se indica ni cuantos datos ni el desplazamiento y la instrucción es la siguiente:

int info = pvm_pkstr (char *cadena)

Una forma mejor sería:

*int info = pvm_packf (const char *formato, ...)*

Se invoca de la misma forma y con parámetros que el *prinf* de C, por lo que se puede elegir esta función y hacer por el momento a un lado las otras funciones.

TIPOS DE DATOS PARA EMPAQUETADO Y DESEMPAQUETADO.

Tipo	**Tipo C, 1ER. parámetro**	**Tipo *Fortran*, 1ER. parámetro**
byte	*char* *	*BYTE 1*
Entero 1 *byte*	*short* *	*BYTE 1*
Entero 2 *bytes*	*int* *	*INTEGER 2*
Entero 4 *bytes*	*long int* *	*INTEGER 4*
Flotante 4 *bytes*	*float**	REAL 4
Flotante 8 *bytes*	*double* *	REAL 8
Complejo 8 *bytes*	*float* *	*COMPLEX 8*
Complejo 16 *bytes*	*double* *	*COMPLEX 16*
Cadena	*char* *	*STRING*

FUNCIONES PARA EMPAQUETADO Y DESEMPAQUETADO.

Tipo	Empaquetado	Desempaquetado
byte	*pvm_pkbyte*	*pvm_upkbyte*
Entero 1 *byte*	*pvm_pkshort*	*pvm_upkshort*
Entero 2 *bytes*	*pvm_pkint*	*pvm_upkint*
Entero 4 *bytes*	*pvm_pklog*	*pvm_upklong*
Flotante 8 *bytes*	*pvm_pkfloat*	*pvm_upfloat*
Complejo 8 *bytes*	*pvm_pkdouble*	*pvm_upkdouble*
Complejo 16 *bytes*	*pvm_pkcplx*	*pvm_upkdcplx*
Cadena	*pvm_pkstr*	*pvm_upkstr*

Envío de datos.

Se puede realizar con la función:

***int info = pvm_send (int TID, int* canal)**

La tarea es enviada con el *TID* indicando el canal y el contenido total de *buffer.*
Para enviar el contenido del *buffer* se emplea la siguiente función:

***int info = pvm_ mcast (int *TIDs, int* número, *int* canal)**

Donde el primer parámetro es una matriz de *TIDs* y el segundo parámetro es el número de *TIDs* contenidos en la matriz.

La tarea cliente quedará bloqueada hasta que en el envío de todo el mensaje se haya transmitido vía el protocolo de control de transmisión (*TCP*) para el *daemon*. Después la tarea seguirá ejecutándose y el servidor intentara tener contacto con el *daemon* de la tarea receptora, este es el único retardo asociado al envío para el algoritmo para la emisión de la trama (longitud del mensaje).

Recepción de datos.

Para recibir datos, se emplean dos pasos:

La recepción del mensaje y el desempaquetamiento de los datos.

Al igual que en el envío, se emplea un *buffer*. Es importante recordar que el *buffer* de envío y de recepción al iniciar una tarea en *PVM* son distintos y tienen distintas instrucciones para ser activados.

Recepción del mensaje.

Al recibir el mensaje la tarea receptora queda bloqueada, contactando con su servidor para un canal determinado. Según el servidor, si está con un mensaje listo en el canal para dicha tarea o no y según el modo de recepción, la tarea quedará bloqueada esperando o no el mensaje. De cualquier forma mientras el mensaje es transmitido entre el *daemon* receptor y la tarea receptora, la tarea receptora es bloqueada. El mensaje es transmitido entre el *daemon* receptor y la tarea receptora vía *TCP*, la espera entre el envío del mensaje y la recepción puede ser bloqueante o no, según la función de recepción:

int bufid = pvm_ recv (int TID, int canal): bloqueante, el receptor espera que la tarea *TID* le envíe un mensaje por el canal indicado, entonces transferirá el mensaje al *buffer* de recepción (*bufid*).

int bufid = pvm_nrecv (int TID, int canal): no bloqueante, si encontramos en el servidor un mensaje de la *TID* en el canal apropiado, transferimos el mensaje de forma bloqueante al *buffer* de recepción (*bufid*), sino la función devuelve el valor de *bufid* como un 0, y no quedará bloqueada.

Analizando:

Caso del esclavo:

En caso de que no haya nadie en recepción es por que no se ha mandado nada al servidor, ya que lo importante es que no se pierda un instante de tiempo, esperando, sino que siga calculando.

Caso servidor:

En caso de que no se encuentre a nadie en recepción es porque un esclavo en particular no ha tenido tiempo de generar un proceso, por ello es mejor atender al siguiente. La función de recepción es:

int bufid = pvm_trecv (int TID, int canal, *struct timeval* *tiempo): bloqueante con temporización. El receptor estará bloqueado durante el tiempo indicado.

La función *pvm_probe* es la que verifica si ha llegado un mensaje determinado, y su comportamiento es parecido a pvm_nrecev, sólo que para recibir el paquete es necesario una instrucción de recepción de paquete. Con sintaxis:

int bufid = pvm_probe (int TID, int canal)

Aun sin bajar el mensaje, podemos obtener información detallada del mensaje con la función *pvm_bufinfo*, que nos proporciona información sobre el mensaje recibido antes de descargarlo, basta con combinarla con *pvm_probe*.

En caso de que no estuviéramos de acuerdo con la semántica de recepción, se puede emplear la función de *pvm_recvf*, que nos permite redefinir el comportamiento de la recepción del mensaje.

Cualquiera de las funciones de recepción de mensajes, puede tener como parámetro un *TID*, con valor de –1, lo que significa cualquier tarea, de la misma manera un canal con valor de –1 significa cualquier canal.

El envío y recepción entre dos tareas siempre se realizara en orden. Esto significa que si una tarea manda dos mensajes a otra, los mensajes serán recibidos por la tarea destinataria en el orden en que le fueron enviados. Por otro lado si dos tareas emiten cada una un mensaje a otra tercera, no tenemos forma de saber en que orden serán recibidas por el receptor, aunque supiéramos que una tarea se envía mucho antes que la otra.

Desempaquetado de datos.

El desempaquetado de datos es exactamente igual que el empaquetado de datos, en C, con sus catorce funciones y con los parámetros iguales, solo que se cambia *pvm_pk* por *pvm_upk*. Es decir, el formato en once de ellas es:

int info = *pvm_upk*XXXX (YYY *que, *int* cuantos, *int* desde _ donde)

Donde XXX es la forma como *PVM* los va a enviar (*byte, cplx, long*) y la forma de C relacionado es YYY. Para la cadena no indicamos ni cuántos datos ni el desplazamiento, siendo la instrucción de la forma:

int info = *pvm_upkstr (char* *cadena)

Exactamente igual que con el empaquetado de datos, tenemos la función:

*int info = pvm_upackf (const char *formato,...)*

Que tiene el mismo formato que el *scanf* de C y será más cómoda, por lo que también en este caso no se necesitarán más funciones. Los datos se desempaquetan en el mismo orden que fueron empaquetados.

Sincronización de barrera.

En este mecanismo de sincronización ninguna tarea pasa hasta que un número suficiente de tareas estén esperando en la barrera. El punto de espera se define con la función *pvm_barrier*. El problema básico es que se debe de saber cuantos procesos han de esperar en la barrera antes de dejarlos pasar a todos, información que el maestro conoce, mas los esclavos no, o al menos, no basta con saber cuántos están en un grupo, ya que puede que algunos elementos todavía no hayan entrado. De ahí que formemos un grupo, y el maestro mande esta información a todo el grupo con *pvm_mcast*.

El formato de la instrucción es:

*int info = pvm_barrier (char *grupo, int cuantos)*

Donde el primer parámetro es el grupo sobre el que estamos realizando la función y el segundo cuántos tienen que esperar para pasar. En caso de que usemos el grupo sólo para sincronía, podemos indicar en el segundo parámetro un –1 lo que significa que tomará todos los miembros del grupo. En ese momento la función tomará el valor de *pvm_gsize* (regresa el número de elementos del grupo especificado y se empleará como parámetro) por lo que no será preciso que el maestro envíe la información (en teoría) ya que en la práctica puede ocurrir que distintos procesos tengan distintos valores de *pvm_gsize* porque se ejecutan en distintos momentos, por lo que –1, puede ocasionarnos

problemas si no tenemos la seguridad de que todos los miembros del grupo ya están bloqueados, para lo cual necesitamos el bloqueo de barrera. La función *pvm_gsize* esta definida como: *int size = pvm_gsize (char *group)*

Salida de la *PVM.*

La salida es bastante fácil, basta con ejecutar la función *pvm_exit( )*. Se puede desde el maestro ir eliminando los esclavos con *pvm_kill o pvm_halt*, para salir de cada uno de los esclavos.

Variables de entorno modificables por los usuarios.

Existe un conjunto de variables de entorno que se pueden modificar, las cuales son:

PVM_ROOT: Ruta del directorio de instalación de la *PVM*.
PVM_EXPORT: Nombre de las variables del sistema que serán heredadas a los hijos engendrados con *spawn*.
PVM_ DPATH: Ruta del directorio de instalación de la *PVM* en los esclavos, por defecto.
PVM_DEBUGGER: Ruta del *script* de depuración empleado por *spawn*.

Variables de entorno no modificables por el usuario.

Existen variables de entorno que no se pueden modificar, pero que es interesante conocer para poder leer sus valores. Los cuales son:

PVM_ARCH: Nombre de la arquitectura de la máquina local.
PVMSOCK: Dirección del socket del daemon de *PVM* local.
PVMEPID: PID supuesto de la tarea que se va a engendrar con spawn esperado.
PVMTMASK: Máscara de depuración de la *libpvm*.

Opciones del parámetro *flag* (bandera) para engendrar tareas.

***pvmTaskDefault*: La *PVM* decide en qué máquina va a engendrar las tareas.**
***pvmTaskHost*: El parámetro host indica en qué máquina se van a engendrar las tareas.**
***pvmTaskArch*: El parámetro host indica en qué arquitectura o conjunto de máquinas con la misma variable de *SPVM_ARCH* se van a engendrar las tareas.**
***PvmTaskDebug*: La tarea será engendrada junto con el depurador.**
***pvmTaskTrace*: Se mantendrá un histórico para las llamadas a las rutinas de la *PVM* para esa tarea.**
***pvmMppFront*: Las tareas serán engendras con un *front_ end* de *MPP*.**
***PvmHostCompl*: Las tareas serán engendradas en aquellos nodos que no cumplan las características indicadas.**

Para indicar varias opciones, lo hacemos asociándolas con el operador //.

Opciones de codificación del *buffer*.

El *buffer* admite varias formas distintas de codificación, seleccionables empleando *flags*. Estos son:

***PvmDataRaw*: Los mensajes serán mandados sin codificar en *XDR*. Si al desempaquetar no es entendido el formato generará un error. No es buena idea para nuestro problema, ya que contamos con una arquitectura heterogénea.**
***PvmDataDefault*: Los mensajes serán mandados codificados con *XDR*. Será la empleada en nuestro problema.**
***PvmDataInPlace*: En el *buffer* tendremos punteros a datos y tamaños de datos, en lugar de datos. Acelera bastante algunas operaciones, pero hemos de tener cuidado con modificar los datos.**
Para indicar varias opciones, lo hacemos asociándolas con el operador //.

Otras funciones de mantenimiento de tareas.

PVM **nos permite desde el código controlar las distintas tareas con un conjunto de funciones. Estas son:**

Eliminar la tarea con el *TID* indicado. Para salir es mejor *pvm_exit* que eliminarse a sí mismo.

int rc = pvm_kill (int tid)

Devuelve el *TID* del proceso que engendró la tarea, o *PvmNoParent* si fue engendrada directamente por el usuario.

int tid = pvm_parent ( )

Devuelve información sobre una tarea

int status = pvm_stat (int tid)

Funciones para operar sobre el conjunto de máquinas.

Se pueden operar un conjunto de máquinas desde el programa, las funciones son:

Agregar un conjunto de máquinas a la *PVM*.

pvm_addhosts (char ** hosts, int nhost, int * infos)

Eliminar un conjunto de máquinas de la *PVM*.

pvm_ delhosts (char ** hosts, int nhost, int * infos)

Devuelve *PvmOk* si la máquina indicada está en la *PVM*, *PvmNoHost* en cualquier otro caso.

int pvm_mstat (char * host)

Devuelve información acerca del estado de la máquina virtual paralela. *Nhost* será el número de máquinas, *narch*, el número de arquitecturas y la host será un puntero a una matriz con los datos de todos los nodos *(TID*, nombre, arquitectura y velocidad relativa).

int pvm_config (int * nhost, int * narch, struct pvmhostinfo **hosts)

Para la máquina virtual paralela.

int rc = pvm_halt ()

El *daemon* se ejecuta (corre) en la máquina donde se procesa la tarea (si procede).

int dtid = pvm_start_pvmd (int args, char **argv, int block)

Devuelve el identificador de la máquina en un proceso. Solamente extrae el dato del *TID* sin comprobar la existencia o si está activo.

int dtid = pvm_tidtohost (int tid)

Funciones para operaciones con *buffers*.

Se pueden realizar operaciones con los *buffers* desde el programa y es permitido emplear más de un *buffer*. Las funciones son:

Devuelve el identificador del buffer activo de recepción.

int bufid = pvm_getrbuf ( )

Devuelve el valor del *buffer* de emisión.

int bufid = pvm_getsbuf ( )

Activa un *buffer* para emisión.

int bufid = pvm_setsbuf ( )

Activa un *buffer* para recepción.

int bufid = pvm_setrbuf ( )

Otras operaciones con grupos en la *PVM*.

Además de las operaciones con grupos básicas se dispone de otras operaciones adicionales, las cuales son:

***pvm_gather*: recoge los datos de todo un grupo para almacenarlos en una matriz.**

***pvm_gettid*: devuelve el *TID* de una tarea para una cierto número de instancia.**

***pvm_reduce*: aplica un operador de reducción (sumatoria, producto, máximo, mínimo y el usuario puede definir algunos otros) a los miembros de un grupo, generando un escalar.**

***pvm_scatter*: distribuye datos de una tarea a los restantes de un grupo. Los datos están originalmente en una matriz y se mandan los *count*.**

***pvm_getinst*: devuelve los números de instancia de una tarea dentro de un grupo.**

Es recomendable bloquear con la función de *pvm_barrier* después de la inscripción de las tareas en un grupo.

4.2. MEJORAS QUE PUEDE BRINDAR *PVM* AL SISTEMA DE OPTIMIZACIÓN DE RECARGAS.

La máquina virtual paralela (*PVM*) puede ser aprovechada para mejorar el tiempo de ejecución en los cálculos y estimados del sistema de optimización de recargas (SOPRAG) desde tres puntos de vista fundamentales y basados en la organización del procesamiento de tareas.

Primero, el modelo de la *PVM* en conjunto con diferentes aplicaciones y en particular con el procesamiento concurrente y con procesos estrechamente relacionados ejecutando el mismo código, se efectuará el procesamiento de SOPRAG en diferentes máquinas, dividiendo el trabajo en cada una de ellas. Este paradigma se dividirá en dos categorías.

1. El modelo maestro-esclavo (terminal-nodo) donde el maestro será el responsable de engendrar los procesos necesarios, inicializando, recolectando, y desplegando los resultados de SOPRAG.

2. El programa esclavo ejecutará el procesamiento de las cargas de trabajo que el maestro determine (estáticamente o dinámicamente).

Segundo, el modelo de la *PVM* para operar a SOPRAG será determinado con procesamiento en forma de árbol. En este escenario los procesos serán engendrados (generalmente dinámicos, como se vayan dando los resultados del procesamiento) en un árbol de tal manera que se establecerá la relación padre-hijo).

Tercero, el modelo híbrido, se llevará acabo a través de la combinación del modelo del árbol y el modelo concurrente. Escencialmente este paradigma se desarrollará en una estructura engendrada arbitrariamente y se presentará durante algún punto en la ejecución de SOPRAG, el proceso de relación de parentesco a través de éstos frecuentemente también

corresponderá a la comunicación en la topología de la red de computadoras. La elección del modelo en SOPRAG dependerá de cómo se seleccionará la mejor estructura natural del programa para paralelizar.

El procesamiento concurrente involucrará tres etapas para la ejecución de SOPRAG.

1. Se deberá de inicializar el grupo de procesos, en este caso el nodo de procesamiento, la diseminación de la información en el grupo y los parámetros del problema, también la asignación de la carga de trabajo.
2. El procesamiento general de SOPRAG.
3. La colección de resultados y los desplegados de salida de estos, durante este punto el grupo de procesos será terminado.

El control de la estructura dependerá de la forma en la que se diseñe el procesamiento secuencial de SOPRAG al momento de ejecutarlo en la *PVM*.

4.3. SISTEMAS OPERATIVOS A UTILIZAR CON *PVM*.

Un sistema operativo a utilizar junto con SOPRAG y *PVM* debe de contar con un programa tal que al iniciar su ejecución y al encender la computadora, se establezca un estado inicial con los componentes de dicha computadora y se tenga como función primaria permitir que otros programas se ejecuten y se tenga acceso ordenado a los recursos que ofrece el equipo. El sistema operativo debe suministrar procesos que controlen los dispositivos periféricos, administrando recursos compartidos por varias tareas como la memoria y equipos periféricos y estableciendo las prioridades de las tareas que ejecuta la computadora. También se debe mantener la integridad del proceso recuperando el control cuando algún programa falle. En un sentido más extenso, el sistema operativo debe de contener otros elementos que amplíen su funcionalidad. Otro grupo de programas de sistemas multitareas y multiusuarios incluyen servicios adicionales, brindando junto con el sistema operativo: correo electrónico, paginas web, bases de datos etc.

En el sistema operativo *MS-DOS* lo importante es la integración. No se trata de simples rutinas utilitarias que se ejecuten sin importar cual se esté utilizando o esté presente. El desarrollo del sistema operativo debe asegurar que las varias piezas se comuniquen entre sí para asegurar la integridad de la operación total. Es por eso que es común que el sistema operativo ofrezca algún sistema de mensajes entre los varios componentes. Estos mensajes llevan información de una tarea a la otra y / o instrucciones de qué hacer a continuación.

Inicialmente, cuando los equipos no eran muy poderosos, una instrucción se ejecutaba después de otra en forma predecible. La lentitud de los periféricos con relación a la velocidad del procesador hizo que se implementará el mecanismo de interrupción por parte de los periféricos, permitiendo al procesador atender a otros servicios mientras el componente periférico realiza su tarea. Al concluir lo que estaba haciendo, sea imprimir una línea o traspasar a memoria un bloque de información, el periférico manda una señal al procesador para que interrumpa lo que está haciendo y le atienda.

Existen varias formas de mantener la sincronización. Una de esas formas es hacer que los módulos o partes de un programa o el sistema operativo envíe mensajes a los

módulos dependientes de los resultados producidos. Si existe un mecanismo estándar, como ocurre con *Windows*, una parte del sistema operativo puede analizar y manejar estos mensajes.

El sistema operativo, con el objetivo de hacer un uso eficiente del potencial del procesador, interrumpe al programa que el procesador está ejecutando para atender a algún periférico que haya concluido lo que estaba haciendo. Esta terminación puede ser exitosa o no. Lo importante es que los periféricos son relativamente tan lentos que hay que atenderlos rápido para ponerlos a trabajar lo más rápido posible para que no atrasen al trabajo que se quiere hacer con la computadora.

Los sistemas operativos en algunos equipos centralizan información, como los que dan el servicio de *Internet*, y en otros equipos realizan tareas muy definidas.

En el caso de un dispositivo el sistema operativo puede colocarse en *ROM* porque es simple y necesita estar en memoria al momento de encender el dispositivo.

En el caso de un sistema de un servidor, el solo proceso de arranque para llevar al sistema a un estado inicial predecible puede requerir de un número amplio de programas que no se justifica que estén todo el tiempo en memoria. Además, pueden existir servicios que se usan ocasionalmente que igualmente no justifican el estar permanentemente en memoria y que se traen desde un almacenamiento secundario, usualmente, un disco magnético.

Un sistema operativo ofrece, por lo general, un mecanismo de comunicación. El servicio de comunicación o interfaz puede ser de línea, como *MS-DOS y UNIX*, o puede ser gráfico con la adición de un apuntador como es un *mouse*, en el caso de *Windows 3/95/98/NT o XWindows (Linux)*.

Un sistema operativo por lo general, es un programa complejo. Dependiendo del equipo en el que se ejecute, éste puede estar compuesto de cientos de instrucciones o millones de instrucciones. En el desarrollo de sistemas operativos grandes como *UNIX y Windows NT* es razonable dividirlo en módulos. Estos pueden ser programados y probados por separado.

A continuación algunos términos que son relevantes para entender el funcionamiento de un sistema operativo.

El *Kernel* es el núcleo del sistema operativo, está permanentemente en memoria y una vez que el equipo inicia su función es realizar las siguientes tareas:

Traslado del control de un programa a otro.
Control y programación de dispositivos periféricos.
Manejo de interrupciones y condiciones de error.
Comunicación entre procesos.
Cronogramación de tareas.
Manejo de la memoria.
Programas, procesos e instancias.

Es importante establecer la diferencia entre un programa, un proceso y una instancia. Un programa es el conjunto de instrucciones que escribe el programador. Éstas, una vez que están en lenguaje de máquina, residen en un medio magnético o pueden fijarse en una Memoria de Solo Lectura (*ROM*).

Un proceso, o tarea, es la ejecución de este programa o una parte de él. A su vez, un proceso puede estar en ejecución simultánea en varias etapas. A esto se les llama instancias de un proceso. Cada una de estas ejecuciones es más o menos independiente de la otra.
Para saber los detalles de una instancia se deben examinar el grupo de datos que la define. Este grupo de datos son; el estado en que encuentra la instancia y la cola de mensajes a procesar. El estado puede contener la instrucción en la que se va a realizar la ejecución, en ese momento los valores asignados a variables deben de iniciar la ejecución o durante la misma.

El programador cuando escribe, decide si el programa lo hace único o lo divide en tareas y si las tareas pueden tener varias instancias de ejecución. Para esto último, se debe preparar las tareas para que sean re-entrantes. Una tarea es re-entrante cuando en cada instancia de ejecución el sistema operativo se asigna un bloque de memoria separado para

el almacenamiento de sus datos. Si alguna parte del proceso no es re-entrante entonces una parte de memoria compartida entre las varias instancias sirve de control para determinar cuando las otras instancias deben detener la ejecución hasta que la instancia controladora salga de la porción del programa que no es re-entrante. Esta memoria compartida recibe el nombre de semáforo.

La mayoría de las computadoras personales *(PCs)* instaladas contienen una sola unidad de procesamiento (*CPU*) para procesar las tareas. Es posible tener varios *CPU* en la tarjeta madre con lo que es posible tener varios procesos corriendo simultáneamente. Cuando no se tienen varios *CPU* se puede crear la ilusión de multiprocesamiento por medio de mecanismos de interrupción de las tareas mudando cierto tiempo de una tarea. Esta mudanza (*switching*) es una de las tareas del *kernel*. Aún con varios *CPU* se pueden hacer *switching* de tareas asignadas a cada *CPU*. *Windows NT* puede atender a varios *CPU*.

Existen dos formas básicas de proceder a la mudanza entre tareas. Las tareas pueden cooperar suspendiéndose voluntariamente cada cierto tiempo o en puntos determinados en su ejecución o el sistema operativo con ayuda de circuitos en el *hardware* interrumpe la tarea que se esté ejecutando. Estos dos mecanismos se les conoce en inglés como *cooperative switching* y *pre-emptive switching*. *Windows 3.1x* es un ejemplo de una mudanza cooperativa. *UNIX y NT* son ejemplos de *pre-emptive switching*.

Cuando el procesador muda una tarea a la otra lo que hace es guardar el estado en que está la tarea al suspenderse y guardar el contenido en registros o memorias del procesador y reemplazar los valores de la próxima tarea a ejecutar. Este cambio de contenido de registros se le denomina cambio de contexto. Este cambio toma un tiempo pero es muy poco en comparación al tiempo que se les da a las tareas para su ejecución.

El sistema operativo de un computador es cargado a memoria de una de dos maneras. Se le carga en la Memoria de Solo Lectura (*ROM)* de modo permanente o se ubica en un almacenamiento externo, generalmente un disco magnético, y se carga al encender el computador.

En el primer caso el sistema operativo toma el control del equipo al momento de arrancar y es muy poco el mantenimiento que tiene que hacer.

En el segundo caso, una parte muy pequeña del sistema operativo está presente en *ROM* para iniciar la carga del sistema operativo desde una unidad externa de almacenamiento. Esta unidad externa es por lo general un disco magnético pero puede ser un Disco Compacto (*CD*) o un cartucho de cinta magnética.

Generalmente esta carga se hace en varias etapas. El programa en la *ROM* revisa el estado de la Memoria de Acceso Aleatorio (*RAM*) y procede a cargar un programa cargador más avanzado que puede realizar otras revisiones y establecer un estado inicial adecuado al sistema operativo. Como etapa final de la carga del sistema operativo se inicia la interface con el operador. Esta interface puede ser de línea o gráfica.

En el sistema operativo *UNIX* es posible atender la entrada de datos y su modificación en lo que se llama modo interactivo. El modo de ver la computación fue cambiando dándole más énfasis a la posibilidad de conectar terminales que a la posibilidad de crear lectores más rápidos de tarjetas y cintas perforadas. La ingeniería de materiales tuvo mucho que ver en el abaratamiento de los circuitos electrónicos. Hoy en día los equipos son de un costo muy reducido así como el costo total del procesamiento de los datos.

Antes de elegir adecuadamente los sistemas operativos a utilizar con SOPRAG, se dividirán en dos grupos:

1. Por la forma de ejecutar las tareas.

Monotarea (*MS-DOS*)

Multitarea (el sistema operativo puede atender varias tareas)

a.- Monousuario (*OS/2*)

b.-Multiusuario (*VMS, UNIX*)

2. -Por la forma de realizar la planificación.

Cómo se reparte el tiempo de *CPU* entre los diversos procesos.

Tiempo compartido (*Round-Robbin*): Se asigna el mismo tiempo para cada uno de los procesos.

Prioridades: Cada proceso tiene asignada una prioridad; hasta que no termina un proceso, no se cede la *CPU* al siguiente.

Estáticas: Las prioridades son fijas, no se modifican.

Dinámicas: Existen ciertos criterios implementados en el S.O.

Mixtas (*VMS, UNIX*): Existe una planificación concreta a base de asignar tiempos en función de prioridades. Si dos procesos tienen asignada una prioridad, se comparte el tiempo entre los dos.

3. –Por la gestión de memoria:

a. -Memoria real: conjunto de posiciones de memoria principal asignadas al programa, sistema operativo etc. y que a su vez definirá el espacio de memoria física.

b. -Memoria virtual (puede ser mayor que la real): proporciona la ilusión de que existe una memoria muy grande. Aun cuando la memoria real (física) sea pequeña. Esta ilusión se logra permitiendo a un programador operar en el espacio de nombres mientras la arquitectura de la computadora provee el mecanismo para traducir las direcciones (virtuales) generadas por el programa (en tiempo de ejecucion) a direcciones de posiciones físicas.

Por otra parte debemos de saber que el conjunto de identificadores únicos definen el espacio virtual o espacio de nombres, y el conjunto de posiciones de memoria principal asignadas al programa definen el espacio de memoria física.

UNIX es un sistema operativo:

- Multitarea,
- Multiusuario,
- Planificación mixta,
- Casi todas las implementaciones son de memoria virtual.

Hoy por hoy es común que se tengan computadoras personales (*PC*) ejecutando un sistema operativo como *Windows 95/98*. En la inmensa mayoría de los usuarios de *PCs*, el sistema operativo es un elemento oscuro. En más de una ocasión confunden el programa de procesamiento de palabras o cualquier otro programa que se utiliza con *Windows*, es importante saber exactamente qué ofrece un sistema operativo. En ocasiones es necesario saber los mecanismos de seguridad, conectividad, multitarea, acceso y seguridad en el manejo de archivos y otra variedad de posibilidades. Así como saber qué posibilidades se brindan al manejar los equipos periféricos que se conectan al procesador, la capacidad para detectar problemas con estos equipos y la posibilidad de sustitución de los equipos periféricos por otros más eficientes o de mayor capacidad.

En materia de equipos servidores los sistemas operativos por excelencia son *UNIX* y *Windows NT*. Va a ser más frecuente que se deban interconectar ambos sistemas operativos. *Windows* 95/98 continuará siendo utilizado mientras este sistema operativo sea una alternativa más económica que *Windows NT Workstation*.

Linux, es una alternativa tipo *UNIX*, este sistema operativo se puede acomodar a los requerimientos del sistema SOPRAG. En este caso particular y con base que *PVM* puede ser utilizado con los principales sistemas operativos(*Windows* 3/95/98/ *NT*, *UNIX* y *Linux*) se echará mano de ellos, utilizándolos en los equipos que posteriormente se especificarán.

4.4. ANÁLISIS DEL *HARDWARE* Y *SOFTWARE* PARA EJECUTAR LOS PROCESOS DE *PVM*.

Análisis del *hardware* para ejecutar los procesos de *pvm*.

La plataforma o arquitectura de las máquinas a utilizar es importante para correr el sistema *pvm*, a pesar de que corre en un conjunto de computadoras heterogéneas se debe resaltar algunas características mínimas en el *Hardware* como las siguientes:

La computadora *host* (principal) debe de contar con un buen procesador:
Velocidad de procesamiento de 500-1000 Mhz.
Diseño superescalar.
RAM-instalada de 128 MB-1GB.
Tarjeta de interfaz de redes.
Puertos NIC integrados.
Disco duro de 1GB-30GB.
Adaptador SCSI.

Los nodos, esclavos (computadoras) deben de contar las siguientes características:
Computadora (PC).
Velocidad de procesamiento de 300-1000 Mhz.
Diseño superescalar.
Disco duro de 1GB.
RAM-instalada de 64-128 MB.
Bus de direcciones de 32 Bits.
Bus de datos 64 Bits.
Cache interno 8k datos y 8k insts.
Registros de 32 B.
Tarjeta interfaz de red.

La máquina virtual paralela permite al usuario iniciar la ejecución de una tarea y poder configurarla o modificarla de acuerdo a las necesidades de cada uno. La consola puede ser iniciada o interrumpida en diferentes tiempos de ejecución o en algunos de los *hosts* (nodos). El contar con una buena infraestructura de comunicación entre el servidor y los nodos tendrá como resultado un buen flujo de los datos e intercambio de mensajes.

Análisis del *software* para ejecutar los procesos de *pvm*.

La importancia de saber cómo se van a ejecutar los procesos radicará fundamentalmente en el intercambio de los mensajes y será la parte medular del diseño.

El envío de los mensajes estará constituido en tres etapas en la máquina virtual paralela, primeramente, se envía una señal para inicializar al *buffer*, segundo, el mensaje es empaquetado en el *buffer* utilizando como clave algún número o identificador de tareas, tercero, el mensaje es enviado a otro proceso y el mensaje es recibido y desempaquetado por una rutina.

Este proceso se puede diseñar para sistemas distribuidos de la siguiente forma:

1. – Dividir el problema en subpartes, tanto tareas como datos.
2. – Repartir las tareas y datos entre procesadores.
3. – Sincronizar las diferentes subpartes.
4. – Tomar en cuenta tiempos de comunicación.
5. – Controlar los accesos concurrentes.
6. – Tener cuidado con las interrupciones (*deadlocks). ¡por sus siglas en ingles!*

El control que se tenga sobre todo el *software* al implementar nuestro algoritmo va a ser de suma importancia ya que la coordinación de los procesos estará desde el inicio, al introducir nuestros datos, archivos, y programas así como a la interpretación de los resultados que se vayan generando cada instante.

4.5. ALTERNATIVAS DE LOS SISTEMAS OPERATIVOS A UTILIZAR PARA LA OPTIMIZACIÓN EN EL SISTEMA DE RECARGAS.

Con los resultados obtenidos anteriormente en el análisis del *software*, se sugerirán tres alternativas para mejorar y hacer más eficiente el tiempo de procesamiento en los cálculos realizados por el sistema SOPRAG.

Alternativa 1. Utilizando SOPRAG con *PVM* y sistemas operativo, *UNIX*, *Linux* o *Windows*.

1. Instalación de *software* en maestro-esclavos.

a-Instalar el sistema operativo *UNIX* o *Linux* en el servidor (maestro).
b-Instalar Linux en los nodos (esclavos).
El maestro y los esclavos deben de estar interconectados en una red local o red de área amplia.
c-Instalar el sistema *PVM*.
d-Compilar librerías de *PVM* para cada una de las arquitecturas del sistema.
e-Compilar el *daemon* de *PVM* para cada una de las arquitecturas del sistema.
f-Crear un archivo $*HOME /.rhosts*.
g-Crear un directorio $*HOME /Pvm3/bin/pvm/ARCH* (*ARCH* es la arquitectura).
h-Instalar el programa coordinador de grupos: .
i-Instalar *software* en $*HOME*/pvm3/lib/*ARCH*: .
j-Definir variable de ambiente (*.profile* o .cshrc).

2.Compilación.

a-Hacer un programa que incluya librerías de *PVM* y SOPRAG.
b-Compilar el programa.
c-Los programas ejecutables se deben de colocar en $*HOME/pvm3/bin/ARCH*

3.Ejecución.

a-Definir un archivo *host.list* en la máquina virtual paralela.

1. **Inicializar el *daemon*.**
2. **Correr el programa.**

4.Verificación de la ejecución.

a-El programa consola permite verificar el estado de la máquina virtual paralela.

Alternativa 2. Utilizando SOPRAG con *PVM* y sistemas operativos *Win NT* y *Win* 98.

1. Instalación de *software* en maestro- esclavos.

a-Instalar el sistema operativo *Windows NT* en el servidor (maestro).
b-Instalar Windows NT o 98 **en los nodos (esclavos).**
El maestro y los esclavos deben de estar interconectados en una red local o red de área amplia.
c-Instalar los archivos de *PVM* en el directorio: C:*program filesPVM*3.4 .**
d-Instalar las entradas del registro para *NT* o variables de ambiente para *Windows* 98.
Si la elección es *NT* instalar las entradas en el registro. Basado en la versión de visual C++ y opcionalmente *Digital Fortran*. Pulsando un doble clic en *pvm_user* y *pvm_user_env.reg* archiva el directorio apropiado. Cada usuario de *PVM* tendrá que instalar las entradas de *pvm_user_env.reg* para su cuenta personal mientras *pvm_reg* solo se hará una vez.
Si la elección fuera *Windows* 98, se instalarán los valores de ambiente en C:\ *Autoexec.bat*.

e-Instalar el *rshd*.

Si la elección es *Windows NT* se utiliza con las variables de ambiente.

Si fuera la elección *Windows* 98 no habría problema.

f-Copiar los directorios apropiados de *PVM* en:

En *NT*, C:\ *Program Files*\ *PVM 3.4* \ *WIN 32* \ *PVM NT*.

En Win 98, C:\ *Program Files* \ *PVM 3.4* *WIN32**PVM*.

g-Activar las variables de ambiente. Para *NT y Win* 98.

2.Compilación.

a-Pulsar Doble vez en el botón de *Desktop* (escritorio) para compilar *PVM* y SOPRAG en la máquina (computadora).

b-Pulsar (teclear) *make.bat* en sistema operativo *MS / DOS* para ver opciones.

c-Revisar que esté listo.

3.Ejecución.

a-Definir el camino más fácil para llegar a la solución (atajos) en la máquina virtual paralela en su menú de salida.

-Se utilizarán usuarios individuales.

-Para todos los usuarios.

4-Instalar una forma más fácil para llegar a la solución (atajos) a todos los esclavos (usuarios, nodos) de *Windows NT*.

5-Instalar la forma más fácil para llegar a la solución (atajos) a todos los esclavos (usuarios, nodos) de *Windows* 98.

Alternativa 3. Utilizando SOPRAG con *PVM* y sistemas operativos *Windows NT*, *Windows*, *Linux*, *Win* 98.

1. Se Instalará el *software* en las computadoras maestro y esclavos de acuerdo a la forma que anteriormente se especificó, los sistemas operativos se pueden instalar combinándolos de la mejor manera para ser ésta la óptima y las instrucciones de cómo se instalarán son similares a las mencionadas en las alternativas 1 y 2.

CAPITULO 5
ESPECIFICACIÓN DE LOS REQUERIMIENTOS DEL SISTEMA

5.1. CARACTERÍSTICAS DEL *HARDWARE*

Como en todo trabajo, desarrollo e investigación es necesario determinar con exactitud qué tipo de equipo se utilizará para obtener los resultados esperados, en este caso en particular se ha determinado utilizar el *hardware* siguiente:

CARACTERÍSTICAS DEL *HARDWARE* A UTILIZAR CON SOPRAG

COMPUTADORA	TIPO	PROCESADOR	MEMORIA	CONEXIÓN
1 ESTACIÓN DE TRABAJO	SERVIDOR (MAESTRO)	*ALPHA* A 500 Mhz.	RAM A 128 MB.	TARJETA DE RED *ETHERNET*
3 COMPUTADORAS PERSONALES	ESCLAVO (NODO)	*PENTIUM* III A 500 Mhz.	RAM A 64 MB.	TARJETA DE RED *ETHERNET*

El equipo o *hardware* estará conectado a través de una red local (LAN) por medio de un concentrador que permitirá el envío de la información entre las diferentes computadoras (figura 15).

5.2 CARACTERÍSTICAS DEL *SOFTWARE*

Contando con el *Hardware* adecuado, es necesario especificar el *software* a utilizar en las diferentes etapas del sistema.

CARACTERÍSTICAS DEL *SOFTWARE* A UTILIZAR CON SOPRAG

COMPUTADORA	SISTEMA OPERATIVO	COMPILADOR	PROGRAMAS DE TRABAJO	CONEXIÓN
ESTACIÓN DE TRABAJO	*UNIX*	C	SOPRAG, *PVM*, *PRESTO*	TCP/IP
COMPUTADORA PERSONAL I	*WINDOWS 98*	*VISUAL C*	SOPRAG, *PVM*, *PRESTO*	TCP/IP
COMPUTADORA PERSONAL II	*WINDOWS 98*	*VISUAL C*	SOPRAG, *PVM*, *PRESTO*	TCP/IP
COMPUTADORA PERSONAL III	*WINDOWS 98*	*VISUAL C*	*SOPRAG*, *PVM*, *PRESTO*	TCP/IP

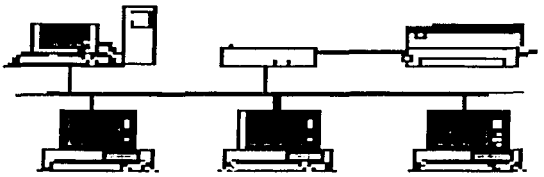
Se necesitará tener el simulador PRESTO, el SOPRAG y el PVM en cada una de las máquinas. El programa SOPRAG será modificado para incorporar las funciones de PVM y será compilado con el compilador del lenguaje C en cada una de las computadoras. La transferencia de información entre las máquinas se hará con el protocolo TCP/IP.

5.3 INTERFAZ CON EL USUARIO.

La comunicación del sistema con el usuario será por medio de archivos, tanto para la entrada de datos (*INPUT)* cómo para la obtención de resultados (*OUTPUT*). Por el momento el sistema no requiere ninguna interfaz de tipo gráfica con el usuario. Este sistema es para investigación, análisis y se necesitarán abrir archivos intermedios con información producida durante las diferentes etapas del proceso o del sistema.

Los archivos que utiliza el sistema son:

1.Archivo *ASCII* de datos de entrada, que proporcionan los datos correspondientes al funcionamiento de los algoritmos.
2.Archivo *ASCII* de datos de entrada, que proporcione los datos al simulador.
2a. Archivo binario con el banco de datos nucleares que requiere el simulador.
2b. Archivo binario con información requerida por el simulador (MASTER FILE)
3.Archivo *ASCII* de datos de salida, del simulador.
4.Archivo con el valor de la función objetivo y los valores de limite de potencia y quemado.
5.Archivos *ASCII* de salida de monitoreo de las variables de los algoritmos genéticos, simulador para intervenir en cualquier instante en SOPRAG.



Construcción del sistema (esquema físico)
Figura 15.

5.4. DESEMPEÑO DEL SISTEMA.

Se espera que con la paralelización del SOPRAG, el tiempo de ejecución de un proceso de optimización de alrededor de 100 generaciones, o de 10,000 simulaciones con PRESTO (es decir poblaciones de 100 individuos) se reduzca al menos a la mitad, comparado con el SOPRAG "secuencial " y que por lo tanto el tiempo total de ejecución sea de 5 horas aproximadamente.

5.5. FUNCIONABILIDAD DEL SISTEMA.

Por el momento no se espera que el sistema sea "amigable" desde el punto de vista de interfaz hombre-máquina. Se espera que sea eficiente y versátil, para esto se deberá de reducir el manejo de archivos innecesarios que pudieran aumentar considerablemente el tiempo transcurrido en el intercambio de datos entre las computadoras a través de la red.

La versatilidad radica en la posibilidad de compilar de nuevo el SOPRAG en cualquier plataforma que se pudiera incorporar al sistema, por lo tanto la codificación en lenguaje C debe de ser estándar.

Se espera también que el sistema pueda incorporar más nodos o computadoras para mejorar los tiempos de ejecución. Para esto la programación y el diseño deberán permitirlo con un mínimo de cambios.

CAPITULO 6
DISEÑO CONCEPTUAL DEL SISTEMA.

En este capitulo se presentan alternativas de solución a nuestro sistema, después del estudio, interpretación y análisis realizado en nuestros capítulos anteriores, se llega a plantear métodos diferentes de solución como son:

Primero, realizar una preparación para manipular el ambiente de *PVM* en cada una de las computadoras (en este caso será similar para el maestro como en los nodos). Segundo, determinar un diagrama general para definir los archivos de entrada y salida de datos para poder monitorear cada una de las etapas de SOPRAG. Tercero, definir tres diferentes soluciones particulares para ser utilizadas en nuestro sistema y explicando variantes de solución al inicio en la lectura de los datos: Cuarto, definir la mejor solución explicando cada una de las etapas de desarrollo [L2].

A. – Preparación del ambiente de PVM.

En esta etapa es necesario especificar y definir lo siguiente:

Se definirán las siguientes operaciones para realizarse en conjunto tanto en el maestro y en los nodos.

1. – Las Subrutinas externas.
2. – Las funciones Externas.
3. - Las Funciones intrínsecas (esenciales).
4. – Los Parámetros.
5. – Las variables, escalares, arreglos etc.
6. – Integración a un grupo de trabajo.
7. – Generación del identificador de tareas (TID).
8. – Verificación de todo el sistema para que no se presenten errores.
9. - El maestro engendrará el número de nodos que utilizará en el sistema (en este caso particular son 3 nodos).
10. – Realizar un bloqueo Barrera para recepción y envío de datos.

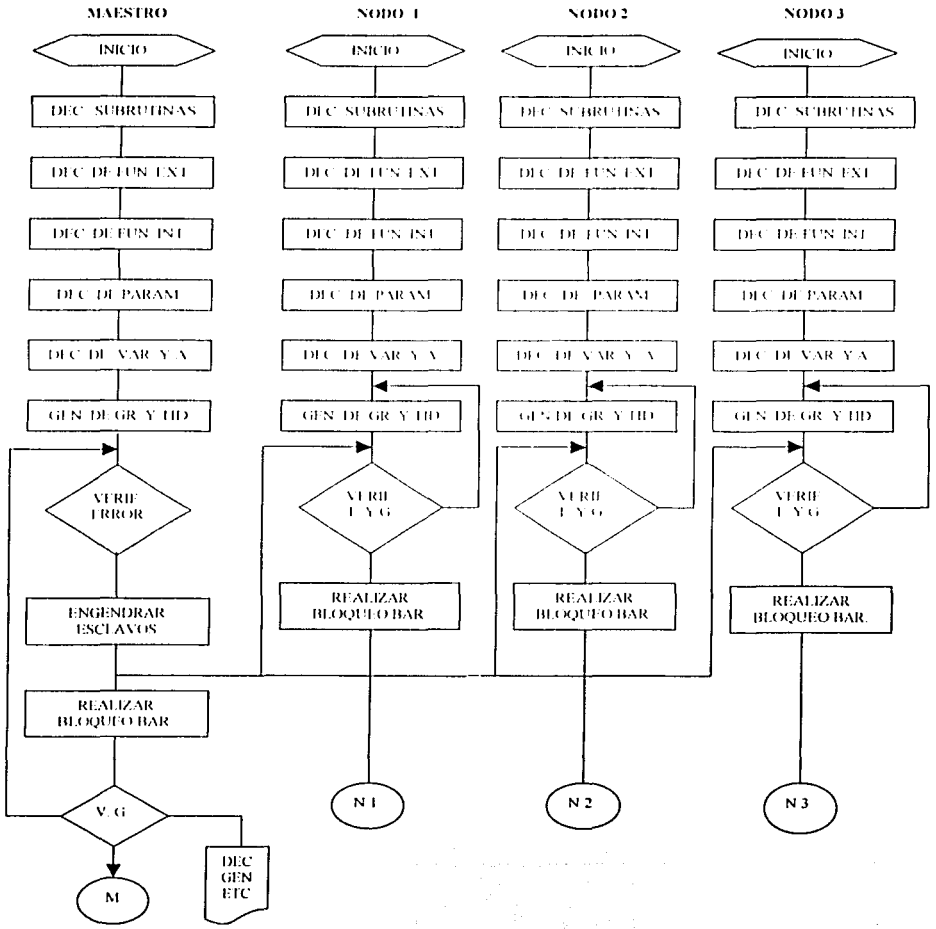
11. – Inicializar el *Buffer*.

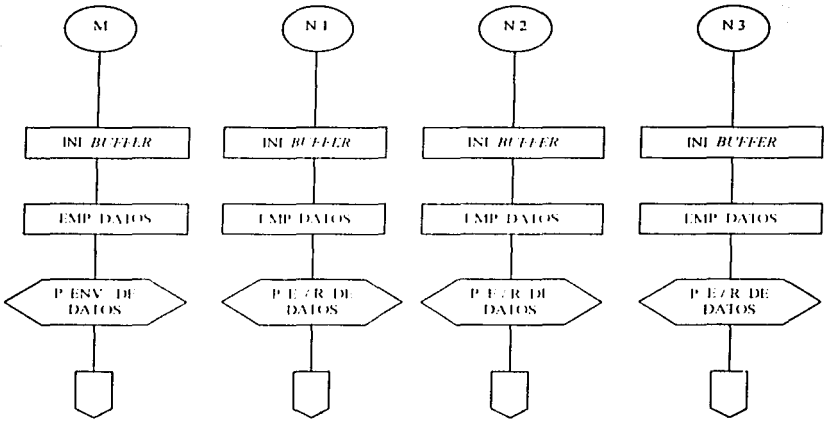
12. – Empaquetado de datos.

13. – Preparación de los datos para la envío y / o recepción.

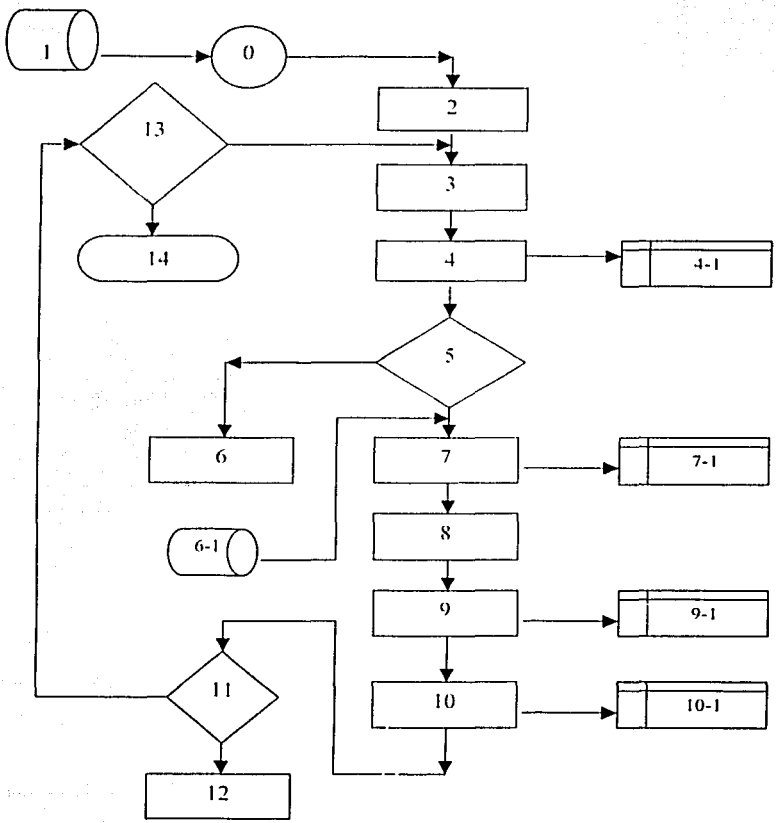
Teniendo los puntos anteriores bien claros, posteriormente continuamos con el planteamiento del método general del sistema que consiste en especificar y localizar los archivos de entrada y salida de cada una de las operaciones importantes o críticas del sistema para así poder monitorearlas cuando se juzgue conveniente, también se describirán cada una de las etapas del sistema.

6.1. DIAGRAMA DE FLUJO DE PREPARACIÓN DEL AMBIENTE PARA *PVM*.




M
N1
N2
N3
INI BUFFER
INI BUFFER
INI BUFFER
INI BUFFER
EMP DATOS
EMP DATOS
EMP DATOS
EMP DATOS
P ENV DE DATOS
P E/R DE DATOS
P E/R DE DATOS
P E/R DE DATOS

Esta propuesta general trata de especificar la forma en que se tomarán y/o leerán los archivos de entrada y definir cuales operaciones o procesos se registrarán en los archivos de salida [L2].



DESCRIPCIÓN DE LA PROPUESTA GENERAL DE ACUERDO CON EL DIAGRAMA DE FLUJO ANTERIOR.

La descripción de las operaciones se realizará de acuerdo a la numeración registrada en el diagrama de flujo anterior y que corresponde a las siguientes [L2]:

0. – Inicio o arranque del sistema.

1. – Lectura de los archivos ASCII de datos de entrada, que proporcione los datos correspondientes al funcionamiento de los algoritmos.

2. – Generar la primera población de LP.

3. – Realizar las operaciones de cruzamiento de LP.

4. – Realizar operaciones de mutación de LP.

4.1. – Almacenar en archivos de salida ASCII, los registros de cruzamientos y de las mutaciones de los LP.

5. – Elección de los LP únicos.

6. – Desechar a los LP repetidos.

6.1. – Lectura de Archivos para el simulador:

a. Archivo ASCII que proporcione los datos al simulador.

b. Archivo binario con el banco de datos nucleares.

c. Archivo binario que requiere el simulador (*MASTER.FILE*).

7. - Realizar simulación con PRESTO.

7.1 – Almacenar, en archivo de datos de salida las simulaciones.

8. – Extraer información de pico de potencia .

9. – Extraer información longitud de ciclo.

9-1. – Almacenar en archivo de datos de salida ASCII, pico de potencia y longitud del ciclo.

10. – Evaluar a los mejores LP, en la función objetivo.

10-1. – Almacenar en archivos de datos de salida ASCII, resultados de la función objetivo.

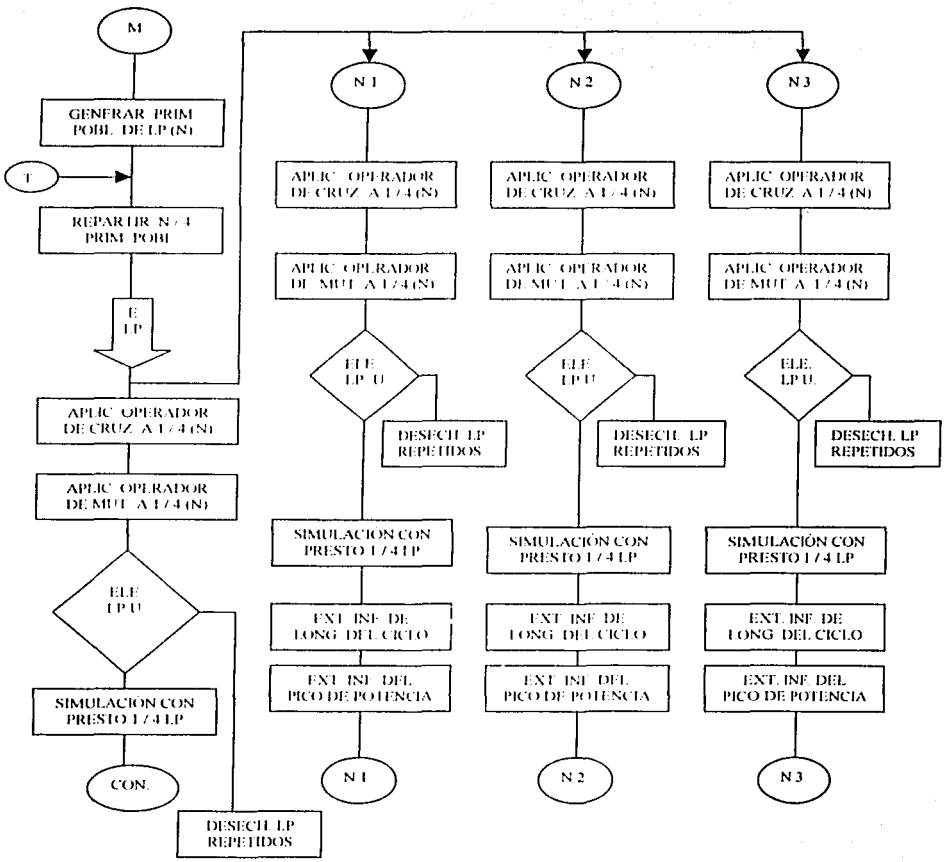
11. – Seleccionar a los mejores LP.

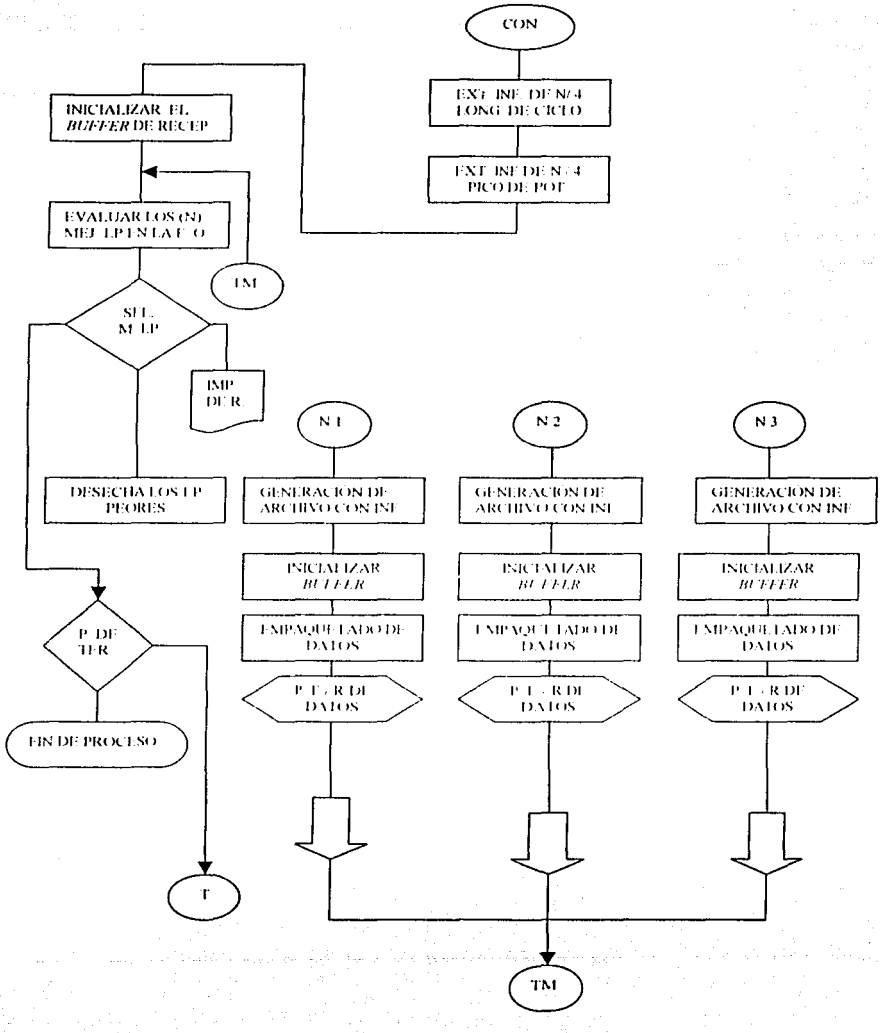
12. – Desechar los peores LP.

13. – Realizar prueba de terminación.

14. – Fin de proceso.

6.2. PROPUESTA 1, DIAGRAMA DE FLUJO DEL SISTEMA PARCIALMENTE DISTRIBUIDO.




CON
EXT INF DE N/4
LONG DE CICLO
EXT INF DE N/4
PICO DE POT
INICIALIZAR EL
BUFFER DE RECEP
EVALUAR LOS (N)
MEJ LP EN LA F O
TM
SEL
M LP
IMP
DER
DESECHA LOS LP
PEORES
P DE
TER
FIN DE PROCESO
T
N 1
N 2
N 3
GENERACION DE
ARCHIVO CON INF
INICIALIZAR
BUFFER
EMPAQUETADO DE
DATOS
P E / R DE
DATOS
TM

DESCRIPCIÓN DEL DIAGRAMA DE FLUJO DEL SISTEMA PARCIALMENTE DISTRIBUIDO.

En este sistema se analizará la forma de cómo los procesos se distribuirán y se deberán de realizar en cada una de las computadoras (maestro, esclavos) [L2].

Descripción de las abreviaturas y operaciones del sistema:

N = LP = población de LP = cantidad de LP = planes de recarga.
M = maestro o servidor.
N1, N2, N3, = esclavos o nodos.

Operaciones que realizará el maestro.

1. Generar la primera población de LP
2. Realizar la división de LP / 4
3. Envío de 3 / 4 (LP) a 3 esclavos
4. Aplicar el operador de cruzamiento a 1 / 4 (LP)
5. Aplicar el operador de mutación 1 / 4 (LP)
6. Elegir LP únicos y desechar LP repetidos 1 / 4 (LP)
7. Simulación con PRESTO 1 / 4 (LP)
8. Extraer información de longitud de ciclo
9. Extraer información de pico de potencia
10. Inicializar *buffer* de recepción de datos
11. Recepción de datos (de 3 esclavos)
12. Evaluar el total de los LP en la función objetivo
13. Seleccionar los LP mejores y desechar los peores e impresión de resultados
14. Prueba de terminación o reiniciar
15. Fin del proceso.

Operaciones que realizarán los esclavos.

En esté caso se describirán las operaciones de un solo nodo(esclavo) ya que las operaciones son similares en los otros dos.

1. **Recepción de 1 / 4 (LP)**
2. **Aplicar operador de cruzamiento 1 / 4 (LP)**
3. **Aplicar operador de mutación 1 / 4 (LP)**
4. **Elegir los LP únicos y desechar LP repetidos**
5. **Simulación con PRESTO**
6. **Extraer información de longitud de ciclo**
7. **Extraer información de pico de potencia**
8. **Generar archivo con información**
9. **Inicializar *buffer***
10. **Empaquetado de LP**
11. **Preparar para señal de envío de datos**
12. **Envío de LP**

Por otra parte, para calcular el tiempo total que tardará nuestro sistema utilizando la alternativa 1, deberemos calcular los tiempos del procesamiento de la siguiente manera.

Si tenemos:

N = N ° de LP = 120
Tiempo de envío en la red = 2 seg. / 10 LP
Tiempo de recepción en la red = 2 seg. / 10 LP

Operaciones realizadas en la computadora (maestro)

Cruzamiento = 1 seg. / 10 LP
Mutación = 1 seg. / 10 LP
Simulación = 5 seg. / 10 LP
Función objetivo = 1 seg. / 10 LP
Otras operaciones = 1 seg. / 10 LP

Operaciones realizadas en cada una de las computadoras esclavos (nodos)

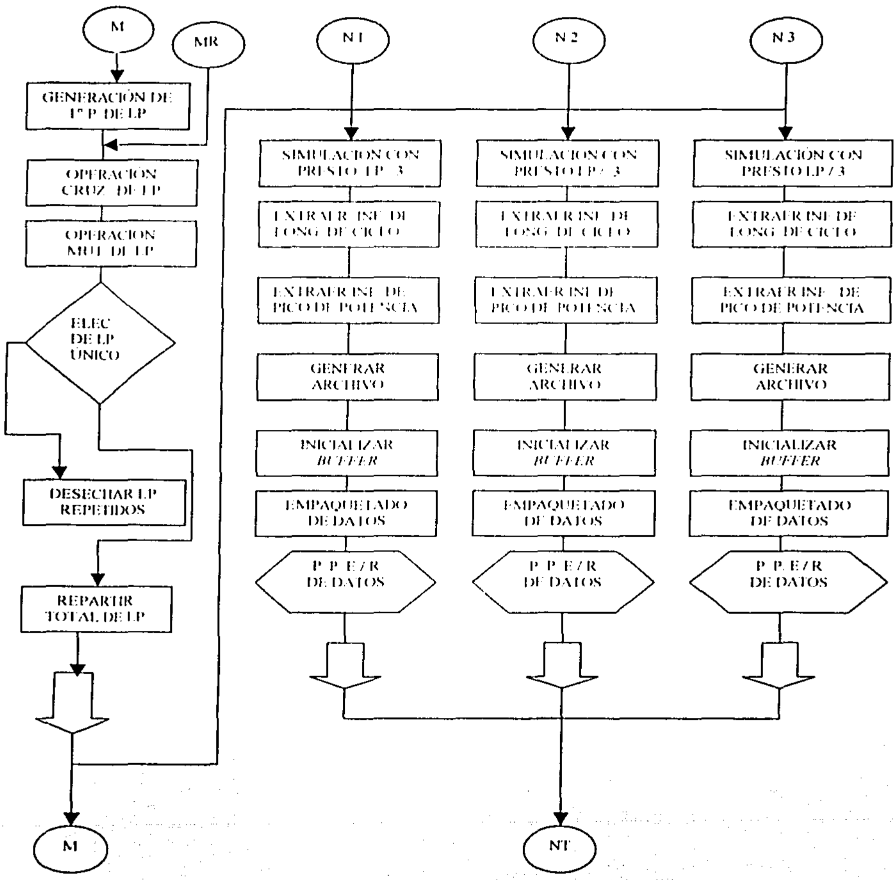
Cruzamiento = 2 seg. / 10 LP
Mutación = 2 seg. / 10 LP
Simulación = 10 seg. / 10 LP.
Otras operaciones = 2 seg. / 10 LP

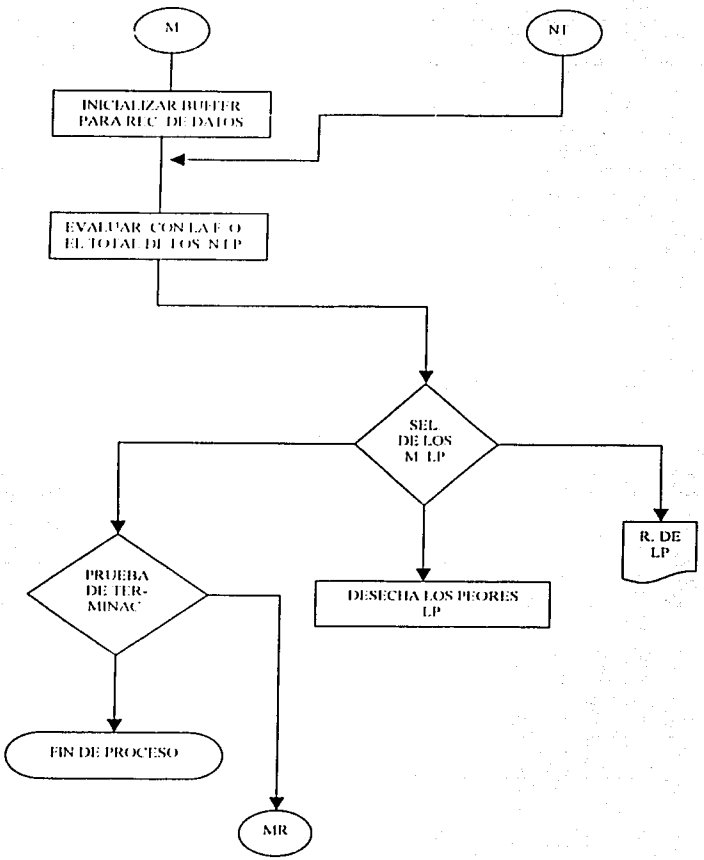
Si realizamos la suma total del tiempo que tardan el maestro, y los esclavos (nodos) tendremos:
Total de operaciones que ejecuta el maestro para cada 30 LP = 9 x 3 = 27 seg.
Como tenemos tres nodos o esclavos (16 seg. / 10 LP) / 3 = 5.3 seg. / 10 LP.

Total de operaciones que ejecutan los nodos para 90 LP = 5.3 x 9 = 47.7 = 48 seg.

Ya que en este caso en particular los nodos tardan en ejecutar sus operaciones con el mayor tiempo, este tiempo es el que se toma como base para los cálculos y para realizar un balance en tiempos de sincronización [L2].
Si nuestro patrón de recargas es de 120 LP, el tiempo total en ejecutar los cálculos será de 48 seg. / 120 LP , si no tuviéramos ningún LP desechado.

6.3. PROPUESTA 2 DIAGRAMA DE FLUJO DEL SISTEMA CON BALANCEO DE PROCESOS.




M
NT
INICIALIZAR BUFFER PARA REC DE DATOS
EVALUAR CON LA F O EL TOTAL DE LOS NTP
SEL DE LOS M LP
R. DE LP
PRUEBA DE TER-MINAC
DESECHA LOS PEORES LP
FIN DE PROCESO
MR

DESCRIPCIÓN DEL DIAGRAMA DE FLUJO DEL SISTEMA CON BALANCEO DE PROCESOS.

En este modelo lo que se pretende es controlar todas las operaciones del sistema con tiempos predeterminados con cronómetro, es decir, si sabemos que:

N = N ° de LP = 120
Tiempo de envío en la red = 2 seg. / 10 LP
Tiempo de recepción en la red = 2 seg. / 10 LP

Operaciones realizadas en la computadora (maestro)

1.Generación de la primera población = 1 seg. / 10 LP
2.Cruzamiento = 1 seg. / 10 LP
3.Elección de los LP únicos Mutación = 1 seg. / 10 LP
4.Función objetivo = 1 seg. / 10 LP
5.Otras operaciones = 1 seg. / 10 LP
(Selección de los mejores LP)

Operaciones realizadas en las computadoras (esclavos)

1. **Simulación = 10 seg. / 10 LP**
2. **Otras operaciones = 2 seg. / 10 LP**
   (Extraer información de long. del ciclo y pico de potencia)

Como se puede observar, la suma total de operaciones, que se realizan en la computadora (maestro) es de cinco, y dejamos que la simulación sea realizada por los esclavos (nodos) entonces a las operaciones restantes les debemos asignar un tiempo fijo, para que el maestro le corresponda un tiempo total de 5 seg. / 10 LP.

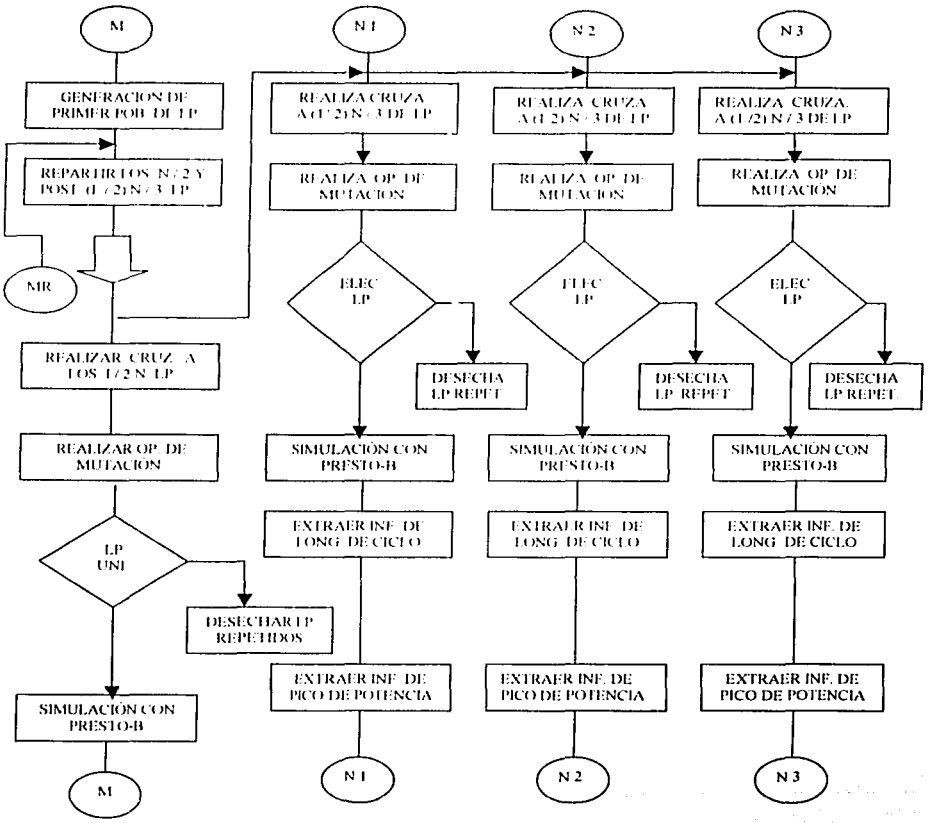
Por otra parte si en cada uno de los nodos, se realizará la simulación y a las demás operaciones, les asignamos un tiempo, entonces su tiempo total será de 12 seg. / 10 LP.

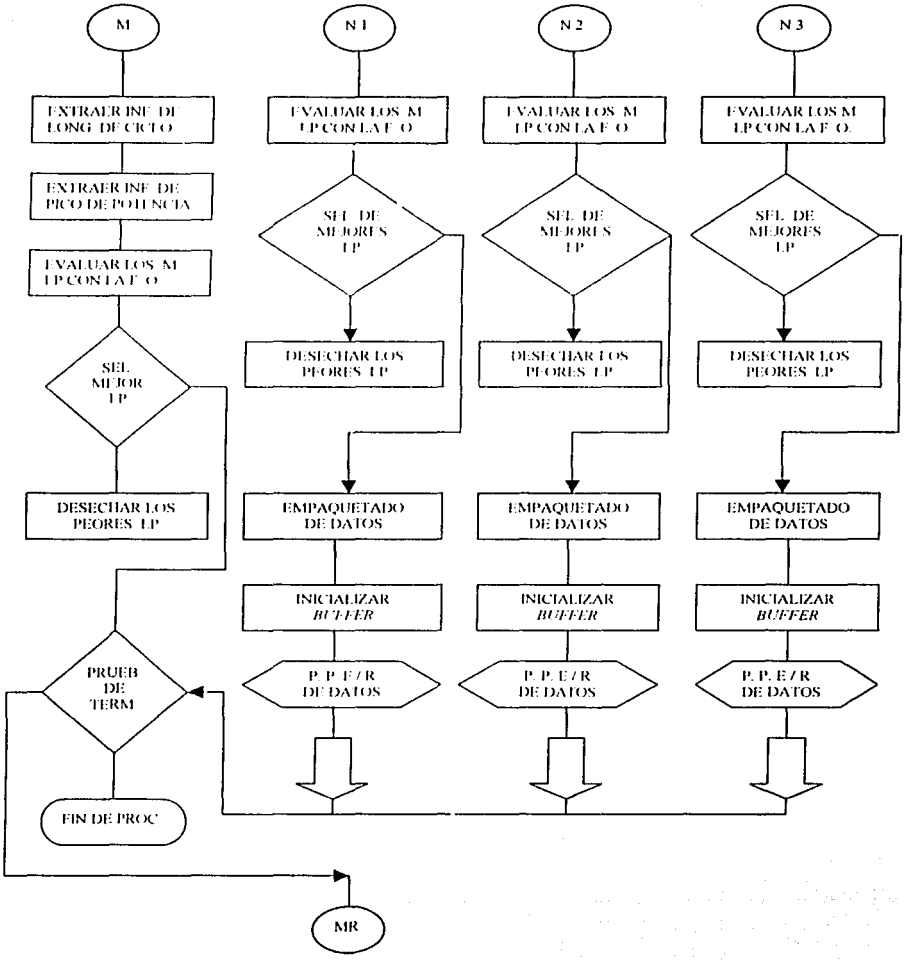
Como tenemos tres computadoras (esclavos) el tiempo total que se requerirá, utilizando las tres computadoras será de 5 seg. / 10 LP.

Analizando el total de operaciones tanto en la computadora (maestro) y (esclavos) tendremos lo siguiente [L2]:

El tiempo total que tardará el sistema en obtener los resultados de 120 LP, contabilizando los tiempos totales de todas las operaciones que realiza nuestro sistema será, aproximadamente de 48 seg. / 120 LP.

6.4.PROPUESTA 3, DIAGRAMA DE FLUJO DEL SISTEMA COMPLETAMENTE DISTRIBUIDO.




M
N 1
N 2
N 3
EXTRAER INF. DE LONG. DE CICLO
EXTRAER INF. DE PICO DE POTENCIA
EVALUAR LOS M LP CON LA F. O.
SEL. MEJOR LP
DESECHAR LOS PEORES LP
EVALUAR LOS M LP CON LA F. O.
SEL. DE MEJORES LP
DESECHAR LOS PEORES LP
EVALUAR LOS M LP CON LA F. O.
SEL. DE MEJORES LP
DESECHAR LOS PEORES LP
EVALUAR LOS M LP CON LA F. O.
SEL. DE MEJORES LP
DESECHAR LOS PEORES LP
EMPAQUETADO DE DATOS
INICIALIZAR BUFFER
P. P. E / R DE DATOS
EMPAQUETADO DE DATOS
INICIALIZAR BUFFER
P. P. E / R DE DATOS
EMPAQUETADO DE DATOS
INICIALIZAR BUFFER
P. P. E / R DE DATOS
PRUEB DE TERM
FIN DE PROC
MR

DESCRIPCIÓN DE LAS OPERACIONES DEL DIAGRAMA FLUJO DEL SISTEMA COMPLETAMENTE DISTRIBUIDO.

A continuación se detalla la forma en que se implementará la propuesta 3.

La preparación e instalación del ambiente de PVM son importantes antes de realizar cualquier operación al sistema. Ya que se instalaron y cargaron los archivos, sistemas operativos y programas correspondientes en cada una de las computadoras, se debe proceder a realizar lo siguiente:

1 - En la computadora que será el servidor (maestro) se generará la primera población de LP.

2 - En el maestro que es más rápido (suponiendo 2 veces más rápido) que los nodos (otras computadoras) se debe de realizar la división de tareas como se especifica a continuación: si N es el total de LP, el servidor realizará la mitad de N y la otra mitad la repartirá enviándola a los tres nodos restantes (N/2)/3. Por ejemplo si N =120, el servidor realizará las operaciones a 60 LP y los demás nodos (computadoras)realizarán las operaciones a 20 LP cada uno. Se deberá hacer un análisis de cuantas veces más rápido es el nodo maestro para repartir la carga de cálculos de la mejor manera [L2].

3 – Las operaciones siguientes se deberán realizar tanto en el maestro como en los nodos al mismo tiempo.

A – Realizar operaciones de cruzamiento.

B– Realizar operaciones de mutación.

C– Elegir a los LP únicos y desechar los LP repetidos.

D- Simulación con PRESTO.

E- Extraer información de la longitud del ciclo.

F- Extraer información del pico de potencia.

G- Evaluar los mejores LP con la función objetivo.

H- Seleccionar los mejores LP y desechar a los peores LP.

I – En los tres nodos se empaquetarán los LP obtenidos anteriormente.

J – Los tres nodos inicializarán el *buffer*.

K – Los tres nodos preparan los LP para recibir la señal de envío del servidor.

L – Como el servidor tiene inicializado su *buffer*, manda una señal a los nodos para el envío de LP.

M – Los tres nodos envían los LP al servidor.

N – El servidor recibe los LP los desempaqueta y realiza la prueba de terminación.

O - Fin del proceso o se repiten todas las operaciones anteriores menos la generación de la primera población de LP.

Por otra parte, para calcular el tiempo total que se tardará nuestro sistema utilizando la alternativa 3, deberemos de calcular los tiempos del procesamiento de la siguiente manera.

Si tenemos que:

N = N ° de LP = 120
Tiempo de envío en la red = 2 seg. / 10 LP
Tiempo de recepción en la red = 2 seg. / 10 LP

Operaciones realizadas en la computadora (maestro) a 60 LP.

Cruzamiento = 1 seg. / 10 LP
Mutación = 1 seg. / 10 LP
Simulación = 5 seg. / 10 LP
Función objetivo = 1 seg. / 10 LP
Otras operaciones = 1 seg. / 10 LP

Operaciones realizadas en cada una de las computadoras (esclavos) a 60 LP.

Cruzamiento = 2 seg. / 10 LP
Mutación = 2 seg. / 10 LP
Función objetivo = 2 seg. / 10 LP
Simulación = 10 seg. / 10 LP.
Otras operaciones = 2 seg. / 10 LP

Si realizamos la suma total del tiempo que tardan el maestro y los esclavos, tendremos:

Total de operaciones que ejecuta el maestro = 9 seg / 10 LP.
Total de operaciones que ejecuta un nodo = 18 seg / 10 LP.
Observamos que el tiempo total de procesamiento en el maestro es de 54 seg / 60 LP y 36 seg / 20 LP en cada nodo, si nuestro patrón de recargas es de 120 LP, entonces nuestro sistema tardará 54 seg / 120 LP en realizar todas las operaciones.

6.5.ANÁLISIS DE LA MEJOR SOLUCIÓN PARA MEJORAR LOS TIEMPOS DE PROCESAMIENTO.

Con base en las tres soluciones de procesamiento en paralelo que fueron descritas en los diagramas de flujo anteriormente referidos, podemos señalar que cada una de ellas tiene sus propias virtudes, pero debemos de elegir una de las tres soluciones planteadas para implementarla a nuestro sistema.

Rendimiento de cada una de las alternativas al realizar el procesamiento de nuestro sistema.

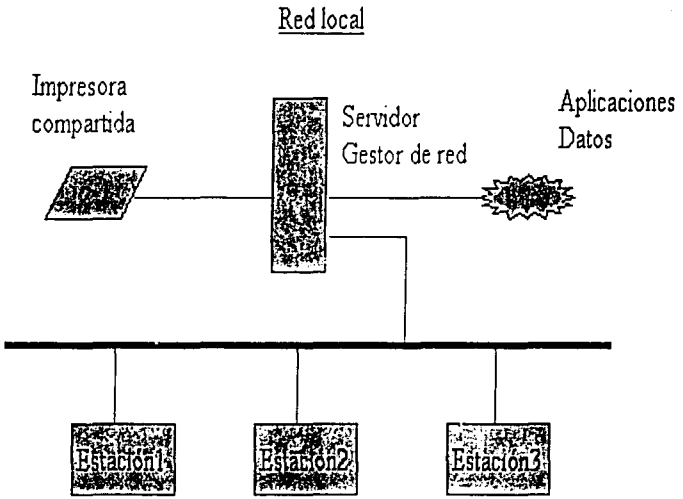
Sistema alternativa 1 "sistema parcialmente distribuido".
Tiempo total de procesamiento por cada 120 LP. = 48 seg.

Sistema alternativa 2 "sistema con balanceo de procesos".
Tiempo total de procesamiento por cada 120 LP. = 48 seg.

Sistema Alternativa 3 "sistema completamente distribuido"
Tiempo total de procesamiento por cada 120 LP. = 54 seg.

De los resultados obtenidos se observa que las tres alternativas son muy parecidas. La elección fue la propuesta 2 (Figura 16) que corresponde al sistema con balanceo de procesos (operaciones, cálculos y procesamiento). Se eligió por ser la mejor alternativa con base al tiempo total que utilizará el sistema para realizar el procesamiento, al tomar como referencia el tiempo de ejecución en las tres alternativas anteriores, se determina que es la mejor solución para aplicarla a SOPRAG. Sin embargo las virtudes de las propuestas 1 y 3, que sugiere utilizar el método completamente distribuido, que requiere un mejor análisis al momento de ser implementado en el diseño del sistema. Actualmente el simulador (PRESTO) no puede ser dividido en sus operaciones internas. Lo que implica que por el momento el balanceo de operaciones no se pueda llevar a cabo utilizando

división de tareas en el simulador, la paralelización en el simulador se debe llevar a cabo si se quiere mejorar las funciones y rapidez en el simulador, lo que sería una buena opción.



Esquema de funcionamiento del sistema propuesto.

Figura 16.

CAPITULO 7

CONCLUSIONES.

Como se observó en el desarrollo del presente trabajo de investigación la idea fundamental fue ejemplificar con varios tópicos y con distintos temas de investigación, una reseña general para poder resolver un problema de ingeniería, partiendo de un tema general hasta concluir con la solución de un caso particular y aplicarlo a el sistema del objetivo propuesto.

Todos los temas que se investigaron me condujeron primeramente a comprender y entender , el origen de la energía nuclear ya que este tema fue algo nuevo para mi tanto en lo profesional, como en lo referente al los estudios que realice en la carrera de ingeniería, de igual manera al adentrarme a la investigación he ir avanzando, se presentaron más temas nuevos, como por ejemplo; los algoritmos genéticos y sus distintas aplicaciones y así como los alcances que ellos involucran, pero creo desde el punto de vista profesional que la visión más importante para mi persona es la aplicación y funcionamiento de la Máquina Virtual Paralela.

Todos los objetivos y alcances planteados al inicio y final, se realizaron con éxito, así también el resultado que se concluye es que se logró plantear la factibilidad de poder utilizar la Máquina Virtual Paralela con varias alternativas de diseño para optimizar el sistema de planes de recarga.

La programación utilizando la Máquina Virtual Paralela, es una herramienta nueva en nuestro país y en gran parte del mundo, será de gran ayuda aplicarla en sistemas que realicen infinidad de cálculos y el aplicarla en cualquier red de computadoras facilitará el procesamiento y mejorará los tiempos de ejecución en cualquier proyecto, investigación o sistema relacionado, no importando su dimensión o estructura de diseño.

Algo muy importante de mencionar de la programación paralela hasta el día de hoy es que es una alternativa que bien aplicada a cualquier sistema facilitará y brindará mejoras en los tiempos de procesamiento, pero talvez el día de mañana sea obsoleto, sí en

nuestro país (México) no llegáramos a utilizarla, posiblemente nos rezagaremos o atrasaremos en comparación con otros países.

Mientras las tecnologías de la computación progresen durante los próximos años, no habrá pausas para reflexionar.

Veremos muchas innovaciones tecnológicas en las arquitecturas de las computadoras y en el desarrollo de más programas y por lo tanto la competencia internacional tenderá a desarrollar y mejorar la velocidad de procesamiento con otras formas de implementaciones ya que cada instante transcurrido las exigencias son cada vez mayores.

Las tentativas posteriores que en este trabajo se sugieren es, la posibilidad de poder paralelizar el simulador (PRESTO) en sus diferentes subrutinas, para así poder aplicar el balanceo de procesos y mejorar el rendimiento a SOPRAG. Debido a que está es la etapa que más tiempo de cómputo utiliza en su funcionamiento el sistema.

Además, habrá que comparar el procesamiento paralelo con una computadora que contenga varias unidades de procesamiento integrados en un mismo gabinete, otra causa importante es no olvidar el tiempo de envió que tarda la información a través de la red, ya que puede ocasionar que el sistema no sea lo rápido que se espera.

Otra de las tentativas que se propone es, utilizar la hibridación de los algoritmos genéticos para así lograr mejores resultados en los cruzamientos, mutaciones y sean mucho más confiables los resultados obtenidos, en el desarrollo donde se apliquen.

En lo personal, esta tesis espero se utilice para contribuir a la solución de problemas relacionados con la ingeniería en computación y con referente a el procesamiento en paralelo.

BIBLIOGRAFÍA.

[B1] Bequelin Adam, Dongarra Jack, Jiang Weicheng, Manchek Robert, Suderam Vaidy, Parallel Virtual Machine (PVM)
(A Users' Guide and Tutorial for Networked Parallel Computing)
Cambridge, Institute Technology Massachusetts
London, England 1994.

[C1] Comisión Federal de Electricidad [Folletos]
" Central nucleoeléctrica de Laguna Verde "
" Que es la energía nuclear "
" Que es una central nuclear "
" Que es el ciclo del combustible nuclear "
" Que es una central nuclear "
Impresos en talleres de artes gráficas grafos, S. A de C. V México,
noviembre 1999.

[C2] Comisión Federal de Electricidad (publicación única)
" Del fuego a la energía nuclear " Impreso en talleres de artes gráficas grafos, S. A. de C. V. noviembre 1999 México.

[C3] Codenotti, Bruno, Introduction to parallel processing, Working ham England: Addison-Wesley 1993

[F1] Francois J. L., H. A. López. SOPRAG: " A System for Boiling Water Reactors Reload Pattern Optimization Using Genetic Algorithms." Annals of Nuclear Energy, Vol. 26/12, pp. 1053-1063, 1999.

[G1] Goldberg, David Edward, Genetic algorithms in reach optimization and machine learning, Reading, Massachusetts: Addison -Wesley Publishing Company 1989.

[H1] Hayes, John Patrick, Computer Architecture and Organization Mc Graw Hill Inc., United States 1998.

[H2] Hwang Kai, Advanced Computer Architecture Parallelism Scalability Programmability, Mc Graw - Hill Inc., Printed Singapore 1993.

[H3] Hwang Kai, Fayé A. Briggs, Arquitectura de computadoras y procesamiento en paralelo, Mc Graw Hill , México 1988.

[L1] Lengthen F. Thomson, Introduction to parallel Algorithms and Architectures Arrays Trees, Hyper cubes, Morgan Kaufmanns Publisher, United States 1992.

[L2] López Sánchez Martín " Sistemas de Control de la producción en los diferentes procesos de fabricación de evaporadores". Tesis para la obtención del Grado de Profesional Técnico en Procesos de producción, (Martín López Sánchez). Colegio Nacional de Educación Profesional Técnica (CONALEP) , Ecatepec Estado de México, Noviembre, 1984.

[L3] López Zamorano H. A., Francois J. L. "Aplicación de la Técnica de Algoritmos Genéticos a la Optimización de Patrones de Recarga de Combustible Nuclear para Reactores Tipo BWR." Octavo Congreso Anual de La Sociedad Nuclear Mexicana. 23-26 de noviembre de 1997. Guanajuato Gto.

[L4] López Zamorano Héctor Arnoldo "Optimación de Recargas de Combustible para Reactores Nucleares Tipo BWR Utilizando Algoritmos Genéticos". Tesis para la Obtención del Grado de Maestro en Ciencias en Ingeniería Nuclear, (Héctor Arnoldo López Zamorano) Escuela Superior de Física y Matemáticas, IPN. Abril, 1999.

[M1] Maccabe, Arthur B. , Computer Systems Architecture Organization and programming, (sistemas computacionales arquitectura y organización) Barcelona, Irwin, España 1995.

[S1] Selling AKI G., The Design and Analysis of parallel Algorithms Prentice Hall, Nueva Jersey, United Estates 1989.

[T1] Tanenbaum Andrew S.
Redes de ordenadores (segunda edición)
Prentice-Hall Hispanoamericana, S.A.
México 1991.

[T2] Temas consultados en la red Mundial de Internet.

[T21] Apuntes de informática

Dirección : http://members.es.tripod.de/klauzen/notas.htlm.

[T22] Los algoritmos genéticos

Dirección : http://www.ocero.org/irbis/disertación/node190.html

[T23] Lenguajes de programación (PVM)

Dirección : http://research.ccm.itesm.mx/jesus/cursos/compd2/s3/node1.html

[T24] Curso de programación paralela

Dirección:http://parallel.cs.buap.mx/acetato1.html

[T25] La máquina MIMD

Dirección :http://agamenon.uniandes.edu.co/~c21437/ma-bedoy/mind.html

[T26] La energía de los núcleos de los átomos

Dirección:

http://
omega.ilce.edu.mx:3000/sites/ciencia/volumen3/ciencia3/119/html/sec.14.html