Computación y Sistemas Vol. 3 No.3 pp. 169 - 181
© 2000, CIC - IPN. ISSN 1405-5546 Impreso en México

Sistema Manejador de Ambientes Reconfigurables para Procesamiento Paralelo/Distribuido

F.J. Hidrobo
SUMA. Facultad de Ciencias
Universidad de los Andes.La Hechicera.
5101. Mérida. Venezuela.
e-mail: hidrobo@ciens.ula.ve

J.L. Aguilar
CEMISID
Universidad de los Andes. La Hechicera.
5101. Mérida. Venezuela.
e-mail:aguilar@ing.ula.ve

Artículo recibido el 19 de febrero, 1999; aceptado el 29 de noviembre, 1999

Resumen

En este artículo se presenta el diseño de un Sistema Manejador de Ambientes Reconfigurables para procesamiento Paralelo/Distribuido. Así, se proponen dos enfoques de diseño, uno en el que el programa se adapta al sistema, y otro que sigue el procedimiento inverso. Los enfoques son basados en la teoría de grafos, por lo que se presentan varios problemas de tipo NP-completos a ser resueltos en la implementación del sistema. Un primer problema es el de particionamiento de grafos (agrupamiento de las tareas), y otros dos problemas son la determinación del máximo D-acoplamiento o del ciclo hamiltoniano del grafo de agrupamiento. Se usan los Algoritmos Genéticos como técnica para encontrar buenas soluciones a dichos problemas.

Palabras Clave: Ambientes reconfigurables, Procesamiento paralelo y distribuido, Gestión de procesos, Sistemas operativos distribuidos, Algoritmos Genéticos.

1 Introducción

La utilización de máquinas masivamente paralelas en el campo científico se ha incrementado aceleradamente en los últimos años. Muchas aplicaciones secuenciales de altos requerimientos computacionales están siendo migradas a plataformas paralelas. Sin embargo, es posible observar que estos procesos de migración o de construcción de programas paralelos refleja, en muchos casos, características particulares de la plataforma de comunicación en la cual se deberían ejecutar (plataforma ideal de ejecución); es decir, las aplicaciones están atadas a una topología de comunicación. En este sentido, se necesitaría una amplia variedad de topologías para dar servicio a todas las posibles aplicaciones (Baiardi *et al.*, 1994; Talbi, 1994; Aguilar, 1995; Hidrobo y Aguilar, 1996).

Por otro lado, el rendimiento de un programa paralelo depende, en gran medida, de la configuración de la arquitectura sobre la cual se este ejecutando (Aguilar y Kluol, 1997; Baiardi *et al.*; 1994; Brazier y Johansen, 1994; Casavant y Singha, 1994; Aguilar y Gelenbe, 1997). El desarrollo de programas paralelos con la restricción de tener que adaptarse a una topología comunicacional fija es un procedimiento difícil y poco natural. La búsqueda del mejor algoritmo para resolver un problema dado, no debe tener restricciones topológicas. Esto plantea la necesidad de disponer de plataformas reconfigurables que den un mayor grado de flexibilidad y disponibilidad.

Una arquitectura paralela es *reconfigurable* si la topología de su red de interconexión puede cambiar en función de los programas que van a ejecutarse sobre ella, o bien, si se pueden realizarse transformaciones en los programas para que se adapten a la topología sin perdidas significativas en sus rendimientos (Kramer, 1985; Adano y Trejo, 1994; Baiardi *et al.*, 1994; Talbi, 1994; Hidrobo y Aguilar, 1996). Las arquitecturas paralelas

reconfigurables han sido usadas en multiprocesadores a memoria compartida. Estos últimos años han empezado a ser usados en arquitecturas a memoria distribuida. Por consiguiente, son muchos los esfuerzos que se están realizando en las áreas de arquitectura de computadores y sistemas operativos dirigidos, principalmente, a hacer mas eficientes estas plataformas (Tanenbaum, 1995; Casavant y Singha, 1994; Brazier y Johansen, 1994; Stallings, 1997). No obstante, los problemas asociados a la adaptación de las aplicaciones a las plataformas o viceversa están lejos de ser completamente resueltos.

En este sentido, son varios los trabajos que se encuentran en la literatura para solventar estos problemas de adaptación (reconfiguración) a nivel de sistema operativo (Kramer, 1985; Adano y Trejo, 1994; Baiardi *et al.*, 1994; Talbi, 1994; Aguilar, 1995; Johnson *et al*, 1996; Aguilar y Gelenbe, 1997; Aguilar y Kluol, 1997; Stallings, 1997), los cuales son problemas que no se presentan en los sistemas operativos para máquinas secuenciales. Dichos mecanismos adaptativos forman parte del sistema de gestión distribuida de procesos en los sistemas operativos distribuidos o paralelos. En este trabajo nosotros nos dedicamos a estudiar estos problemas, y proponemos mecanismos de solución basados en la teoría de grafos. Así, el trabajo apunta hacia la conceptualización y el modelado de las plataformas de procesamiento paralelo/distribuido, así como al análisis y solución de los problemas asociados a la adaptación de las aplicaciones a dichas plataformas, o viceversa.

2 Marco Teórico

2.1 Motivación

Un Sistema Operativo Distribuido (SOD) gobierna la operación de un sistema informático distribuido y proporciona una abstracción de máquina virtual a sus usuarios (Tanenbaum, 1995; Stallings, 1997). El objetivo base de un SOD es la transparencia, es decir, que los componentes y recursos se puedan acceder en el momento y lugar en que sean requeridos. Además, los SOD proporcionan medios para compartir recursos.

En los SOD existen un conjunto de aspectos claves a considerar no comunes en los sistemas operativos tradicionales; entre ellos tenemos la gestión distribuida de procesos, la cual permite utilizar eficientemente los diferentes recursos del sistema por parte de las aplicaciones. Entre los aspectos a considerar para realizar esta tarea, tenemos (Brazier y Johansen, 1994; Casavant y Singha, 1994; Tanenbaum, 1995; Stallings, 1997):

- **Equilibrio de la carga:** es una de las alternativas al distribuir la carga total en un sistema (la otra es la repartición de la carga). En este caso, lo que se busca es mantener una carga total equilibrada entre los diferentes recursos del sistema, tomando en consideración las características particulares de cada procesador, de la aplicación, del sistema de comunicación, etc. La Figura 1 muestra la situación que puede presentarse por desequilibrio de carga; en este caso, existen algunos nodos que tienen un nivel de ocupación mucho mayor que otros. Cuando se desea optimizar el rendimiento de la plataforma de procesamiento, se debe asegurar que la carga esta uniformemente repartida (procesadores homogéneos), ya que de esta manera todos los procesadores tendrán un uso similar y no habrá procesos compitiendo por un procesador mientras haya otros libres.

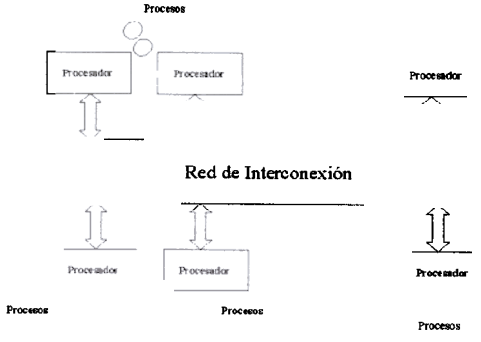


Figura 1: Ambiente de procesamiento sin equilibrio de carga

- **Migración de procesos:** es la transferencia del estado de un proceso desde una máquina a otra, para que el proceso pueda continuar su ejecución en la máquina destino. Esta migración se realiza, principalmente, para asegurar el equilibrio de la carga o para garantizar la culminación de las tareas (por falla en algún procesador), aunque la aplicación de este concepto se extiende más allá de estos campos.

- **Asignación de procesos:** es el proceso de decisión que se realiza con cada proceso en el sistema, para asignarle un procesador o algún otro recurso (Aguilar y Kluol, 1997). Para realizar tal decisión se pueden usar muchos criterios, tales como equilibrio de la carga en el sistema, minimización de la comunicación entre procesos comuminantes, minimización de la interferencia entre procesos concurrentes, etc. En la literatura, han sido implementados muchos esquemas para realizar estas tareas. Algunos de ellos están en (Aguilar, 1995; Aguilar y

Gelenbe, 1997; Aguilar y Kluol, 1997).

- **Descomposición de programas y Multiprogramación:** Las máquinas paralelas permiten, en principio, que un programa pueda ser *dividido* en tareas las cuales pueden ejecutarse simultáneamente (Casavant y Singha, 1994; Aguilar, 1995; Stallings, 1997). Sin embargo, es probable que el número de tareas obtenidas sea mayor que el número de procesadores disponibles; por lo cual algunas tareas deben esperar por algún procesador o competir por este (Casavant y Singha, 1994; Aguilar, 1995; Stallings, 1997). Esto último es conocido como *multiprogramación.* Así, cuando se desea ejecutar varias tareas en una misma máquina puede emplearse la multiprogramación, a sabiendas de que esto podría degradar significativamente el rendimiento de cada procesador y/o programas por la interferencia en el procesamiento de tareas que deberían ser ejecutadas en paralelo.

- **Interconexión entre los procesadores:** La forma de interconexión de los procesadores del sistema tiene un gran influencia en el ancho de banda y punto de saturación de las comunicaciones en el sistema (Brazier y Johansen, 1994; Casavant y Singha, 1994). En este sentido, para la tarea de gestión de recursos, la escalabilidad es la propiedad más relevante a estudiar, entendiéndose como la capacidad del sistema de soportar el mayor número de enlaces entre procesos comunicantes ubicados en distintos lugares.

- **Tolerancia a fallas:** un sistema falla cuando alguno de sus componentes tiene un problema (mal funcionamiento, etc.). En ciertos sistemas, una falla puede ser catastrófica; así, hacerlos tolerantes a fallas es fundamental. Existen diferentes nociones que deben ser usadas para hacer un sistema tolerante a fallas, tales como *la migración de procesos, la reconfiguración de máquinas*, etc (Tanenbaum, 1995; Stallings, 1997). En este último caso, lo que se busca es tomar en cuenta la nueva topología de interconexión después que una falla a ocurrido, sea a nivel de los procesadores o de los canales de comunicación, para que los programas puedan continuar su ejecución.

- **Manejo del sistema reconfigurable:** otro aspecto clave a definir es el mecanismo que se usará para asegurar una ejecución eficiente de los procesos. En este sentido, existen dos esquemas a seguir, uno a través del cual el Ambiente de procesamiento se adapta al flujo de ejecución de los programas, (en particular, su sistema de interconexión por lo cual debe ser reconfigurable); y el otro en el cual el programa es el que se adapta al ambiente de procesamiento a través de un esquema de agrupación de procesos y asignación de los mismos al sistema según ciertos criterios. En este trabajo, nos dedicaremos a hacer propuestas para ambos casos.

2.2 Ambientes Reconfigurables de Procesamiento Paralelo/Distribuido

Tradicionalmente, cuando se habla de arquitecturas paralelas reconfigurables, se hace referencia a aquellos sistemas donde la red de interconexión no es fija, sino que es definida en función del programa que se va a ejecutar (Kramer, 1985; Adano y Trejo, 1994; Baiardi *et al.*, 1994; Talbi, 1994; Hidrobo y Aguilar, 1996). Es decir, la reconfigurabilidad de ambientes de procesamiento paralelo se refiere básicamente a la posibilidad de definir la topología de interconexión de los procesadores para cada aplicación. Dicha definición se refiere al número de procesadores y los enlaces entre ellos; de esta manera quedará identificada una topología particular.

Es posible que la definición hecha pueda realizarse de forma real, es decir, los procesadores se interconectan físicamente como lo indica la topología especificada; en este caso se estaría hablando de *reconfiguración por hardware o reconfiguración por hardware programable (switching).* Alternativamente, la definición topológica puede hacerse completamente por software, implementándose a través de una *topología virtual.* En este caso, la creación y la manipulación de dicha topología se hace por *software.*

Bajo el esquema de topología virtual, la interconexión real de los procesadores sólo es conocida por el **Sistema Manejador del Ambiente** (SMA). El SMA debe determinar la topología sobre la cual se debe ejecutar una aplicación dada, para generarla virtualmente. Además, realizarátodas las tareas de gestión sobre esa topología, de manera de optimizar el tiempo de ejecución de la aplicación.

Si la reconfiguración se realiza antes de iniciarse la ejecución del programa, se esta hablando de una *reconfiguración estática*, de lo contrario, si ésta es modificable en el curso de la ejecución de un programa y/o de la operación del sistema, se habla de una *reconfiguración dinámica.*

2.2.1 Caracterización de los Ambientes de Procesamiento Paralelo/Distribuido

Un ambiente de procesamiento Paralelo/Distribuido (APPD) puede definirse como una plataforma compuesta de diversos recursos (Procesadores, Sistemas de Comunicación, Discos, Memoria, Sistema Operativo, etc.) que proveen servicios computacionales a aplicaciones

que resuelven problemas específicos (Brazier y Johansen, 1994; Casavant y Singha, 1994; Tanenbaum, 1995). La descripción de un APPD se hace en término de los componentes del mismo y de la interrelación entre éstos. A continuación, se presentan algunos de los componentes de un APPD:

- Unidades de procesamiento (*CPU's*)
  Responsable de las tareas de control y ejecución de los procesos. Caracterizadas, principalmente, por la capacidad de memoria y la velocidad relativa (la relación entre la velocidad del CPU y la velocidad de algún CPU que sea tomado como referencia para todos).
- Sistema de interconexión
  Representa los mecanismos, la topología y la tecnología usada para la comunicación entre las unidades de procesamiento.
- Aplicaciones de gestión (Ejemplo: Asignación de tareas, Balance de cargas , etc.)
  Estas permiten definir los esquemas/políticas utilizadas para administrar los recursos en el sistema, con el fin de optimizar el rendimiento.
- Estructura de las aplicaciones (paradigma de programación paralela)
  Expresa el flujo de ejecución y las características comunicacionales de las aplicaciones. Normalmente, dicha estructura es representada usando un grafo de tareas, con la respectiva descripción de cada tarea y los requerimientos de comunicación entre las mismas.

2.2.2 Sistema Manejador del Ambiente (SMA)

El SMA es el *ente* que realiza todo el procesos de adaptación entre el APPD y las aplicaciones en el sistema de gestión distribuido; por consiguiente, sirve como mediador entre la plataforma paralela y las aplicaciones del usuario (Kramer, 1985; Baiardi *et al.*, 1994; Tanenbaum, 1995; Hidrobo y Aguilar, 1996). Específicamente, el SMA debe optimizar el uso de los recursos (CPUs) minimizando el tiempo de ejecución de múltiples aplicaciones con diversos requerimientos. Desde el punto de vista global, el SMA debe considerar:

- *Cambios en la Estructura del Sistema:* son derivados por modificaciones en los recursos que posee el sistema, debido a fallas o mantenimiento de estos, nuevos recursos en el sistema, etc.
- *Optimización del Uso de los Recursos:* de esta manera se responde a necesidades tales como equilibrar la carga en el sistema, minimizar los tiempos de ejecución de los programas, etc.
- *Cambios dinámicos en el Código de Ejecución:* esto puede generar cambios en la jerarquía de ejecución de los procesos de un programa (implicando, posiblemente, un cambio en la topología de comunicación) y en la carga de trabajo en el sistema.
- *Cambios debido a la Multiprogramación:* Esto ocurre cuando llegan nuevos programas al sistema o cuando culminan viejos programas, lo que implica demanda/liberación de recursos, aumento/disminución de la carga de trabajo en el sistema, etc.

3 Propuesta de diseño del Sistema Manejador del Ambiente

A continuación se presenta el esquema global de funcionamiento del SMA, y de los módulos que lo componen. Además, se describe detalladamente el núcleo administrador del sistema.

3.1 Funcionamiento del SMA

El funcionamiento del sistema esta compuesto por las tres etapas siguientes: caracterización del sistema y de las aplicaciones, reconfiguración del sistema y asignación de las tareas (figura 2).

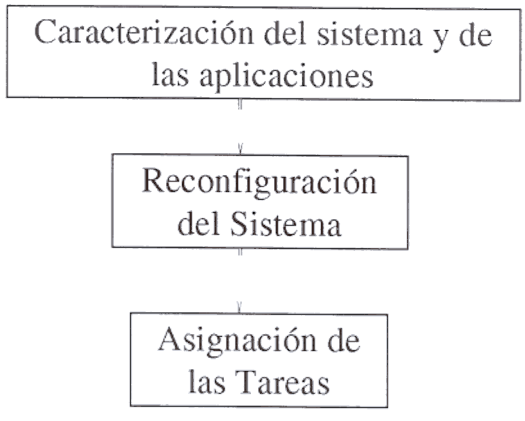


Figura 2: Módulos del SMA

3.1.1 Caracterización del sistema y de las aplicaciones

Una aplicación paralela/distribuida puede modelarse mediante un grafo $G_p = (V_p, E_p)$ donde los vertices representan las tareas (procesos) y los pesos de los vertices representan los tiempos de ejecución (estimados o conocidos) para esas tareas(ver figura 3) (Aguilar, 1995;

Johnson *et al*, 1996). Los arcos representan los requerimientos comunicacionales entre los procesos, en particular, sus pesos modelan la cantidad de información a transferir entre los procesos. Este esquema de representación asume que el grafo de la aplicación es estático, puesto que no hay generación dinámica de procesos.

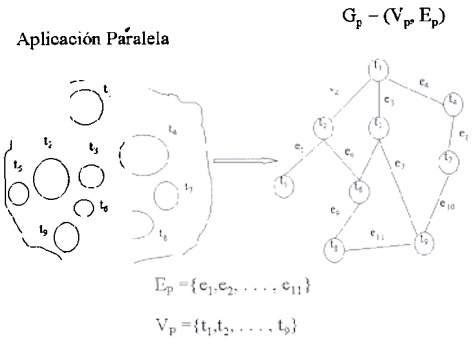


Figura 3: Representación de la aplicación

Bajo un esquema similar, la arquitectura paralela se modela a través de un *grafo conexo no dirigido* $G_t = (V_t, E_t)$, donde los vertices representan procesadores y los arcos enlaces de comunicación entre los mismos, esto se ilustra en el ejemplo de la figura 4. De esta manera, el sistema quedará descrito por:

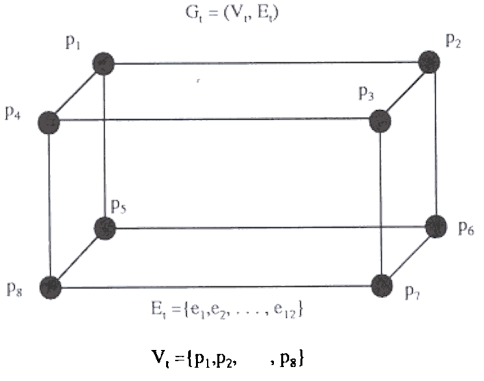


Figura 4: Ejemplo de representación de la arquitectura mediante grafos

- Las características de la arquitectura paralela: número de enlaces máximo por procesador (D_{max}), número de procesadores en el sistema (K), topología de interconexión de los procesadores (G_t), etc.
- El número de tareas de los programas (n) y el flujo de ejecución de las tareas, conocido como el grafo de los procesos (G_p)

3.1.2 Reconfiguración del Sistema

La reconfiguración del sistema es la parte fundamental de la propuesta, llamado también *núcleo administrador del sistema*, el cual comprende la agrupación de los procesos y el acoplamiento (mapeo) del grafo de los procesos sobre la topología de interconexión de la plataforma. La reconfiguración se está entendiendo como un proceso que de forma transparente realiza las transformaciones necesarias para optimizar el rendimiento de la aplicación sobre la plataforma. Se ha separado el funcionamiento de este módulo en dos etapas, como se muestra en la figura 5, una de análisis estático y otra de acoplamiento topológico. En la siguiente sección se explicará detalladamente este módulo.

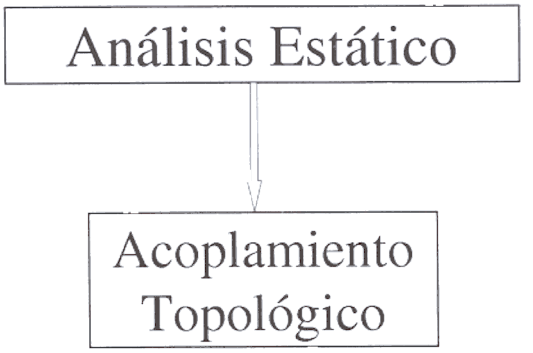


Figura 5: Etapas del módulo de reconfiguración

3.1.3 Asignación de las tareas

En esta fase, se asignarán las tareas a los procesadores según lo indicado por el grafo de agrupamiento y el acoplamiento hechi en la fase anterior, creándose y ejecutándose los procesos de control necesarios, lo cual involucra la definición de una tabla de procesos para la aplicación y una entrada en la tabla general de aplicaciones para el sistema. Es en esta etapa donde cada proceso es enviado a un procesador, registrándose la información referente a la ubicación, tiempo de creación, estado, etc.

3.2 Diseño del Núcleo administrador del sistema

Como se dijo antes, el núcleo administrador del sistema se encarga de la etapa de reconfiguración del mismo, el cual debe resolver los problemas de agrupación de las tareas y acoplamiento de la topología del grafo de ejecución a la plataforma.

En nuestro SMA proponemos dos alternativas para el desarrollo del núcleo administrador. Una primera alternativa trata de adaptar el grafo de ejecución de las tareas a la topología real del sistema (adaptación de la aplicación), y una segunda alternativa genera una topología virtual de interconexión de los procesadores la cual se adapta al grafo de ejecución de las tareas (adaptación del sistema). En este último caso, se debe hacer la gestión de esa topología virtual.

3.2.1 Esquema general

Como se indicó en la sección 3.1.2, el esquema general de funcionamiento del núcleo administrador consta de las dos etapas, las cuales son:

- *Etapa de Análisis Estático:*. En esta etapa se determinan y calculan los parámetros de evaluación de la función objetivo (Costos de Comunicación, de Ejecución, etc).
- *Etapa de Acoplamiento Topológico:* En esta etapa, según la alternativa escogida, se pasa a la fase de adaptar al sistema o al programa:
  - Primera Alternativa: (adaptación de la aplicación)
    * *Agrupación de tareas:* tal que el número de grupos de tareas sea el mismo que el número (K) de procesadores en el sistema. Este nuevo grafo se denominarágrafo de agrupamiento G_p^1.
    * *Verificación de las Restricciones del Sistema Físico:* en este caso, es necesario eliminar el menor número de arcos del grafo de agrupamiento, tal que se adapte al número máximo de enlaces (D_{max}) que puede tener cada procesador del sistema. Además, se debe asegurar que el grafo de resultante sea conexo para garantizar que no existen grupos de tareas aislados que no podrán comunicarse como estaba previsto en el grafo de la aplicación.
  - Segunda Alternativa: (adaptación del sistema)
    * *Agrupación de tareas:* Esta fase es la misma que para la otra alternativa.
    * *Generación de la Topología Virtual:* Una vez que se tiene el grafo de agrupamiento, se genera la topología virtual sobre la plataforma computacional que satisface los requerimientos del grafo.

3.2.2 Especificación detallada de las alternativas

La primera parte del núcleo administrador, que consiste en *agrupar las tareas*, se modela como un problema de partición de grafos (ver sección 3.2.3) donde el número de grupos de tareas es igual al número de procesadores del sistema. Así, la agrupación de tareas es resuelta como un problema de partición de grafo, donde los vertices del grafo representan las tareas, los arcos representan las comunicaciones entre las tareas y k (número de particiones) representa la cantidad de procesadores.

La segunda parte, en la cual se intenta *mapear el grafo de ejecución sobre la topología*, puede ser desarrollada bajo dos enfoques distintos, el primero basado en la verificación de las restricciones del sistema, y el segundo basado en la topología virtual. En ambos casos, el mapeo del grafo de agrupamiento sobre la topología de interconexión se convierte en un algoritmo de asignación. En el caso de verificación de las restricciones del sistema se presentan dos alternativas.

- **Esquema basado en la verificación de las restricciones del sistema**

  La parte de *verificación de las restricciones del sistema* es compleja. En este caso, lo que se busca es que el grafo de agrupamiento de tareas sea lo más parecido al grafo de interconexión de los procesadores del sistema. Este problema puede ser definido como un problema de *máximo D_{max}−acoplamiento*, en el cual, lo que se busca es minimizar el número de arcos eliminados del grafo de agrupamiento tal que el grado de cualquier nodo de ese grafo sea menor o igual a D_{max} (D_{max} representa el grado máximo del grafo de interconexión de la arquitectura). El grafo final que se obtiene de este proceso tiene que ser conexo. Este problema, con esta última restricción es NP-completo (Narsingh, 1974; Johnson y Gakey, 1979; Foulds, 1984), por lo que se proponen dos algoritmos para resolverlo:

  1. **Primer Algoritmo:**

     Buscar el *Circuito Hamiltoniano máximo* del grafo de agrupamiento (ver sección 3.2.3). En este caso, se supone que el grafo de la plataforma tiene un circuito hamiltoniano para poder realizar el mapeo

directo entre estos dos circuitos. El circuito encontrado será el nuevo grafo de agrupamiento G_1^a.

- Agregar arcos del grafo de agrupamiento a G_1^a mientras que el grado de cualquier nodo de este grafo sea menor o igual a D_{max}, tratando de minimizar a la vez, el costo de comunicación en el sistema.
- Asignar los vertices de G_1^a a los vertices de P.

2. **Segundo Algoritmo:**
   - Buscar un D_{max}−*acoplamiento de cardinalidad máxima* para el grafo de agrupamiento (ver sección 3.2.3), es decir, con el mayor número de arcos posibles. Esto da el borde superior de la configuración final del grafo de agrupamiento.
   - Añadir arcos tal que se asegure que el grafo de agrupamiento sea conexo y que no se viole la restricción de que cualquier v_i del grafo final tenga un grado menor o igual a D_{max}.
   - Asignar los vertices de G_1^a a los vertices de P.

- **Esquema basado en topología virtual**

El algoritmo basado en topología virtual trata de asignar los nodos (vertices) del grafo de agrupamiento sin necesidad de realizar eliminaciones de arcos. Así, todos los arcos del grafo de agrupamiento permanecerán en la asignación definitiva. Esto se logra estableciendo enlaces entre nodos no vecinos del grafo de la plataforma. El macroalgoritmo que describe el funcionamiento de este esquema se presenta a continuación:

1. Seleccionar el vertice de mayor grado v_{max} (aleatorio si hay más de uno) del grafo de agrupamiento G^a.
2. Asignar v_{max} al vertices de mayor grado del grafo de la plataforma p_{max}.
3. Asignar los vecinos de v_{max} ($S(v_{max})$) a los vertices vecinos de p_{max} ($S(p_{max})$). Si la cardinalidad de $S(v_{max})$ es mayor que la cardinalidad de $S(p_{max})$, la diferencia se asignará a la vecindad de $S(p_{max})$.
4. Eliminar v_{max} de G^a (los arcos que inciden en v_{max} también serán eliminados).
5. Seleccionar un nuevo v_{max} de $S(v_{max})$ siguiendo el mismo criterio del paso 2, y el nodo donde este es asignado como el nuevo p_{max}.
6. Repetir el proceso desde 3 hasta que todos los vertices de G^a hayan sido asignados a un vertice de P.

3.2.3 Problemas NP-Completos asociados a las alternativas

Como se mostró antes, existen varios problemas NP-completos que deben ser resueltos por el SMA. Particularmente, en el esquema de funcionamiento en dos etapas del núcleo administrador del sistema. En la primera etapa es necesario generar el grafo de agrupamiento de las tareas, el cual se define como un problema de **partición de grafos**. En la segunda etapa, en el caso de verificación de las restricciones del sistema, se debe resolver alguno de los problemas NP-Completos siguientes: **Circuito Hamiltoniano Máximo** o **Máximo D-acoplamiento** . A continuación se describen los problemas, el mecanismo de resolución empleado en cada caso y como son usados por el SMA:

- **Partición de grafos**

Dado un grafo G, lo que se desea es particionar sus vertices en K subgrafos, de tal manera que la suma de los pesos de los vertices por subgrafo sea igual (de ser posible) y la suma de los pesos de los arcos de vertices que estén en diferentes subgrafos sea mínima. De esta manera, lo que se busca es encontrar un agrupamiento que minimice tanto la comunicación entre grupos como el desbalance de la carga de trabajo del sistema. En nuestro caso, este problema se resuelve usando los algoritmos propuestos en (Hidrobo y Aguilar, 1998), en los cuales se describen varias heurísticas paralelas basadas en Algoritmos Genéticos (AGs) para resolver problemas de Optimización Combinatoria. Cada una de las heurísticas presentadas en esos trabajos explotan diferentes conceptos del paradigma de programación paralela, tales como descomposición del espacio de datos, explotación del paralelismo implícito, incorporación de conceptos ligados a la inteligencia colectiva, etc.

El esquema general que se sigue, es el siguiente:

1. Se toma como entrada el grafo que representa a la aplicación.
2. Se aplica una AG para resolver el problema de partición de grafos, el cual minimiza la siguiente función de costo:

$$F_C = \sum_{i,j \in D} a_{ij} + b\frac{\sum_{z=1}^{K} (N_{G_z} - \sum_{i=1}^{n} p_i/K)^2}{K} \quad (1)$$

donde:

a_{ij} = peso del arco entre el vertice i y el vertice j.
$D = \{i \in G_m \ \& \ j \in G_l \ \& \ l \neq m\}$ tal que, G_m y G_l son subgrafos de G.
N_{G_z} = suma de los pesos de los vertices en el subgrafo G_z
b = factor de equilibrio, $0 \leq b \leq 2$
K = número de subgrafos

3. La mejor solución que da el AG define el grafo de agrupamiento.

- **Circuito Hamiltoniano Máximo**
El circuito hamiltoniano es un camino cerrado que pasa por todos los vertices de un grafo una y solo una vez. El circuito hamiltoniano máximo representa el circuito que involucra los arcos de mayor peso (Narsingh, 1974; Johnson y Gakey, 1979; Foulds, 1984).

En términos de nuestro sistema, lo que se quiere es encontrar un circuito hamiltoniano máximo en el grafo de agrupamiento, con el objeto de mapear directamente la mayor cantidad posible de arcos del grafo de agrupamiento (comunicaciones) sobre la topología de la plataforma. Esta alternativa puede aplicarse siempre y cuando la plataforma tenga un circuito hamiltoniano. En nuestro caso, para resolver este problema se toma como base el AG para el agente viajero propuesto en (Hidrobo y Aguilar, 1998). A continuación, se describe el procedimiento que sigue el SMA:

1. Se toma como grafo de entrada el grafo de agrupamiento generado en la etapa 1.
2. Se aplica un AG para resolver el problema de hallar el circuito hamiltoniano máximo, el cual maximiza la siguiente función objetivo:

$$F_C = \sum_{i,j \in D} a_{ij} \qquad (2)$$

donde:

a_{ij} = peso del arco entre el vertice i y el vertice j.
$D = todoslosverticesdelgrafotalque\{i \neq j\}$

3. La mejor solución dada por el AG se toma como el circuito hamiltoniano máximo.
4. El circuito hamiltoniano máximo se mapea sobre el circuito hamiltoniano de la plataforma tomando como puntos de inicio el procesador con mayor número de enlaces y el nodo del grafo de agrupamiento con mayor volumen de comunicación. Así, se realiza la asignación de cada nodo del grafo de agrupamiento a un procesador de la plataforma.

- **Máximo D-acoplamiento**
El problema de acoplamiento consiste en hallar un conjunto de arcos, los cuales no son adyacente entre ellos, por lo cual, no tienen vertices en común (Narsingh, 1974; Foulds, 1984; Johnson y Gakey, 1979). El problema de máximo $D-$acoplamiento de un grafo G=(V,A) es un subconjunto de A (A^*) tal que para todo v_i que pertenezca a V, el subconjunto de arcos de A^* del cual v_i es una extremidad $A^*(v_i)$, verifica la siguiente restricción: $A^*(v_i) <= D$, de tal forma que ese subgrafo A^* es el de cardinalidad máxima entre todo el conjunto de subgrafos posibles de A.

En el problema de asignación de tareas, lo que se desea es hacer un mapeo óptimo de las tareas de una aplicación sobre los procesadores de una plataforma. En estos términos, el problema de máximo D-acoplamiento puede ser utilizado como base para definir un mapeo que maximice el número de enlaces de la plataforma que son utilizados, minimizando el número de arcos del grafo de agrupamiento que deben ser eliminados.

El acoplamiento máximo tiene como objetivo el uso máximo de los enlaces de la plataforma. Para este caso, los individuos del AG representarán soluciones al problema de máximo D-acoplamiento. Para la codificación de los individuos se usará un arreglo de valores enteros, donde la posición i del arreglo representa al procesador i y el valor del arreglo en la posición i contiene el nodo del grafo de agrupamiento que ha sido asignado al procesador i.

A continuación se presenta el macroalgoritmo que sigue el SMA para resolver este problema:

1. Se toma como grafo de entrada el grafo de agrupamiento generado en la etapa 1.
2. Se aplica un AG para resolver el problema de máximo D-acoplamiento, el cual maximiza la función objetivo:

$$F_C = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ij} * t_{u_i u_j} \qquad (3)$$

donde:

a_{ij} = peso del arco entre el vertice i y el vertice j en el grafo de la aplicación.

$t_{ij} = 1$ si el procesador i tiene enlace directo con el procesador j, de lo contrario 0.

u_i = número del proceso que se asignó al procesador i.

Además, D_{max} es el número máximo de enlaces de los procesadores de la plataforma.

3. La mejor solución del AG es el máximo D_{max}-acoplamiento.
4. Dicha solución es usada para hacer la asignación directa de los nodos del grafo de agrupamiento a los procesadores de la plataforma.

4 Pruebas y resultados

4.1 Descripción de las aplicaciones

Desde el punto de vista experimental, es necesario caracterizar las aplicaciones a las cuales el sistema dará servicios. Así, deben definirse los mecanismos de representación y los tipos de aplicaciones en el sistema.

4.1.1 Estructura de las aplicaciones

Las aplicaciones se representaran a través de un grafo (ver sección 3.1.1). Para la implementación, se tiene un arreglo de n elementos, cada uno de esos elementos es una tarea con el tamaño de la misma (p.e: su tiempo de ejecución). La matriz de adyacencia se puede convertir en una matriz de pesos, lo cual indicará el tiempo que se requiere para la comunicación entre dos tareas dadas. Un valor de cero (0) en dicha matriz indicara que no existe comunicación entre esas dos tareas.

4.1.2 Tipos de aplicaciones

La aplicaciones se clasifican tomando en cuenta tres parámetros, cuyos valores pueden variar según tres rangos: alto, medio y bajo. Los parámetros y sus rangos son:

- Número de tareas (**NT**): Cuantos procesos, concurrentes o no, componen la aplicación. Sus rangos son, alto: (50 - 70), medio: (20 - 30) y bajo: (5 - 10)
- Tamaño promedio de las tareas (**TPT**): Tiempo promedio de ejecución, en minutos, de las tareas en un procesador de velocidad relativa 1. Sus rangos son, alto: (40 - 60), medio: (15 - 25) y bajo: (1 - 8)
- Volumen de comunicación promedio por tarea (**VCP**): Porcentaje de tiempo, respecto al tiempo de ejecución, que se emplea para comunicar dos tareas dadas. Sus rangos son, alto: (30 - 50), medio: (15 - 20) y bajo: (2 - 10)

Tomando en cuenta los parámetros anteriores y las aplicaciones comunes en un APPD, se han definido los siguientes tipos de aplicaciones:

1. Grano fino con muchos procesos y poca comunicación. (NT= alto, TPT=bajo, VCP=bajo)
2. Grano fino con alta dependencia de datos. (NT= alto, TPT=bajo, VCP=alto)
3. Grano medio con comunicación media. (NT= media, TPT=media, VCP=media)
4. Grano medio con alta dependencia de datos. (NT= media, TPT=media, VCP=alto)
5. Grano grueso con muchas tareas y poca dependencia de datos. (NT= alto, TPT=alto, VCP=bajo)
6. Grano grueso con pocas tareas y muy poca comunicación. (NT= bajo, TPT=alto, VCP=bajo)

Además, se define un parámetro que refleja la cantidad de enlaces que tiene cada tarea; el cual permite medir el volumen de arcos del grafo de la aplicación, su profundidad y su uniformidad (en cuanto a número de arcos por nodo). De esta manera, se describen tres grupos:

número bajo de arcos por tarea (2-4)

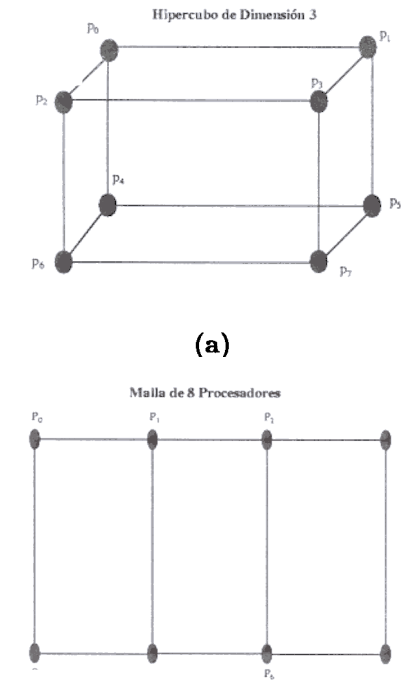
número alto de arcos por tarea (6-8)

número de arcos por tarea que va entre mediano y alto (5 - 10)

Con este parámetro adicional, se generan tres categorías por cada tipo de aplicación.

4.2 Evaluación del núcleo administrador del sistema

Las pruebas del núcleo administrador del sistema persiguen, por una parte comprobar la efectividad del esquema propuesto, y por otro lado, comparar las diversas alternativas en la fase de mapeo para diferentes características de las aplicaciones y de la plataforma del ambiente. Para cumplir tales objetivos, se simuló un ambiente computacional paralelo que permite comparar las alternativas según dos aspectos: sus tiempos de ejecución y el volumen de comunicación que no puede ser mapeado directamente del grafo de ejecución de la aplicación al grafo de interconexión de los procesadores.



(a)

(b)

(c)

Figura 6: Ejemplos de Arquitecturas utilizadas

El ambiente computacional paralelo es descrito por los procesadores, como ellos están interconectados (topología de interconexión), y si existe o no un circuito Hamiltoniano en la topología de interconexión. Con esos datos, el sistema calcula D_{max}. Además, recibe como entrada las características de la aplicación, para que sean generadas por el sistema para simular la carga de trabajo en el ambiente. Dichas características son:

Número de tareas de la aplicación

Tamaño promedio de las tareas.

Porcentaje del tiempo empleado para comunicaciones en cada tarea.

Número de enlaces promedio por tarea

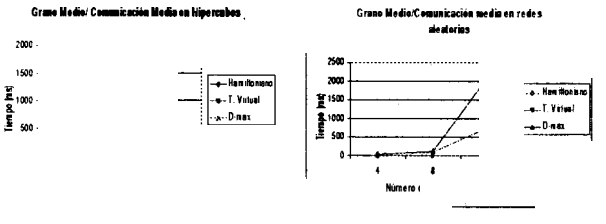
Finalmente, el ambiente es ejecutado con cada alternativa (excepto que alguna no pueda ser probada, por ejemplo, la basada en Circuito Hamiltoneano cuando no existe dicho circuito en la topología de interconexión) 30 veces. Los números que aparecen en las tablas son el promedio de esas ejecuciones para cada instancia ejecutada. Para realizar las mediciones se tomaron los rangos de valores para las aplicaciones descritos en la sección 4.1; agregándose aplicaciones con un número de tareas superior a 100. En cuanto a las plataformas de prueba, se seleccionaron las siguientes:

- Hipercubos de dimensión 2, 3 y 4 (ver figura 6.a)
- Mallas: de 8 y 16 procesadores, ya que la de 4 es un hipercubo de dimensión 2 (ver figura 6.b).

Además, se generaron arquitecturas aleatorias de 4, 8 y 16 procesadores (ver figura 6.c) para medir, de alguna forma, el comportamiento de los algoritmos para estructuras desconocidas.

4.3 Resultados y Análisis

Debido a la gran cantidad de resultados, sólo se mostrarán los más representativos de los estudios realizados. Como se dijo antes, para la obtención de cada punto en nuestras gráficas se realizaron 30 simulaciones por instancia del problema.

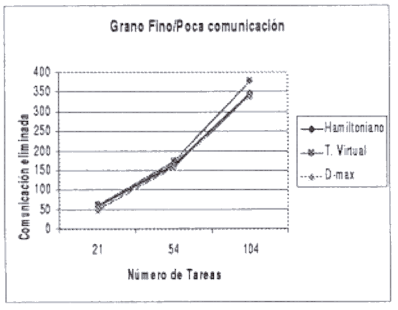


(a) (b)

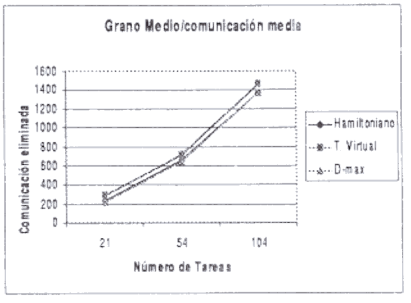
Figura 7: Tiempo de ejecución vs. Número de procesos, para NT= medio y VCP=medio

Tanto en la Figura 7(a), como en la figura 7(b), se observa que el tiempo de duración de la alternativa basada en la topología virtual es siempre cero (0), lo que era de esperarse, puesto que dicha alternativa no involucra un procesamiento complejo. En cuanto a las alternativas que requieren soluciones a problemas NP-completos, se observa que la de D-acoplamiento, tiene en todos los casos una mayor duración que la basada en circuito hamiltoniano máximo. La diferencia entre los tiempos de estas dos alternativas se debe, fundamentalmente, a que la de D-acoplamiento utiliza una función de evaluación

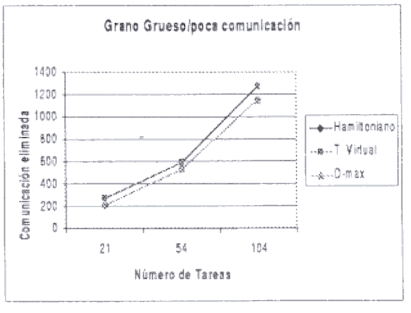
más compleja que la de circuito hamiltoniano. En cuanto al orden de magnitud de los tiempos, se presenta el incremento esperado de los mismos a medida que aumenta el número de procesadores (dimensión del problema), el cual es exponencial.



(a)



(b)

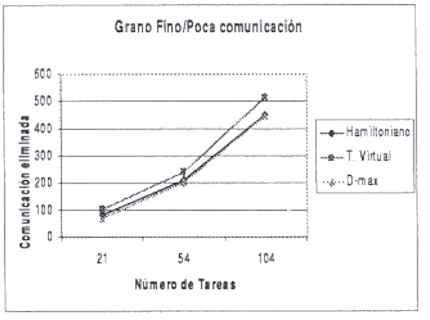


(c)

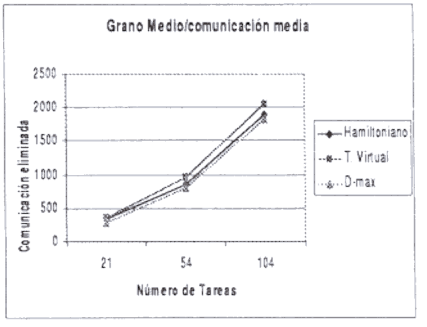
Figura 8: Volumen eliminado vs. Número de tareas, para hipercubo de dimensión 3 y número medio de arcos por tarea

Para las observaciones referentes al volumen de comunicación que es eliminado en cada alternativa (por lo cual, deberá ser enrutado), se presentan a continuación las gráficas para tres tipos de aplicaciones.

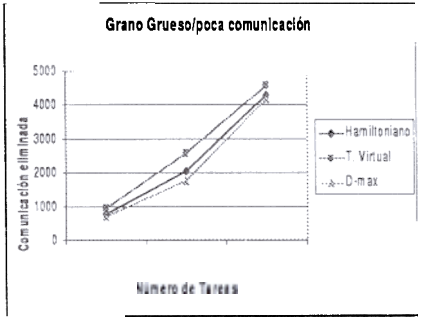
Las Figuras 8 y 9 muestran que las alternativas de circuito hamiltoniano máximo y máximo D-acoplamiento siempre obtienen un mejor resultado que la de topología virtual. La deficiencia de la topología virtual, en cuanto a sus resultados, se debe a que este esquema trata de minimizar el volumen eliminado en cada paso sin tomar en cuenta el volumen global, cosa que si hacen los otros dos esquemas.



(a)



(b)

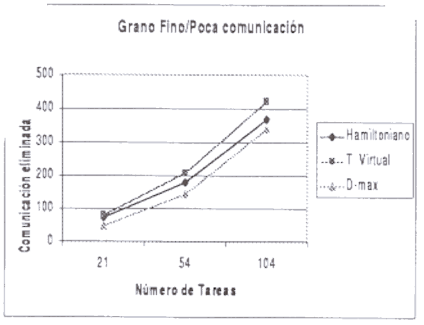


(c)

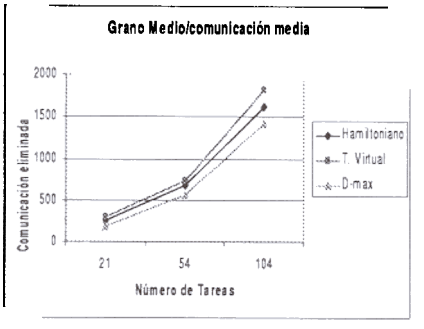
Figura 9: Volumen eliminado vs. Número de tareas, para malla de 16 procesadores y número medio de arcos por tarea

En las Figuras 8 y 9, no se aprecia una diferencia significativa entre D-acoplamiento y circuito hamiltoniano. Esta diferencia se puede observar en la Figura 10 que muestra que con máximo D-acoplamiento siempre se obtiene un resultado igual o mejor al de circuito hamiltoniano máximo. Por lo tanto, D-acoplamiento aventaja a la alternativa de circuito hamiltoniano, puesto que

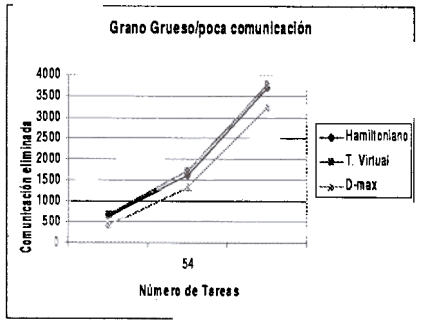
da tan buenos, o mejores resultados que esta última, y puede utilizarse aún cuando la plataforma no tenga un circuito hamiltoniano. Los mejores resultados de D-acoplamiento se deben a que esta alternativa trata de minimizar el volumen eliminado globalmente; mientras que con circuito hamiltoniano se consigue un circuito máximo que no toma en cuenta los enlaces que no pertenecen a dicho circuito. Para gran número de comunicaciones entre tareas el comportamiento es semejante a los otros dos, por eso no se grafican resultados de ese tipo.



(a)



(b)



(c)

Figura 10: Volumen eliminado vs. Número de tareas, para redes aleatorias de 16 procesadores y número medio de arcos por tarea

En todas las arquitecturas Malla e Hipercubo con 4 procesadores, las alternativas basada en ciclo hamiltoniano máximo y D_{max}-acoplamiento siempre dan los mismos resultados. Para estos casos, la dimensión del problema hace que la solución para ambas alternativas sea la misma, ya que las soluciones al problema D_{max}-acoplamiento, con D_{max} igual a 2, es un circuito hamiltoniano.

5 Conclusiones

En este trabajo se han descrito dos enfoques para resolver el problema de adaptación en los APPD, uno en el que la aplicación se adapta al sistema y otro que sigue el procedimiento inverso. La propuesta presentada involucra la resolución de los problemas de particionamiento, y asignación y/o mapeo de procesos. Todos estos problemas son de tipo NP-completos, los cuales se resolvieron usando Algoritmos Genéticos.

En la evaluación del núcleo administrador del sistema, es claro, según los resultados, que la mejor alternativa es la basada en el algoritmo de máximo D-acoplamiento, puesto que resuelve los problemas de asignación con el menor costo; sin requerir que la plataforma y el grafo de agrupamiento tengan algún circuito hamiltoniano. Sin embargo, debe destacarse el incremento que tiene el tiempo de ejecución de esta alternativa, el cual podría ser significativo para plataformas de muchos procesadores; en cuyo caso puede establecerse un compromiso entre el tiempo de duración del algoritmo y la calidad de la solución (tiempo de ejecución de la aplicación).

Estos resultados abren el camino a la elaboración de sistemas complejos de manejo distribuido de recursos; los cuales incorporarían a las propuestas anteriores mecanismos sofisticados de toma de decisiones y de colaboración. Así, nuestra propuesta podrá ser incorporada a un Sistema Operativo Distribuido o Paralelo dentro de su módulo de *gestión distribuida de procesos*. En ese caso, a nuestro sistema deberá incorporarsele mecanismos de planificación, de determinación del estado global, de toma de decisiones para la selección alternativas, etc; dependiendo de las situaciones en el sistema y de las características de las aplicaciones.

Referencias

Adano, J. and **Trejo L.**, *"Programming Environment for Phase-Reconfigurable Parallel Programming on Supernode"*, *Journal of Parallel and Distributed Computing*, Vol 23, pp 273-292, 1994.

Aguilar, J., *"Parallel Programs: task graph generation & task allocation"*, *Proceeding of the* 10^{th} *International Conference on Mathematical & Computer Modelling and Scientific Computing*, pp 854-860, July 1995.

Aguilar, J. and **Gelenbe E.**, *"Task Assignment and Transaction Clustering Heuristics for Distributed Systems", Information Science Informatics & Computer Science*, Vol 97, No 1, pp 199-219, 1997.

Aguilar, J. y **Kluol L.**, *"Estudio de Problema de Asignación de Tareas en los Sistemas Distribuidos: funciones de costo y métodos de resolución", Revista Técnica de la Facultad de Ingeniería de la Universidad del Zulia*, Vol 20, No 3, pp 203-213, 1997.

Baiardi, F., Ciuffolini D., Lomartine A., Montanari A., and **Pesce G.**, *"A Tool for Parallel System Configuration and Program Mapping based on Genetic Algorithms", En Programming Enviroments for Massively Parallel Distributed Systems*, Monte Verita, pp 379-383, 1994.

Brazier, F. and **Johansen D.**, *"Distributed Open Systems", IEEE Computers Society Press*, 1994.

Casavant, T. and **Singha M.l**, *"Distributed Computing Systems", IEEE Computers Society Press*, 1994.

Foulds, L.R., *"Combinatorial Optimization for Ungraduates", Springer-Verlag*, 1984.

Hidrobo, F. y **Aguilar J.**, *"Algorithms for adapting Parallel Programs to the interconnection topology of Parallel Processing Systems", Proceedings of the International Conference on Informatic Systems Analysis & Synthesis*, pp 499-504, July 1996.

Hidrobo, F. and **Aguilar J.**, *"Toward a Parallel Algorithm based on Collective Intelligence for Combinatorial Optimization Problems", Proceeding of IEEE International Conference on Evolutionary Computation*, pp 715-720, May 1998.

Johnson, D. and **Gakey M.**, *"Computers and intractability: A guide for the theory of NP-Completeness", W.H Freeman & Company.* 1979.

Johnson, T., T. Davis, and **Hadfield S.**, *"A concurrent dynamic task graph", Parallel Computing*, Vol 22, pp 327-333, 1996.

Kramer, J. and **Majee J.**, *"Dynamic Configuration for Distributed Systems", IEEE Transaction on Software Engineering*, Vol II, No 4, 1985.

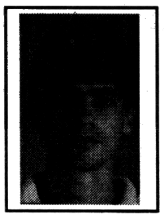
Narsingh D., *"Graph Theory with Applications to Engineering and Computer Science". Prentice-Hall, INC*, Englewood Cliffs, 1974.

Stallings, W., *"Sistemas Operativos", 2da. Edición, Prentice Hall*, 1997.

Talbi, E., *"Configuration Automatique d'une architecture parallèle", Calculateurs Parallèles*, Vol 3, pp 7-26, 1994.

Tanenbaum, A., *"Distributed Operating Systems", Prentice Hall*, 1995.

F. J. Hidrobo *es profesor de la Universidad de Los Andes e investigador del Centro de Cálculo Científico de esa misma universidad. Autor de varias monografías, artículos en congresos internacionales y en revistas especializadas. Trabaja en las areas Computación Inteligente y Computación Paralela-Distribuida; con infasis en la aplicación de técnicas de computación inteligente para el manejo de ambientes de procesamiento paralelo. Recibis el título de Ingeniero de Sistemas en 1993 y un M.S. en Computación en 1998 ambos de la Universidad de Los Andes.*



J. L. Aguilar *nacio en Valera-Venezuela. Se graduo de Ingeniero de Sistemas en 1987 en la Universidad de los Andes-Merida-Venezuela, de Master en Computacion en 1991 en la Universidad Paul Sabatier-Toulouse-Francia, y de Doctor en Computacion en 1995 en la Universidad Rene Descartes-Paris-Francia. El es un Profesor Asociado del Departamento de Computacion de la Universidad de los Andes e investigador del Centro Nacional de Calculo de Venezuela (CeCalCULA). Actualmente, es el un profesor invitado del Departemento de Computacion de la Universidad de Houston, donde es el un investigador postdoctoral. De 1995 a 1999 ha sido investigador visitante en diferentes universidades Francesas (Proyecto de Cooperacion CNRS-CONICIT). El ha sido coordinador de varios eventos cientificos y coeditor de varias actas cientificas. El es autor de mas de 70 articulos cientificos. Sus areas de investigacion incluyen computacion inteligente (Redes Neuronales,*

Computacion Evolutiva, Logica Difusa) y Sistemas Paralelos (Administracion de E/S, Optimizacion de Desempeno, Asignacion de tareas, etc.)