UNIVERSIDAD AUTONOMA DE BARCELONA

FACULTAD DE CIENCIAS

Departamento de Informática

Políticas de "Scheduling" Estático para Sistemas Multiprocesador

Memoria presentada por Porfidio Hernández Budé para optar al grado de Doctor en Informática

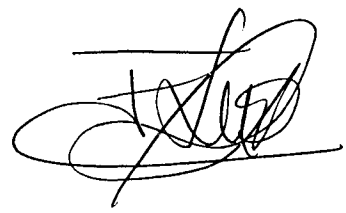
Barcelona, Octubre 1991

POLÍTICAS DE "SCHEDULING" ESTÁTICO PARA SISTEMAS MULTIPROCESADOR

Memoria presentada por PORFIDIO HERNANDEZ BUDE para optar al grado de Doctor en Informática por la Universidad Autónoma de Barcelona. Trabajo realizado en el Departamento de Informática de la Facultad de Ciencias de la Universidad Autónoma de Barcelona, bajo la dirección del Dr. Emilio Luque Fadón.

Bellaterra, Octubre de 1991

Vo. Bo. Director Tesis



Fdo. Emilio Luque

Deseo expresar mi agradecimiento al Dr. D. Emilio Luque Fadón por su magistral dirección y constante crítica, que ha hecho posible la realización de este trabajo.

A la Dra. Dña. Ana Ripoll Aracil por sus acertadas sugerencias y constante estímulo.

A D. Tomás Margalef Burrull por su colaboración en el desarrollo del software utilizado en la experimentación realizada en este trabajo, y sus valiosos comentarios en el desarrollo general de la investigación llevada a cabo.

Al Dr. D. Joan Sorribes Gomis por el apoyo recibido.

A D. Remo Lucio Suppi por sus sugerencias y apoyo técnico recibido.

A todos los miembros de la Unidad de Arquitectura de Ordenadores y Sistemas Operativos, por su aliento y constante apoyo .

INDICE

Prólogo

Capítulo I: Procesamiento paralelo 1

- Introducción 1

1.1.- Métricas de control 3

1.2.- Factores que influyen en el cómputo paralelo 6

1.2.a.- Arquitectura 6

1.2.b.- Características de los algoritmos 11

1.2.c.- Lenguajes paralelos 14

1.2.d.- Políticas de planificación y partición de programas paralelos 18

1.2.e.- El problema del "Scheduling" 20

1.3.- Formulación del problema y objetivo del trabajo 41

1.3.1.- Tiempos de comunicación despreciables 42

1.3.2.- Elección del número de procesadores 43

Capítulo 2: "Schedulers" basados en lista. Sensibilidad 45

- Introducción 45

2.1.- Justificación de los "Schedulers" de lista 46

2.2 "Schedulers" basados en lista 48

2.3.- Algoritmo CP/MISF 53

2.4.- Sensibilidad de los algoritmos de lista frente al tiempo de ejecución 55

Capitulo 3: Fundamentos del método propuesto 66

- Introducción 66

3.1.- Método para resolver el problema de la sensibilidad 67

3.2.- Algoritmos desarrollados 72

3.2.1.- Descripción del algoritmo CP/MISF/TD 74

3.2.2.- Descripción del algoritmo CP/MISF/TD/RC 86

3.2.3.- Requerimientos de sincronización 94

3.3.- Aplicación de los algoritmos CP/MISF/TD y CP/MISF/TD/RC sobre un conjunto de grafos 94

3.4.- Asunciones y planteamientos a la hora de medir 120

3.5.- Mecánica de experimentación 121

Capítulo 4: Aproximación del modelo de DAG a la situación real. Adaptabilidad de los algoritmos de duplicación 130

- Introducción 130

4.1.- Representación de secciones cuyo tiempo de ejecución puede variar dentro de un rango de valores 133

4.1.a.- Generalización de los métodos de duplicación de tareas: Replicación 134

4.1.b.- Extensión de los métodos de duplicación a un rango de valores 137

4.1.1.- Estudio realizado para analizar la familia de grafos 138

4..1.2- Comparación de ambos métodos. Análisis de resultados 140

4.2.- Experimentación realizada y resultados obtenidos 141

Conclusiones del capítulo 146

Aportaciones 147

Lineas abiertas 150

Referencias 151

Apéndice A a1

Apéndice B b1

Prólogo

Uno de los problemas que debe ser tratado si se desea obtener el potencial de cómputo que ofrecen las arquitecturas paralelas, es el relacionado con el mejor conocimiento de las interrelaciones entre algoritmos paralelos y arquitecturas paralelas, así como el estudio de políticas de planificación ("scheduling") de tareas a procesadores. En este trabajo, se refleja la investigación realizada en el estudio de políticas de planificación estática, para sistemas multiprocesador, basadas en métodos heurísticos.

El problema del "scheduling" está siendo estudiado desde hace más de veinte años, aunque muchos de los logros en este area han tenido lugar en los últimos cinco años. Muchas de las técnicas que se aplican hoy día en el campo de las políticas de "scheduling" para sistemas multiprocesador, son adaptaciones de otras más antiguas.

Nuestro trabajo estudia el problema del "scheduling" de tipo determinístico y de naturaleza no apropiativa, para aplicaciones donde existen relaciones de precedencia entre las tareas, asumiendo, que el tiempo de ejecución de algunas de ellas puede variar entre un conjunto de valores conocidos, con objeto de desarrollar algoritmos de "scheduling" estático, más acordes con la naturaleza de las aplicaciones reales. En este sentido, proponemos algoritmos heurísticos obtenidos mediante la aplicación de los mecanismos clásicos de "scheduling de lista". En esta linea, presentamos algoritmos de "scheduling" de diferentes complejidades, y basados en distintos principios introduciendo duplicación de tareas. Igualmente, mostramos la eficiencia de los algoritmos propuestos presentando los resultados obtenidos para grafos sintéticos unos, y correspondientes a aplicaciones reales otros.

CAPITULO I

PROCESAMIENTO PARALELO

-Introducción

El estudio del cómputo distribuido en las áreas de predicción metereológica, simulaciones aerodinámicas, defensa militar, bases de datos distribuidas, inteligencia artificial, visión por computador, procesamiento de imágenes e ingeniería genética entre otras muchas disciplinas ha crecido de forma significativa en la última década. Muchos de los retos en el campo del conocimiento humano no podrían ser abordados en un tiempo razonable sin la potencia que nos proporcionan los superordenadores. Aquí tiempo "razonable" significa el máximo tiempo que la máquina puede estar dedicada a una persona, o el que el científico puede esperar entre experimentos.

Hoy por hoy, el procesamiento paralelo es la forma más prometedora de mejorar el rendimiento de un computador ahora que los avances tecnológicos en materia de semiconductores se están aproximando a su inevitable límite físico.

La búsqueda de arquitecturas avanzadas, teorías de procesamiento paralelo, asignación óptima de recursos, implementación de algoritmos rápidos y lenguajes de programación eficientes que satisfagan el requerimiento de las aplicaciones antes citadas, con una relación rendimiento/coste razonable, constituyen la piedra filosofal de la investigación en el área de arquitectura de computadores.

Desde muy al comienzo del desarrollo de la computación digital, los diseñadores tuvieron presente el incremento de la velocidad de las operaciones. Existen diversas formas a la hora de conseguir este objetivo: mejorar la tecnología, refinar el diseño lógico de los algoritmos con objeto de obtener mayores velocidades en las operaciones de suma, multiplicación, etc; no obstante, existen otras alternativas a la hora de conseguir mejores velocidades de computación que consisten en realizar tantas operaciones como sea posible de forma concurrente o paralela.

En las arquitecturas Von Neuwman tradicionales, las operaciones a nivel de la ejecución de instrucciones eran realizadas en forma secuencial; no existían solapamientos entre las distintas fases de búsqueda, decodificación y ejecución de las instrucciones. Los primeros esfuerzos en el estudio de la concurrencia fueron realizados a este nivel. En algunos sistemas había disponibles varias unidades aritmético-lógicas con objeto de operar en paralelo (IBM 360/91, CDC 6600).

Un paso adelante en la búsqueda de paralelismo lo aporta la noción de "pipeline" donde los n estados del "pipe" pueden operar simultaneamente. Como se ha intentado mostrar anteriormente hay más de una forma de mejorar la velocidad de las operaciones en un computador digital. Las posibilidades se incrementan de manera significativa cuando los diseñadores comienzan a introducir más de una CPU dentro del mismo sistema. Podemos pues definir "procesamiento paralelo" como el método de organizar las operaciones en el sistema de computación donde más de una operación puede realizarse de forma concurrente; no obstante el término "procesamiento paralelo" no revela que tipo o clase de sistema se está utilizando, ni aquello que se está ejecutando en paralelo. Es interesante por tanto considerar algún método de clasificación que nos permita clarificar los términos expuestos anteriormente. A este fin han aparecido multitud de clasificaciones en base a diferentes parámetros:

1.1.- Métricas de control

Dentro de este apartado se encuentra una de las clasificaciones más ámpliamente aceptada debida a Flynn [1,52], que divide a los sistemas paralelos en cuatro categorias, que son las siguientes:

1.1.a.- SISD (Single Instruction Stream, Single Data Stream)

En esta categoria se encuadran la mayoria de computadores serie de los que se dispone hoy en día, en los que las instrucciones y los datos se encuentran almacenados en la memoria principal y una unidad de control y procesamiento se encargan de ejecutar dichas instrucciones secuencialmente. Es frecuente que estos computadores incorporen paralelismo en forma de "pipeline" y/o paralelismo funcional.

1.1.b.- SIMD (Single Instruction Stream, Multiple Data Stream)

En estos sistemas existen múltiples elementos de procesamiento supervisados por la misma unidad de control. Todos los elementos de procesamiento reciben la misma instrucción emitida por la unidad de control y la ejecutan sobre distintos conjuntos de datos. Dentro de este grupo podemos encuadrar arquitecturas como el ILLIAC IV, ICL DAP o el Loral Procesador Paralelo Masivo..

1.1.c.- MISD (Multiple Instruction Stream, Single Data Stream)

En esta organización existen n unidades de procesamiento, cada una de ellas recibe un conjunto distinto de instrucciones que se ejecutan sobre el mismo flujo de datos. No existe ninguna materialización real de esta categoria.

1.1.d.- MIMD (Multiple Instruction Stream, Multiple Data Stream)

En estos sistemas cada elemento de procesamiento, ejecuta una parte de programa de forma autónoma. Las arquitecturas que se enmarcan dentro de esta categoría son muy diversas, por ejemplo, IBM 370/168 MP, Univac 1100/80, Cray-X MP, Alliant, Encore, BNN, Supernode, Cosmic Cube, Ametek Series 2010, etc.

En general podemos decir que los sistemas SIMD, son diseñados en la mayoria de los casos para ser aplicados en problemas muy concretos, con lo que no suelen ser aplicables a cualquier tipo de problema. Son sistemas muy apropiados cuando hay que realizar muchas veces la misma operación sobre un conjunto amplio de datos.

Por el contrario, los sistemas, MIMD, ofrecen una mayor flexibilidad, y por lo tanto, son aplicables a un conjunto de problemas mucho más amplio, aunque para aplicaciones concretas para las que se ha diseñado un SIMD, éste último puede ofrecer un rendimiento mayor que el sistema MIMD. Podrían entenderse los MIMD como sistemas de propósito general.

1.1.2.- Máximo número de bits que pueden ser procesados en paralelo

Una clasificación en este sentido es debida a Feng [2], donde los distintos sistemas son clasificados según el número de bits que pueden procesarse en paralelo. Así por ejemplo, el multiprocesador C.mmp que consta de 16 procesadores de una longitud de palabra de 16 bits, la clasificación de Feng, le asignaría un par (16,16), ya que el sistema puede procesar simultaneamente 16*16 bits. Esta clasificación es raramente utilizada en la literatura profesional.

1.1.3.- Organización del sistema de memoria

Dependiendo de la ubicación del subsistema de memoria y su accesibilidad por parte de los elementos de procesamiento, se distinguen dos tipos básicos de organizaciones [20].:

1.1.3.a.- Sistemas paralelos fuertemente acoplados

Donde existe una memoria global que es compartida y accesible por todos los procesadores. Normalmente la forma de realizar la comunicación entre procesadores es a través de variables compartidas.

1.1.3.b.- Sistemas paralelos débilmente acoplados

La memoria en este tipo de sistemas está distribuida en los procesadores y cada elemento de memoria es local a cada procesador, de tal manera que no hay accesibilidad por parte de un procesador a elementos de memoria no locales. Por tanto la sincronización en este tipo de sistemas tiene lugar a través de paso de mensajes.

La definición de sistema multiprocesador que nosotros utilizaremos en este trabajo es debida a Enslow [23], definición por otro lado ampliamente aceptada.

Un multiprocesador de acuerdo con Enslow, debe satisfacer las siguientes cuatro propiedades:

a.- Debe contener dos o más procesadores de aproximadamente capacidades comparables.

b.- Todos los procesadores comparten el acceso a una memoria común. Esto no excluye la existencia de memorias locales en todos o algunos procesadores.

c.- Todos los procesadores comparten el acceso a canales de entrada / salida, unidades de control y dispositivos. Lo que no excluye la existencia de interfaces y dispositivos locales de entrada / salida para algunos procesadores.

d.- El sistema es controlado por un único sistema operativo.

De acuerdo a la definición de Enslow, un multiprocesador es algunas veces presentado como un sistema fuertemente acoplado, siendo los sistemas débilmente acoplados nombrados a menudo como redes de computadores, o sistemas distribuidos; aunque no hay una terminología exacta para definir los mismos.

El principio del procesamiento paralelo es simple; si un procesador puede ejecutar un millón de instrucciones por segundo, entonces cien de ellos serán capaces de ejecutar cien millones de instrucciones en la misma cantidad de tiempo. No obstante cuando el paralelismo de la aplicación no está bien determinado, la asignación de tareas a procesadores mal realizada, o la ejecución de las tareas en cada procesador mal ordenada, el tiempo de ejecución final de la aplicación puede ser mayor para el sistema multiprocesador que para un entorno monoprocesador.

1.2.- Factores que influyen en el cómputo paralelo

La tendencia significativa hacia el concepto de procesamiento paralelo ha introducido nuevos grados de libertad desde el punto de vista del diseño de arquitecturas y de algorítmos, pero precisamente muchas de estas decisiones que no se presentan en el procesamiento convencional serie, van a influir de manera directa en la eficiencia del sistema paralelo. Los factores más significativos que van a caracterizar un sistema de cómputo paralelo podríamos resumirlos en:

1.2.a.- Arquitectura

1.2.b.- Características de los algoritmos

1.2.c.- Lenguajes paralelos

1.2.d.- Políticas de asignación de tareas a procesadores

1.2.a.- Arquitectura

Un mito extendido a la hora de programar algoritmos, es que el programador no necesita conocer el hardware donde se ejecutarán sus programas. Como todos los mitos, contiene una parte de verdad. Pero no obstante, cuando las restricciones de tiempo impuestas por la aplicación son importantes, el programador tiene que utilizar sus conocimientos de paginación, tamaño de memoria cache, etc para sintonizar su programa.

La situación se agrava cuando se trabaja en entornos de multiprocesamiento debido a la gran variedad de arquitecturas existentes. Mas que intentar exponer una taxonomía para la clasificación de arquitecturas, intentaremos mostrar algunos aspectos que van a afectar de manera decisiva al estilo de codificación. Intentaremos englobar a todos los sistemas en una de las tres clases que presentamos a continuación [24]:

- Arquitecturas de memoria compartida, 1.2.a.1

- Arquitecturas de paso de mensajes, 1.2.a.2

- Arquitecturas híbridas, 1.2.a.3

1.2.a.1.- Una máquina con memoria compartida tiene una única memoria global accesible a todos los procesadores, o bien el sistema operativo se encarga de mostrar la apariencia de tener un único espacio de direcciones de memoria. Una propiedad que va a caracterizar a este tipo de arquitecturas es que los tiempos de acceso a los datos van a ser independientes de el procesador que haga la petición. En este tipo de arquitecturas la red de interconexión más común entre los procesadores, el sistema de memoria, y los dispositivos de entrada salida es uno o más buses de alta velocidad

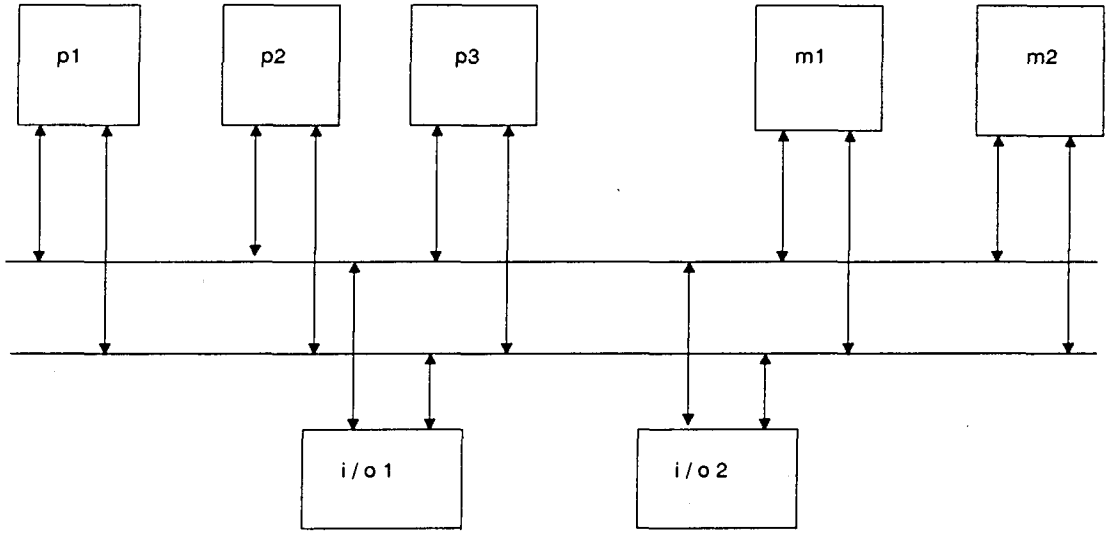


Figura 1.1

figura 1.1; donde el ancho de banda del bus común va a limitar el número de procesadores que pueden ser soportados sin contención. Los diseñadores de este tipo de arquitecturas suelen mejorar el rendimiento de un sistema basado en buses añadiendo caches a cada procesador. Las memorias caches, pueden responder a muchas referencias a memoria,a nivel local, reduciendo la contención en el sistema de bus. No obstante la consistencia de la cache puede ser un problema ahora, ya que cada procesador puede modificar a cada elemento de manera independiente. Una solución al problema es añadir hardware que sincronice el contenido de las caches, con el consiguiente aumento de costo para cada procesador en el sistema.

El FX/8 construido por Alliant Computer Systems Corp. of Litleton, Mass., es un típico sistema multiprocesador basado en bus, tiene ocho procesadores y puede ejecutar aproximadamente 35 millones de instrucciones por segundo (MIPS). El FX/80 tiene un ancho de banda de 188 megabytes por segundo; el sistema también incorpora un mecanismo hardware para la cache local de cada procesador.

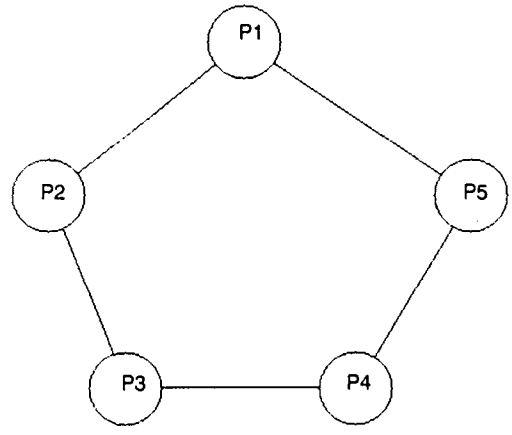


Figura 1.2

1.2.a.2.- Los sistemas basados en espacios independientes de direcciones de memoria, son configurados con una memoria local para cada procesador, pero ninguna de ellas es globalmente accesible. La única manera para que una aplicación pueda compartir datos entre procesadores es que el programador explicitamente codifique comandos para mover datos de un procesador a otro.

Un gran número de estructuras de interconexión han sido usados. La más simple es conectar las máquinas en anillo capacitando que cada procesador pueda hablar con con sus dos vecinos cercanos figura 1.2. En tal situación, una máquina toma un tiempo proporcional al número de procesadores para enviar datos a cada procesador.

Una máquina con más cableado es una malla en la cual los procesadores son conectados en una rejilla bidimensional figura 1.3, donde cada procesador habla a sus vecinos cercanos del norte, sur, este y oeste. Mallas con cableados más densos entre procesadores, generan interconexiones en hipercubos figura 1.4, hoy por hoy, uno de los diseños más populares. En este tipo de arquitecturas, cada nodo del hipercubo tiene su propio elemento de procesamiento y una memoria local; cada elemento de procesamiento tiene n caminos de comunicación directa con otros nodos del sistema a través de los arcos del cubo que son conectados a cada procesador. El número total

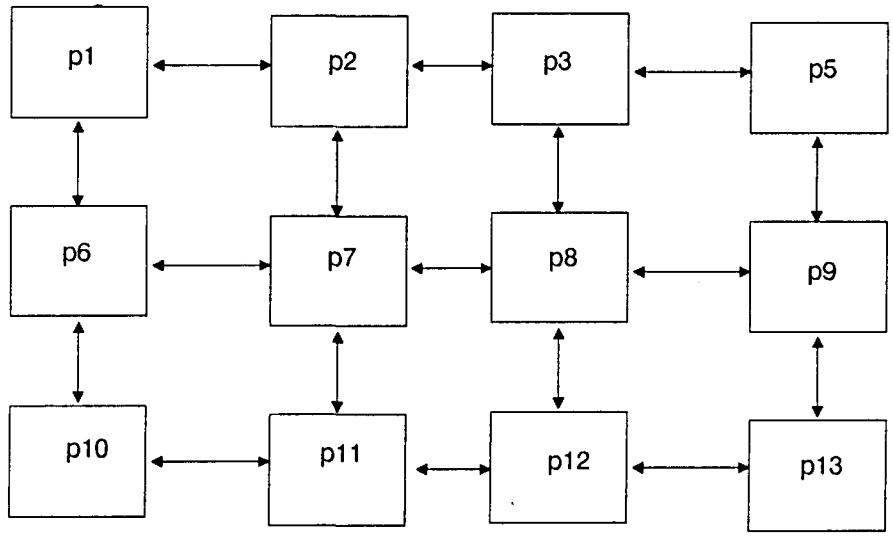


Figura 1.3

de dimensiones es llamada la dimensión del hipercubo. Un cubo de dos dimensiones tiene cuatro procesadores, cada uno conectado a otros dos; un cubo 3D tiene ocho procesadores, un 4D tiene 16 procesadores, cada uno tiene cuatro conexiones y así sucesivamente. Los hipercubos pueden ser expandidos para un gran número de procesadores, ya que carecen de bus global para limitar el total. Ametek Inc. y Floating

Point Systems Inc. todos ellos constructores de sistemas basados en hipercubos realizan sistemas entre 16 y 1024 procesadores. La principal ventaja de esta configuración es que la distancia más larga entre procesadores (diametro) es proporcional a log2 p, donde p es el número de nodos del hipercubo; lo que significa que dos nodos podrán comunicarse con relativa rapidez en sistemas con un elevado número de procesadores.

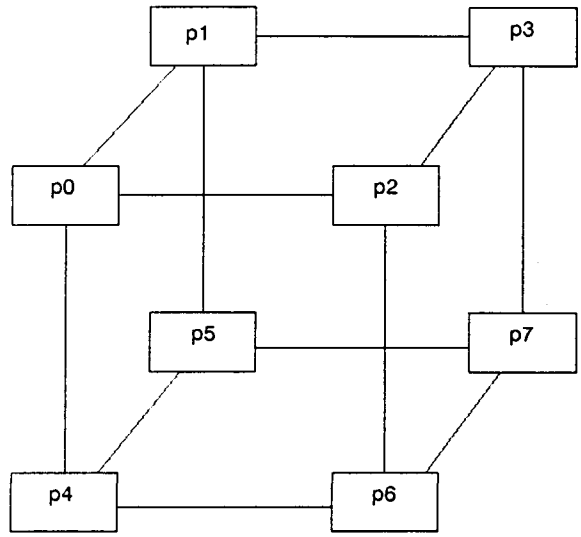


Figura 1.4

Otro tipo de red de interconexión usual entre procesadores la forman las basadas en redes de conmutación multietapa figura 1.5, que conectan los procesadores y módulos de memoria a través de una red especializada de conmutación. Como en las arquitecturas basadas en buses, la totalidad de la memoria puede ser accedida por cualquier procesador, pero este tipo de redes puede crecer hasta al menos 200 procesadores, muchos más que los sistemas basados en bus. En estos sistemas muchos procesadores pueden simultaneamente acceder a multiples memorias ya que existen multiples caminos a través de la red.

Un clásico ejemplo de estas arquitecturas es el multiprocesador Butterfly de BBN Advanced Computers. Este puede ser expandido hasta 256 procesadores teniendo una potencia de cálculo de 250 MIPS. El diseño de Butterfly enpareja a cada

procesador con un módulo de memorias así cada procesador puede leer o escribir variables almacenadas en su memoria local sin usar la red conmutación.

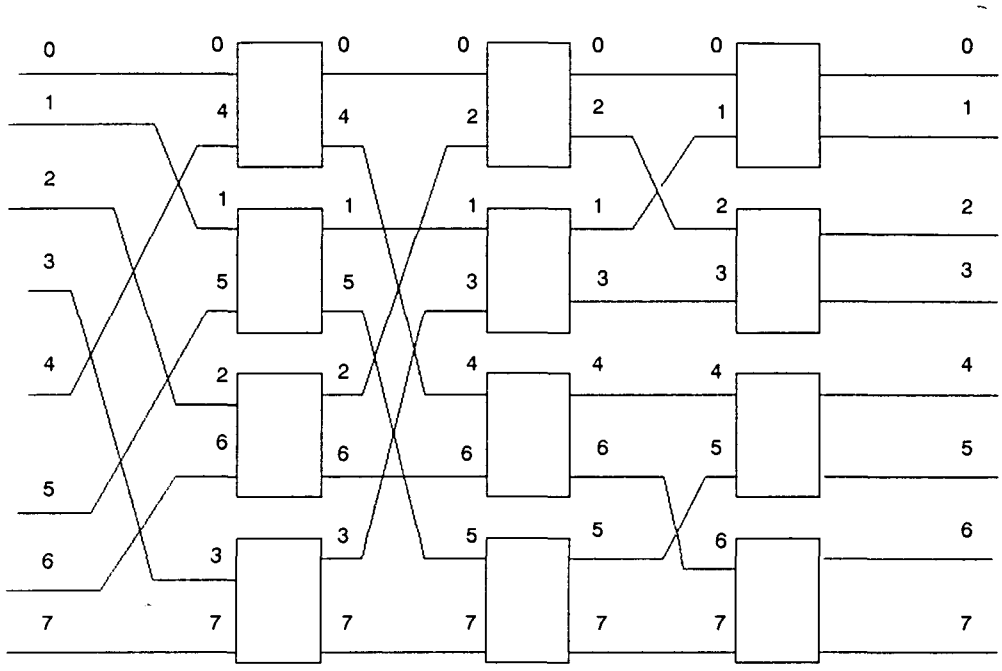


Figura 1.5

1.2.a.3.- Los sistemas híbridos tienen algunas propiedades de los sistemas de memoria compartida y otras típicas de los sistemas basados en paso de mensajes. Toda la memoria es local a un procesador dado, pero el sistema operativo hace que la máquina parezca un sistema de memoria global. Así los programas se escriben como para un sistema de memoria compartida. No obstante los datos son pasados por el sistema que maneja los mensajes. El IBM RP3, BBN Butterfly, y Cedar son ejemplos de máquinas híbridas.

1.2.b.- Características de los algoritmos

Los algoritmos paralelos pueden ser caracterizados en base a un gran número de atributos [25]. Idealmente, un conjunto de características definirán los algoritmos; granularidad del módulo, tipo de paralelismo, granularidad de los datos, etc, y un conjunto de características describirán las distintas arquitecturas [20,26], con una única relación o mapeo de una en otra. Experiencias realizadas por [26,27]muestran

que las relaciones entre los algoritmos paralelos y las arquitecturas paralelas es demasiado compleja como para estar de acuerdo con el modelo anterior.

Algunas de las características más releventes que podemos destacar de los algoritmos las describimos brevemente a continuación [28]:

1.2.b.1.- Naturaleza del paralelismo. Hace referencia fundamentalmente al tipo de paralelismo que es usado y la forma en la que el algoritmo y los datos pueden ser descompuestos.

a.- paralelismo funcional, realizado mediante unidades independientes para llevar a cabo simultaneamente diferentes operaciones en diversas unidades aritmético-lógicas sobre distintos datos.

b.- paralelismo de tipo "pipeline", donde las operaciones matemáticas y de control de la unidad de procesamiento se dividen en "pasos", encadenados que operan de forma concurrente. Este paralelismo, "vertical" permite en teoría tener en ejecución tantas instrucciones como pasos posee el "pipeline". En una máquina con un "pipe" de n estados, puede haber n instrucciones en ejecución simultaneamente.

c.- paralelismo de segmentación encauzada, denominado también paralelismo en forma de "baterias" ("arrays") de procesadores idénticos, todos funcionando bajo la misma unidad de control, y realizando la misma operación sobre distintos datos.

d.- paralelismo de tipo multiprocesador, donde unidades de cómputo independientes, cada una de ellas con su propio conjunto de instrucciones, cooperan en la resolución de una aplicación.

Las dos primeras posibilidades están ámpliamente extendidas y han sido incorporadas por muchos sistemas monoprocesador. Se trata de un tipo de paralelismo más general que no depende del problema a resolver sino que se centra en el funcionamiento de los procesadores. Por el contrario, las dos últimas, se aplican

en sistemas multiprocesador, donde cada procesador individual puede incorporar a su vez el paralelismo de "pipeline" y/o el funcional a nivel interno.

1.2.b.2.- La granularidad de los datos [39], que tiene que ver con el tamaño de los item de datos procesados como unidad fundamental, y que van a influir de manera importante en los requerimientos de comunicación, asignación de los datos, necesidades de memoria, etc.

1.2.b.3.-Granularidad del módulo [12,37,38,53,59]. Cuantifica la cantidad de procesamiento asociado a cada porción de código secuencial, cuando el tamaño de la sección de código asociado a cada tarea de la aplicación es equivalente a un pequeño conjunto de instrucciones del procesador se habla de tarea de granularidad fina, "fine grain" con lo que se consigue aprovechar al máximo el paralelismo de la aplicación, pero por el contrario, el número de comunicaciones se incrementa de manera desmesurada y pasan a ser el cuello de botella del sistema; de granularidad gruesa, "large grain" cuando el volumen de cómputo asociado a cada tarea es muy importante, con lo que se reducen las comunicaciones, pero se desaprovecha parte del paralelismo del programa. También podemos considerar las posibilidades intermedias "coarse-grain". La situación ideal sería alcanzar un compromiso entre las dos primeras posibilidades que aprovechase al máximo el paralelismo de la aplicación, pero no saturase el sistema de comunicaciones.

1.2.b.4.- Grado de paralelismo [12,39,58]. Intimamente ligado con la granularidad de los datos y la granularidad del módulo, fundamentalmente es la parte que más directamente incide en el rendimiento final del sistema. El grado de paralelismo está relacionado en la práctica con el modo de operación (paralelismo masivo, raramente estará asociado con la ejecución MIMD) y con la organización de la memoria (con paralelismo masivo, una organización global de la memoria puede mostrar un problema importante de contención en el acceso a la misma).

1.2.b.5.- La uniformidad de las operaciones [47,48]. Generalmente va asociada al paralelismo de datos, si el conjunto de operaciones a realizar no son uniformes, se

deberá elegir procesamiento de tipo MIMD y deberemos también establecer políticas de planificación a la hora de balancear la carga a los procesadores del sistema.

1.2.b.6.- Requerimientos de sincronización [55,56]. Además de los requerimientos de sincronización debidos a la granularidad del módulo, la consideración de las restricciones de precedencias van implícitas en la caracterización de los elementos de sincronización.

1.2.b.7.- Tipo de dependencia de datos [28]. Las dependencias de datos juegan un papel privilegiado a la hora de dictar los caminos de comunicación y las características de comunicación.

1.2.b.8.- Operaciones fundamentales [47,48]. Las operaciones a realizar por el algoritmo dictarán las capacidades que necesitarán las unidades de procesamiento.

1.2.b.9.- Estructuras de los datos [28]. Muchos algoritmos pueden ser caracterizados por tener una estructura de datos natural. La habilidad de una determinada arquitectura a la hora de explotar posibles irregularidades en las estructuras de datos, será un factor clave a estudiar con objeto de sintonizar el binomio algoritmo/arquitectura.

1.2.c.- Lenguajes paralelos

El paralelismo puede ser expresado de una gran variedad de formas. Un factor importante a considerar es la unidad básica de paralelismo del lenguaje. En un lenguaje secuencial, la unidad de paralelismo es el mismo programa. En un lenguaje orientado hacia el cómputo distribuido la unidad de paralelismo puede ser un proceso, una sentencia, un objeto, una expresión o una claúsula (en los lenguajes lógicos). A continuación se describen algunas de las características más relevantes de cada uno de ellos dependiendo de la unidad de paralelismo que se representa. Los tipos más representativos de lenguages paralelos existentes en función de la unidad de paralelismo que se representa podemos dividirlos en:

1.- Lenguajes Orientados a Procesos

2.- Lenguajes Orientados a Objetos

3.- Sentencias paralelas

4.- Paralelismo funcional

5.- Paralelismo AND/OR

1.1.2.c.1.- Lenguajes Orientados a Procesos

En muchos lenguajes de tipo procedural para computación distribuida, el paralelismo se realiza basado en la noción de proceso. Diferentes lenguajes tienen distintas definiciones de este concepto, pero en general un proceso es una actividad que ejecuta código secuencialmente y que tiene su propio estado y datos, donde los procesos (o tipos de procesos) son declarados como procedimientos (y tipos de procedimientos).

Los procesos pueden ser creados implícitamente por su declaración , o explicitamente por medio de alguna construcción explícita "create". Mediante la creación implícita, en primer lugar se declara un tipo de proceso y depués se crean los procesos declarando procesos mediante la declaración de variables de ese tipo. En algunos lenguajes basados en la creación implícita de procesos, el número total de procesos es fijado en tiempo de compilación. Esto facilita la tarea de mapear los procesos a procesadores, pero impone restricciones en cuanto al tipo de aplicaciones que pueden ser representadas por el lenguaje, ya que requiere que el número de procesos deba ser conocido a priori.

El hecho de disponer de construcciones explícitas para crear procesos, permite una mayor flexibilidad que la creación implícita de los mismos. Algunos ejemplos

representativos de lenguajes basados en procesos lo constituyen, Ada, C-Concurrente, Linda, Nil [58,59,60], etc.

1.2.c.2.- Lenguajes Orientados a Objetos

En general, un objeto es una unidad autocontenida que encapsula tanto las estructuras de datos como el código que aloja, y que interactua con el exterior a través del paso de mensajes. Los datos contenidos en los objetos son visibles solamente desde dentro de los objetos. El comportamiento de un objeto viene determinado por la clase a la que pertenece, la cual comprende una serie de operaciones que pueden ser invocadas mediante el envio de un mensaje al objeto. Existen básicamente dos opiniones diferentes en relación a lo que se debe o no considerar como objeto, la visión que proporciona Smalltalk-80, donde cualquier cosa practicamente se considera un objeto [31], y una segunda visión tachada de menos pura en el sentido que permite al programador decidir cuales serán los objetos [32]. Algunos lenguajes representativos lo constituyen, Esmeralda, Smalltalk Concurrente, etc.

1.2.c.3.- Sentencias paralelas

Otra forma de expresar el paralelismo es agrupar un conjunto de sentencias que van a ser ejecutadas en paralelo. Occam [33], permite que varias sentencias puedan ser ejecutadas o bien en serie, mediante expresiones de la forma:

SEQ S1,S2 (Ejecución secuencial S1 y depués S2)

o bien en paralelo, mediante la expresión:

PAR S1,S2 (Ejecución "simultanea" S1 y S2)

Este método es fácil de usar y de entender, no obstante, este mecanismo proporciona poco soporte para estructurar programas paralelos de gran tamaño. Con objeto de poder representar sentencias de tipo iterativo, como en los lenguajes

clásicos se dispone también de sentencias de tipo FOR, IF, CASE, etc , (a nivel de funcionamiento) adaptadas para expresar el tipo de paralelismo deseado.

1.2.c.4.- Paralelismo funcional

En este tipo de lenguajes, las funciones se comportan como si de funciones matemáticas se tratara; estas, calculan un resultado que dependen exclusivamente de sus datos de entrada. Tales funciones no tienen efectos laterales, en contraste con los lenguajes de tipo procedural (imperativos) donde se permite que sus resultados afecten a valores de variables globales. Una característica importante de los lenguajes funcionales es que las funciones no importa el orden en que son ejecutadas. Por ejemplo h(f(3,4), g(8)) es irrelevante si es la función g o f la que se ejecuta primero; por tanto ambas se pueden realizar en paralelo. En principio cualquier función puede realizarse en paralelo, la única condición es que una función que utiliza resultados obtenidos de otra función, debe esperar hasta que estos están disponibles. Por ejemplo la función h esperará hasta que la función f y g hayan calculado sus resultados. El tipo de paralelismo implícito es realizado a nivel de "fine grain", y es útil para arquitecturas que manejan este tipo de paralelismo (máquina de flujo de datos). Algunos lenguajes que utilizan este principio lo constituyen Id y VAL [34]. No obstante existe un gran número de problemas que deben ser resueltos, para aplicar esta metodología al cómputo distribuido en general [35]. ParAlf1 y FX-7, constituyen algunos ejemplos de este tipo de lenguajes.

1.2.c.5.- Paralelismo AND/OR

La programación lógica ofrece muchas oportunidades para el paralelismo [36]. Los programas lógicos pueden ser leidos declarativamente como los de tipo procedural. A continuación se codifican dos clausulas declarativas para el predicado A:

(1) A:- B, C, D.

(2) A:- E,F.

La lectura de las clausulas es; si B, C y D son verdaderos, entonces A es cierto (clausula (1)), y si E y F son ciertos entonces A es cierto (clausula (2)). Proceduralmente, las clausulas pueden ser interpretadas como: para probar el teorema A, se deben probar o bien los subteoremas B, C y D o E y F. Donde el paralelismo a la hora de probar los teoremas puede darse:

-Las dos cláusulas para A, pueden estar trabajando en paralelo, hasta que una de ellas se cumple, o ambas fallan.

-Para cada de las clausulas, los subteoremas pueden estar ejecutándose en paralelo, hasta que todos se cumplen o alguno de ellos falla.

La primera clase descrita de paralelismo se denomina paralelismo OR; la segunda se denomina paralelismo AND. Algunos ejemplos representativos de este tipo de lenguajes lo constituyen Prolog Concurrente y Parlog [61].

1.2d.- Políticas de planificación y partición de programas paralelos

La ejecución de programas paralelos en sistemas multiprocesador implica la solución de dos problemas fundamentales:

1.- Particionar el programa en módulos concurrentes que permita obtener el mejor tiempo de ejecución posible para la aplicación propuesta.

2.- Realizar la asignación de tareas de forma óptima. Asumiendo que disponemos de la granularidad óptima de los módulos concurrentes de un programa, cómo asignar dichos módulos a los distintos procesadores del sistema especificando el orden de ejecución de las tareas asignadas a cada procesador con objeto de minimizar el tiempo de ejecución del programa.

El punto clave a la hora de estudiar la granularidad óptima de la aplicación reside fundamentalmente en sintonizar adecuadamente el paralelismo potencial que podemos encontrar en la aplicación y el "overhead" que se debe pagar a la hora de explotar el paralelismo definido.

Cuando ejecutamos tareas en paralelo, claramente debe existir un compromiso entre el paralelismo generado para la aplicación y el "overhead" que se genera debido a las comunicaciones. Estas, obviamente, serán minimizadas cuando todas las tareas sean ejecutadas en el mismo procesador (mínimo paralelismo), o bien cuando la granularidad de las mismas es grande (large grain), en cuyo caso el nivel de paralelimo es reducido. Por otro lado, cuando la granularidad de las tareas es pequeña (fine grain) con respecto a las comunicaciones generadas, el nivel de comunicaciones puede afectar negativamente el tiempo final de ejecución de la aplicación. Este problema, de simultaneamente maximizar el paralelismo de la aplicación y minimizar las comunicaciones, es un problema notorio que cae dentro de los definidos como NP-completos [9,10]. Por tanto, deberá buscarse un punto de equilibrio o solución de compromiso entre ambos objetivos.

Fundamentalmente existen dos técnicas a la hora de particionar programas [37]:

a.- "Top-down". Solución que comienza considerando el programa como un único módulo y continua, aplicando determinadas reglas que descomponen el problema en tareas más pequeñas mediante la aplicación de técnicas recursivas.

b.- "Botton-up". En esta solución se parte del nivel más bajo posible de granularidad de los nodos que componen la aplicación e intenta agrupar tareas para formar otras más grandes.

Se ha demostrado que la composición "botton-up" es superior a la descomposición "top-down" [37,38], ya que durante la fase de composición disponemos de información acerca de la estructura global del programa así como de sus componentes básicos.

En algunas implementaciones el nivel de granularidad es determinado por las construcciones del lenguaje, expresiones compuestas, rutinas de usuario; implicando que el estilo de programación afecte de manera notable el rendimiento de la aplicación en el sistema multiprocesador [39]. Un programador normalmente no conoce los detalles de cómo influye la granularidad de sus tareas y su relación con el costo de comunicación; es pues de esperar que esta relación no será óptima. Normalmente una mala partición de la aplicación, independientemente del algoritmo de "scheduling" generará un resultado final bastante alejado de los óptimos de ejecución que podrían obtenerse.

Para aplicaciones donde los tiempos de ejecución de las distintas tareas pueden ser predecibles, tanto el problema de la partición como el problema de la planificación deben realizarse en tiempo de compilación [38,39], lo que permite por tanto realizar un análisis estático de la aplicación con objeto de maximizar el grado de paralelismo y minimizar las comunicaciones atendiendo a los parámetros que guiarán el proceso de partición: número de procesadores, distancias entre ellos, "overhead" de comunicaciones etc.

1.2.e.- El problema del "Scheduling"

Con objeto de obtener el mejor rendimiento del procesaminto paralelo, uno de los indices de prestaciones a optimizar, lo constituye la minimización en el tiempo de ejecución de las aplicaciones. En base a la consecución de este fin, deberemos encontrar algoritmos de planificación eficientes que nos permitan encontrar tanto asignaciones de tareas a procesadores, como determinar el orden de ejecución de las tareas asignadas a cada procesador.

Este es un problema que ha sido ampliamente estudiado [3,4,7,10,11,12,13,14], extremadamente dificil de resolver, y generalmente intratable. Es bien conocido que la formulación general del problema cae dentro de la clase de problemas considerados NP-completos [9,10]. La dificultad de encontrar la solución, depende de factores tales como la inclusión o no de la propiedad de apropiatividad, el número de procesadores,

y de ciertos atributos de las tareas, tales como la topología del grafo, y la uniformidad de los tiempos de ejecución de las tareas.

1.2.e.1.-Factores de clasificación

La forma de caracterizar los distintos tipos de "schedules" es una tarea compleja debido a la cantidad de factores que pueden influir a la hora de establecer una clasificación. A continuación identificaremos estos factores con objeto de establecer cuales son las características más relevantes en orden a establecer una taxonomía que nos permita discernir distintas propiedades de los mismos.

1.2.e.1.a.- Número de procesadores

Tradicionalmente los entornos monoprocesador han dominado las instalaciones de sistemas de cálculo. No obstante la inclusión de multiples procesadores se ve como la alternativa más clara de obtener mayores prestaciones ahora que la tecnología VLSI está llegando a su límite teórico.

Mucho se ha escrito comentando los límites en cuanto a la ganancia máxima en velocidad que se puede obtener mediante la utilización de sistemas multiprocesadores para resolver aplicaciones específicas [21,22]. Una de las leyes más conocidas es la formulada por Amdahl [21], la cual afirma que el límite máximo en velocidad respecto al algoritmo secuencial que puede ser alcanzado en un sistema multiprocesador viene limitada como la fracción del algoritmo que no puede ser paralelizada.

Si definimos la velocidad de "p" procesadores (speed-up) como:

$Sp = T1/Tp$

donde T1 = tiempo de ejecución serie para el mejor algoritmo en un procesador, Tp = tiempo de ejecución para algoritmos paralelos que utilizan "p"

procesadores y Fs = fracción de algoritmo secuencial que debe ser ejecutado secuencialmente (no paralelizable). Nosotros tenemos entonces que:

$$Tp = Fs\,T1 + ((1 - Fs)\,T1)/p$$

y la velocidad respecto al sistema monoprocesador

$$Sp = p/(1 + (p - 1)\,Fs$$

Por tanto, hay un límite para la velocidad alcanzable independientemente del número de procesadores usados así:

$$\lim p \text{ ---- infinito } [Sp] = \lim p \text{ ---- infinito } [1/(1/p + (1 - 1/p)\,Fs)] = 1/Fs$$

Así, por ejemplo, si el 5% de un algoritmo no puede ser paralelizado, la máxima velocidad alcanzable será de 20 independientemente del número de procesadores disponibles.

En contra de este límite aparente a la utilidad potencial del procesamiento paralelo, se ha observado que gran parte de los problemas de ingeniería y de cálculo científico, la fracción de Amdahl Fs depende del tamaño del problema n. Lo que significa que Fs es una función de n, Fs (n). Definiendo un algoritmo paralelo eficiente como aquel que lim n---- infinito Fs (n) = 0• .

Teniendo en cuenta lo anterior la velocidad de un algoritmo paralelo eficiente tiene como límite el definido como la velocidad lineal:

$$\lim \text{ ---- infinito } [Sp] = p/1 + (p - 1)\,Fs\,(n) = p$$

Resultados experimentales recientes [62], muestran una notable concordancia con los valores predichos.

1.2.e.1.b.- Duración de las tareas

Un factor determinante a la hora de estudiar algoritmos de "scheduling" será si todos los tiempos de las tareas tienen el mismo valor o no; este hecho, junto con otros factores (número de procesadores, tipo de precedencias), será determinante a la hora de encontrar o no algoritmos que proporcionen soluciones óptimas al problema planteado [10]. En el apartado 1.2.e.2 se muestra bajo qué condiciones el problema del "scheduling" es tratable o intratable.

1.2.e.1.c.- Estructura del grafo de precedencia

Los nodos de un grafo puden estar relacionados unos con otros de diferentes maneras. Por ejemplo es posible que todas las tareas sean independientes unas de otras. En esta situación se dice que no hay precedencia u ordenación parcial entre las tareas, encontrandonos con grafos llamados de comunicación. Otra posiblidad es que las tareas estén relacionadas mediante precedencias que deberemos respetar para la correcta ejecución del grafo.

1.2.e.1.d.- Interrumpibilidad de una tarea

Si permitimos la interrupción (y posterior reasumpción) de una tarea antes de su finalización, hablaremos de "schedule" apropiativo ("premmptive schedule"). Si no permitimos interrupciones antes de la finalización de la tarea hablaremos de "nonpremmptive" o "schedule" básico. En general los "schedules" de tipo apropiativo proporcionan mejores resultados que los generados mediante la aplicación de disciplinas no apropiativas, aunque en los primeros hay un claro "overhead" por el hecho de de la conmutación de tareas. Este "Overhead" puede ser aceptable si ocurre de forma infrecuente [11].

1.2.e.1.e.- Permitir tiempos muertos en los procesadores

Tomando como referencia de medida el tiempo de ejecución del grafo podemos encontrarnos situaciones en las que sea mejor de cara a minimizar el tiempo final de ejecución, permitir que uno o varios procesadores no ejecuten ninguna tarea aunque

tengan asignadas para su ejecución. No obstante, debido a la complejidad que desde el punto de vista del algoritmo de asignación implica comprobar estas situaciones, se prefiere no introducir tiempos muertos en los "schedules", de tal forma que siempre que hay procesador libre y tarea lista está deberá ser ejecutada.

1.1.2.e.1.f.- Presencia o ausencia de restricciones temporales

Normalmente los "schedules" realizan asignaciones globales en base a minimizar alguna función de costo, no obstante hay veces que un determinado conjunto de tareas del grafo necesitan ser ejecutadas en unos tiempos acotados, cuando sucede esto hablamos de "hard dead-line" [11].

1.2.e.1.g.- Tipo de procesadores

Con este apartado queremos poner de manifiesto las características de las unidades de procesamiento, hablaremos de procesadores homogéneos cuando los procesadores tengan las mismas características, y heterogéneos cuando no las tengan. Notar la importancia de este apartado en lo que hace referencia a la indiferencia de los "schedules", en sistemas homogeneos, a la hora de asignar tareas a cualquier procesador libre en ese momento, y la posible utilización de cualquiera de ellos cuando se observa un mal funcionamiento en un determinado procesador del sistema.

1.2.e.2.- Complejidad del Problema del "Scheduling"

Con objeto de mostrar la influencia de los parámetros anteriormente expuestos, a continuación expondremos una clasificación del problema del "scheduling" desde el punto de vista de la complejidad temporal, dependiendo de los valores que pueden tomar los parámetros anteriormente descritos.

La tabla 1.1 [10], muestra la complejidad de los algoritmos de "scheduling" para diferentes planteamientos del problema, en función de los distintos valores asociados a las diferentes magnitudes que se contemplan.

Problema Numero	Numero de procesadores	Tiempo de procesamiento	Precedencias	Complejidad
1	arbitrario	igual	arbol	O(n)
2	fijo = 2	igual	arbitrario	O(n2)
3	arbitrario	igual	arbitrario	NP
4	fijo = 2	1 o 2	arbitrario	NP
5	arbitrario	igual	arbitrario	NP-Compl.

Tabla 1.1

El primer caso fue estudiado por Hu [3] desarrollando un algoritmo eficiente de complejidad O(n), donde los tiempos de procesamiento de las tareas eran para todas iguales, y el grafo de precedencia tenía forma de árbol. El segundo caso, permite relaciones de precedencias arbitrarias, pero limita el número de procesadores a dos. Bajo estas condiciones, Coffman y Graham [4] encontraron un algoritmo de complejidad $O(n^2)$. Salvo estos dos casos, para los demás podemos observar que no podemos encontrar algoritmos de tipo polinomial. Por tanto a la hora de afrontar el problema anterior, la mayoría de los esfuerzos van dirigidos a la búsqueda de heurísticas que permitan encarar el problema, aunque como es obvio, serán políticas subóptimas, que se intentan ajustar lo más posible a las cotas de óptimo que se definan como referencia.

1.2.e.3.- Modelo de representación

Una de las representaciones más utilizada en la literatura del "Scheduling" para representar el conjunto de parámetros que definen el conjunto de tareas en un entorno de cómputo paralelo la constituyen los grafos dirigidos acíclicos (DAG). La

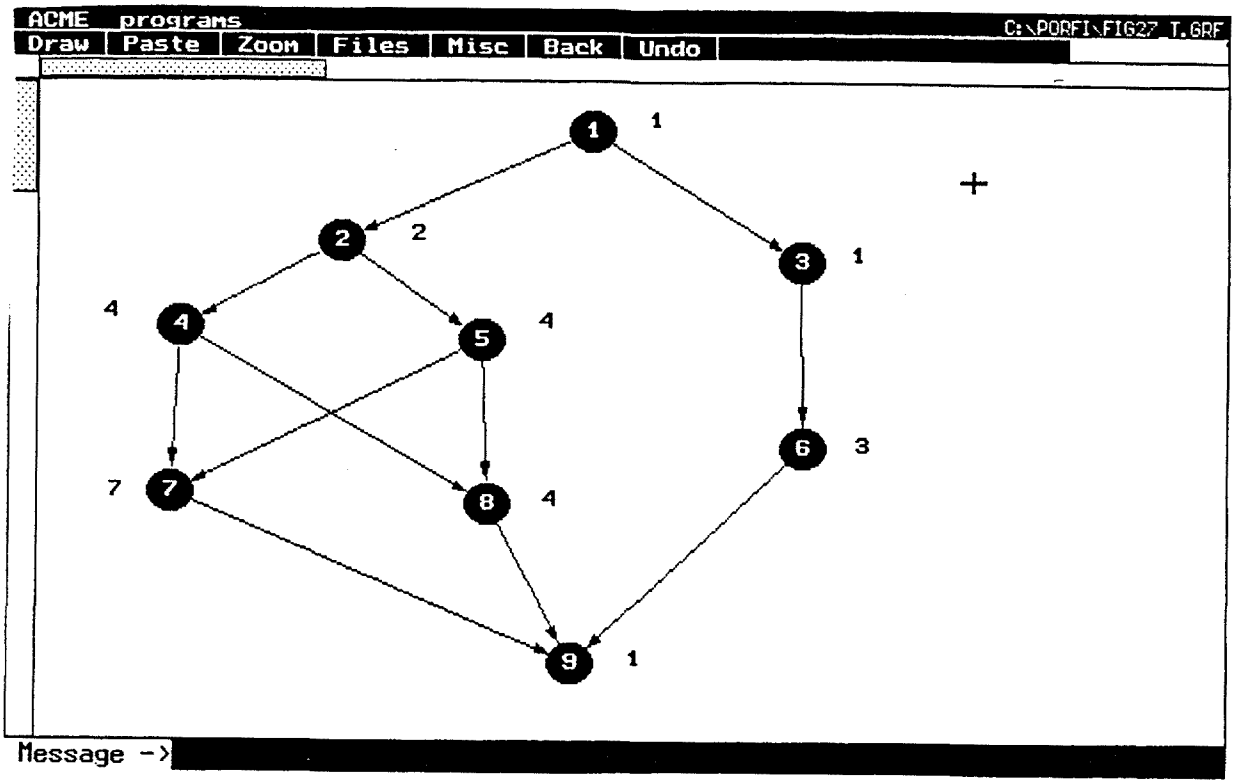


Figura 1.6

figura 1.6, muestra una de las posibles representaciones para trabajos o tareas. Los nodos en estos grafos representan operaciones independientes o partes de un programa que están relacionadas en el tiempo unas con otras, los arcos entre tareas, las relaciones de precedencia entre ellas.

En la figura 1.6, el conjunto de nodos representa el conjunto de tareas T = (t1,t2, ... , tn). Los arcos entre nodos indican las relaciones de precedencia entre las distintas tareas que forman el grafo; de tal forma que una tarea no puede ser ejecutada hasta que hayan finalizado todas sus tareas predecesoras. En la figura 1.6, también se muestra la correspondencia entre el identificador de la tarea y su tiempo de ejecución

asociado, haciendo referencia al volumen de cómputo asociado a cada sección de código.

Observar que en la figura 1.6, no se hace referencia al número de procesadores disponibles para ejecutar el conjunto T de tareas. Por supuesto que el número de procesadores utilizados en la ejecución del grafo T, influye notablemente en el rendimiento del sistema.

Nosotros adoptaremos este modelo de representación basado en DAG, ahora bien, el problema que se suscita hace referencia a la capacidad del modelo para representar programas reales. Debido a la naturaleza intrínseca de los programas de computo, la mayoría de las veces es imposible calcular de forma precisa los tiempos de ejecución que implica la ejecución secuencial del código asociado a cada tarea representada en el DAG, no solamente en lo referente a estructuras de tipo condicional o iterativas, sino también cuando en porciones de tipo serie nos encontramos que el tiempo de ejecución del código asociado a la tarea depende de los datos de entrada. Por tanto deberá tenerse en cuenta que los tiempos de procesamiento de las tareas representadas por este modelo (DAG), son valores estimativos, debido a la dependencia que frente a los datos presentan los volúmenes de procesamiento asociados a las tareas del grafo.

1.2.e.4.- Clasificación jerárquica

El objetivo básico de la clasificación propuesta en este apartado es proporcionar una visión de los distintos tipos de planificadores, con objeto de compararlos y posteriormente enmarcar nuestro trabajo dentro de la misma.

La estructura de la taxonomía se muestra en la figura 1.7 . A continuación se describen algunas de las características más relevantes que han dado origen a esta clasificación debida a Casavant [19].

1.2.e.4.1 Local Versus Global

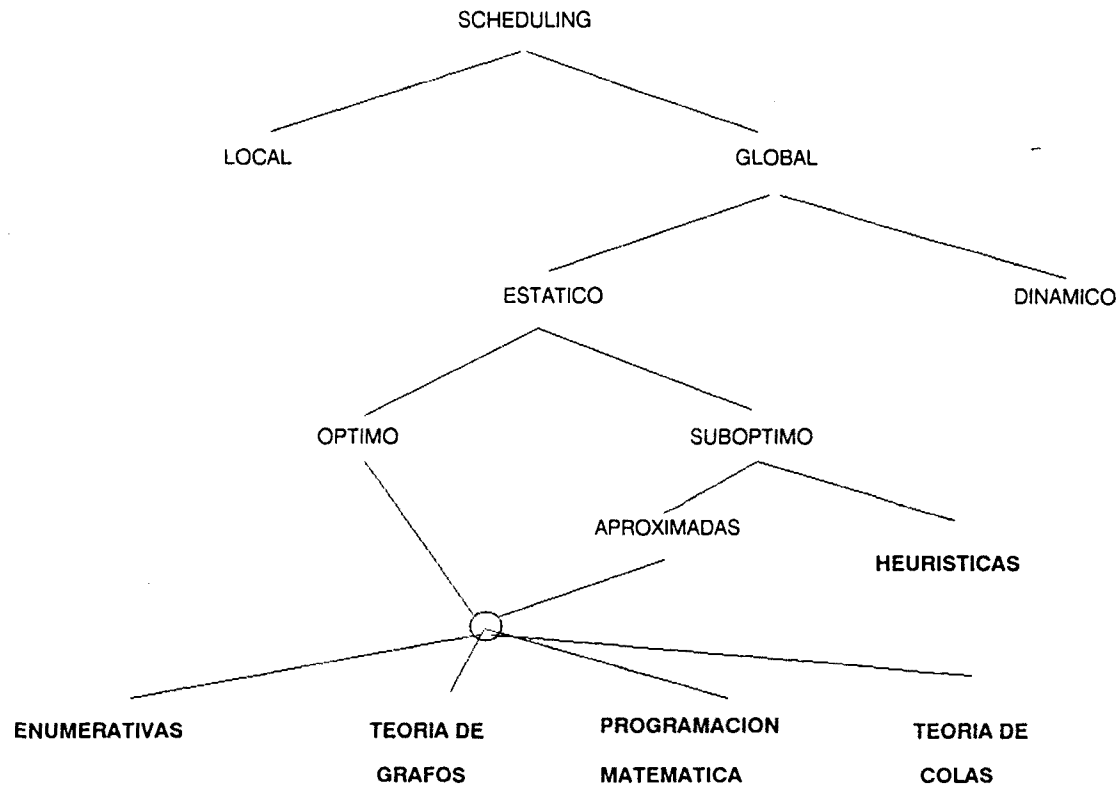


Figura 1.7

A este nivel podemos distinguir entre planificación local y planificación global. La asignación local hace referencia a la asignación de los ciclos de procesador a los procesos en entornos monoprocesador, donde la concurrencia se realiza a nivel de "interleaving" entre las distintas tareas asignadas, o bien a la asignación de trabajos independientes en entornos distribuidos "job shop scheduling" [10], donde el índice a maximar es principalmente el número de trabajos capaz de realizar el sistema por unidad de tiempo. El problema del "sheduling" global es decidir dónde se ejecutará un proceso y cual deberá ser su orden de ejecución en el procesador asignado; en este tipo de "scheduling" las tareas forman parte de un único programa, o algoritmo teniendo como objetivo mayoritariamente ,minimizar el tiempo de ejecución del algoritmo.

1.2.e.4.2 Estático Versus Dinámico

El siguiente nivel en la jerarquía es la elección entre Planificación estática o dinámica. Esta elección fundamentalmente indica cúando se realiza la asignación. En el "scheduling" estático todos los parámetros representativos del algoritmo se conocen en tiempo de compilación, con lo cual se pueden preparar asignaciones "off-line" sin producir ningún "overhead" en tiempo de ejecución, notar que normalmente este conocimiento tan preciso del programa no será posible obtenerlo en tiempo de compilación la mayoría de las veces debido a la existencia de indeterminaciones en las estructuras de ruptura de secuencia, y de sentencias iterativas. Algunos autores definen como planificación determinista este tipo de "scheduling".

El "scheduling" dinámico se realiza en tiempo de ejecución, no necesita conocer de forma tan precisa los parámetros del algoritmo, y es capaz de realizar mejor el balanceo de la carga, no obstante suele generar bastante "overhead" en tiempo de ejecución.

1.2.e.4.3 Optimos Versus Subóptimos

En el caso de que toda la información referente al algoritmo sea conocida, así como las características del sistema donde se va a ejecutar se pueden conseguir asignaciones óptimas basadas en la maximización o minimización de alguna figura de mérito seleccionada: minimización del tiempo de respuesta de la aplicación, maximización en cuanto a la utilización de los recursos del sistema, minimización del tiempo de ejecución, etc. Algunas de las técnicas utilizadas están basadas en teoría de grafos, soluciones de tipo enumerativo, etc; la figura 1.7, recoge estas técnicas remarcándolas en la taxonomía. En el caso de que los problemas anteriores sean intratables desde el punto de vista algoritmico, podemos intentar soluciones subóptimas, basadas en técnicas aproximadas o heurísticas.

1.2.e.4.4 Aproximadas Versus Heurísticas

Las primeras aplican el mismo modelo computacional para el algoritmo que en el caso de óptimo, pero en vez de buscar en todo el espacio de soluciones, nos conformamos que la solución este dentro de unos rangos fijados a priori, considerando como posible valor de referencia, el tiempo de ejecución del camino crítico.

En la otra rama de nuestra clasificación nos encontramos con los algoritmos heurísticos, estos pertenecen al conjunto de métodos para resolver problemas cuya solución no puede ser expresada de forma directa con una expresión matemática (por ejemplo, el algoritmo que proporciona las soluciones de un polinomio de grado dos, puede ser expresado de forma directa, obteniendo las soluciones para los valores de la variable mediante una formula matemática), y utiliza métodos indirectos para obtener las posibles soluciones. Estos realizan las asunciones más reales acerca de las características de los algoritmos así como del sistema. También representan las soluciones al "scheduling" estático, que requiere una cantidad de tiempo más razonable en relación con la bondad de la solución obtenida.

1.2.e.5. Estrategias básicas para afrontar el problema del "scheduling". Funcionamiento.

Una vez planteados los tipos más comunes definidos a la hora de plantear el problema del "scheduling", ver figura 1.7, pasaremos a describir someramente el mecanismo de funcionamiento de las estrategias de resolución planteadas y que aparecen remarcadas en la taxonomía. La no consideración de las comunicaciones interproceso por parte de la teoría de colas, hacen estas técnicas inapropiadas para resolver el problema general de "scheduling", por tanto nos centraremos en la descripción de aquellas que se aplican para casos más generales.

1.2.e.5.1 Basadas en teoría de grafos

Esta técnica ha sido adoptada por algunos investigadores [5]. El método representa los módulos a ser asignados como un conjunto de nodos en un grafo. Es importante señalar la naturaleza del conjunto de grafos utilizados, estos, son grafos de comunicación, en el sentido, de que las tareas del mismo son independientes, de tal forma, que las necesidades de comunicación que requieren las tareas del grafo, no implica ninguna relación de precedencia entre ellas. Asumiremos que los tiempos de comunicación entre cada par de módulos es conocido y es representado por el peso de un arco no dirigido que conecta los dos nodos. Un costo cero significa que ninguna comunicación tiene lugar entre dos nodos, y por tanto no están conectados en el grafo. Un costo infinito significa que esos nodos deberán ser asignados al mismo procesador. Los pesos de los arcos representan el costo de comunicación entre dos nodos no coresidentes en un procesador. Cualquier par de nodos asignados al mismo procesador se asume una comunicación interproceso igual a cero.

La estrategia de asignación en este modelo consiste en minimizar una función de costo, definida como la suma entre el costo de procesamiento y el costo de comunicación. A la hora de representar la asignación de módulos a procesadores definiremos una matriz X donde Xi,k = 1 si el módulo Mi es asignado al procesador Pk y cero en cualquier otro caso; una matriz Q = {qi,k}, i = 1 ... m, k = 1, ...,n donde qi,k representa el costo de procesamiento para el módulo Mi en el procesador Pk. Asumiremos que este costo es conocido. Un valor infinito de qi,k implica que el módulo Mi no puede ser ejecutado en el procesador Pk.

Sea ci,j el costo de comunicación entre los módulos Mi y Mj. El costo total de procesamiento para una tarea dada puede ser expresado por la siguiente función de asignación, Y.

$$\text{Costo}\ (Y) = \sum_{k}\sum_{i}\left\{q_{i,k}\,x_{i,k} + \sum_{i<k}\sum_{j<i} c_{i,j}\,x_{i,k}\,x_{j,l}\right\} \qquad \text{(Ec-1)}$$

Donde el primer término de (1) representa el costo de procesamiento para cada módulo según el procesador donde ha sido asignado. El costo mínimo para la asignación es obtenida aplicando un algoritmo de corte mínimo en el grafo.

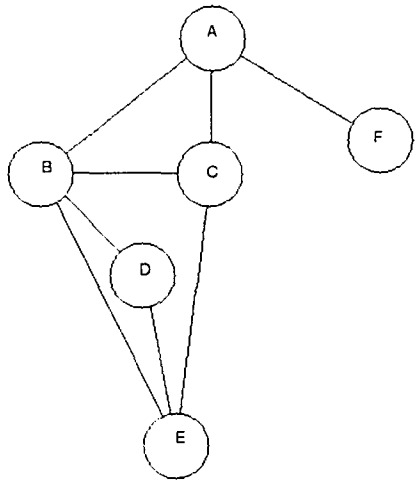


Figura 1.8

Para ilustrar el funcionamiento del método anterior, consideraremos un ejemplo de aplicación, debido a Stone [5]. La tarea consiste de seis módulos, { A, B, C, D, E, F }, los cuales tienen los tiempos de comunicación entre módulos que se muestran en la tabla 1.2 , y los costos de procesamiento que se muestran en la tabla 1.3, el ejemplo será realizado para dos procesadores P1 y P2, el grafo de tareas puede ser construido asociando nodos a cada uno de los módulos y arcos representando la comunicación entre módulos, este se muestra en la figura 1.8 . Con objeto de representar los costos de procesamiento, tal como viene fijado en la ecuación (1), se añaden dos nuevos nodos en el grafo para representar los dos procesadores disponibles P1 y P2. El costo de ejecutar cada módulo en el P1 es denotado por el arco que une el módulo en cuestión con el nodo P2. De la misma manera, el costo de ejecutar cada módulo en el procesador P2, es denotado por el arco que une el nodo que representa el módulo con el nodo que representa a P1. Si ahora aplicamos un algoritmo de corte mínimo al grafo de la figura 1.8, obtendremos el corte que se muestra en la figura 1.9 representado por una linea gruesa. Este proporciona la

MODULO	A	B	C	D	E	F
A		6	4	0	0	12
B			8	12	3	0
C				0	11	0
D					5	0
E						0
F						

Tabla 1.2

MODULO	Costo de Procesamiento P1	Costo de Procesamiento P2
A	5	10
B	2	infinito
C	4	4
D	6	3
E	5	2
F	infinito	4

Tabla 1.3

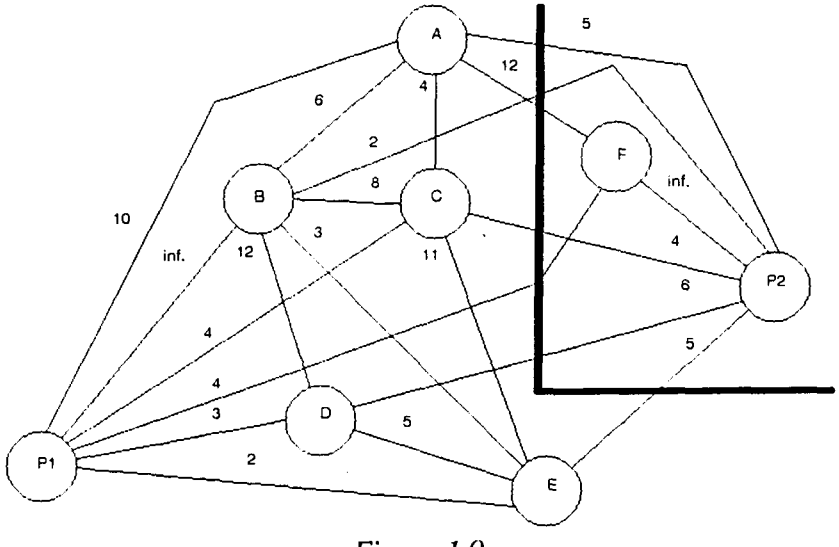


Figura 1.9

asignación que genera el mínimo coste para los módulos anteriormente dados en un entorno de dos procesadores.

Mientras este método es atractivo por su simplicidad, tiene unas limitaciones bastante importantes. En primer lugar, la solución de corte mínimo solamente es aplicable para dos o tres procesadores, una extensión de este método para un número no determinado de procesadores,requiere un algoritmo de corte mínimo que rapidamente se convierte en intratable. Esto limita la utilidad del método para la mayoría de aplicaciones. Una extensión de este método ha sido propuesta por Stone para entornos de más de cuatro procesadores cuando la interconexión entre módulos tiene estructura de árbol. Pero quiza la restricción más importante es la imposibilidad del método para permitir tratar sistemas donde debamos considerar precedencias a la hora de ejecutar los distintos módulos que componen la aplicación.

1.2.e.5.2.- Programación entera 0-1

En este método se formula el problema de la asignación de tareas como un problema de optimización y lo soluciona a través de la estrategia matemática de programación entera.

Como en la estrategia anterior, el objetivo es maximizar el rendimiento del sistema minimizando el costo total del sistema según una ecuación similar a la definida en (Ec-1).

El costo de procesamiento puede ser representado por la matriz Q definida anteriormente. Con objeto de representar las comunicaciones entre procesos se reemplaza la función simple de costo C, por el producto de dos matrices; una matriz volumen y una matriz de distancias. La matriz de volumen podemos definirla como $V = \{ v_{i,j} \}, i = 1, \ldots, m ; j = 1, \ldots, m$ donde $v_{i,j}$ representa el volumen de datos enviados desde el módulo Mi al módulo Mj. Si $v_{i,j} = 0$, el módulo Mi no comunica con el módulo Mj.

Consideremos el entorno de procesamiento distribuido mostrado en la figura 1.10. Se define la matriz distancia entre el conjunto de procesadores D = {dk,l},

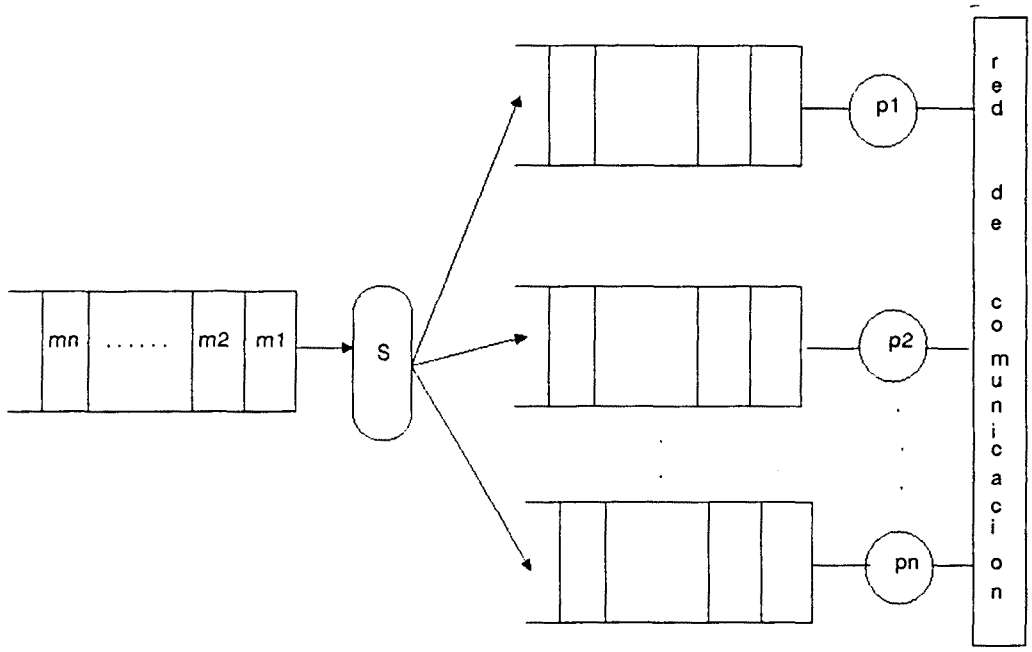


Figura 1.10

k = 1, ... ,n ; l = 1, ... ,n donde n es el número de procesadores. La distancia, dk,l es una medida de los costos de comunicación entre Pk y Pl. Si dos procesadores no están conectados, entonces dk,l es infinito. Esta medida de distancia permite en el modelo representar las interconexiones en la red. Permitiendo que dk,k sea distinto de cero, podemos representar el costo de comunicación entre módulos residentes. Esto puede ser útil para algunos entornos de tipo distribuido. Algunas características importantes de la matriz distancia son discutidas en detalle en [6]. Nosotros asumiremos que las distancias son conocidas y fijadas para cada par de procesadores.

Nosotros ahora podemos formular la función objetivo para el modelo por la siguiente función de costo:

$$\text{Costo}\ (X) = \sum_{k}\sum_{i}\left\{q_{i,k}\ x_{i,k} + \sum_{l<k}\ \sum_{j<i} wv_{i,j} d_{k,l} x_{i,k} x_{j,l}\right\} \qquad \text{(Ec-2)}$$

El primer término de la ecuación (Ec-2) representa el costo de procesamiento para cada módulo en su procesador asignado. El segundo término de la suma el producto entre el volumen y distancia representa el costo de cmunicación entre procesos. La constante w es utilizada para escalar los valores de costos de procesamiento y comunicación entre procesos para diferentes unidades de medida.

Un entorno de memoria limitada será representado por:

$$\sum s_i \; x_{i,k} \leq R_{k,k} = 1 \; \ldots\ldots \; n \qquad \text{(Ec-3)}$$

, donde si representa la cantidad de almacenamiento requerida por el módulo Mi y Rk la capacidad de memoria del processador Pk. Las restricciones de tiempo real vienen expresadas por:

$$\sum u_i \; x_{i,k} \leq T_{k,k} = 1 \; \ldots\ldots \; n \qquad \text{(Ec-4)}$$

, donde ui representa el tiempo requerido por el módulo Mi, y Tk el tiempo límite requerido para procesar los módulos que residen en el procesador Pk para una tarea. La ecuación (Ec-4) especifica que la cantidad de tiempo requerida para procesar todos los módulos que residen en un procesador simple no debe exceder del tiempo límite real requerido.

En este estado estamos en disposición de realizar la asignación de tareas minimizando la ecuación (Ec-2) sujeta a las restricciones de la ecuación (Ec-3) y (Ec-4). Esto puede resolverse como un problema de programación no-lineal 0-1 entera. Por tanto, la ecuación no lineal entera 0-1 puede ser linealizada añadiendo además restricciones, que simplifican de gran manera el proceso de solución.

La técnica expuesta anteriormente es bastante flexible ya que permite introducir restricciones dependiendo del tipo de aplicaciones a evaluar, lo cual es bastante didicil si no imposible en el modelo anteriormente expuesto de grafos teóricos.

No obstante el modelo de programación lineal 0-1 entera tiene fundamentalmente dos puntos de crítica bastante importantes; la representación del estado en curso en las resticciones de tiempo real, y la representación de las relaciones de precedencia entre módulos; ya que ambos factores introducen colas de espera en el sistema de manera compleja. Bajo estas consideraciones, las restricciones de tiempo real aparecen como $fi(ui\ xi,k ; pi,j)\ \ Tk, k = 1, ..., m$ donde pi,j representa la relación de precedencia entre los módulos Mi y Mj. La función fi representa el tiempo de procesamiento, incluyendo la cola de espera para procesar el módulo Mi en el procesador Pk.

1.2.e.5.4.- Métodos iterativos

Como es conocido, los mejores algoritmos hasta la fecha para resolver problemas de tipo combinatorio son los basados en técnicas de enumeración implicita (Branch and bound y programación dinámica) [7]. Es importante notar que este tipo de algoritmos no son de complejidad polinomial por tanto la bondad de la solución obtenida dependerá fuertemente del tiempo de ejecución de los mismos y del tamaño del problema. No obstante son de hecho efectivos para generar soluciones óptimas en muchos casos. Con objeto de dar una idea de su funcionamiento general presentaremos una extensión del mecanismo de "branch and bound" analizada por Kholer y Steiglitz [7], información más detallada sobre esta clase de algoritmos se puede encontrar en [8].

Cada algoritmo es caracterizado por una tupla de nueve parámetros (Bp, S, E, F, D, L, U, BR, RB) donde:

1) Bp, es la regla de bifurcacción para las permutaciones problema.

2) S, es el siguiente nodo en la regla de selección.

3) E, es el conjunto de reglas para la eliminación de un nodo.

4) F, es una función característica utilizada para identificar soluciones parciales que no son factibles.

5) D, es la relación de dominancia del nodo.

6) L, es la función de costo limite inferior para el nodo. Normalmente suele considerarse el límite inferior como cota, representando este, el mínimo tiempo (suponiendo que se dispone del número de recursos) que se necesita para la ejecución del nodo sin introducir retardos.

7) U, es el límite superior inicial de la solución. Puede ser una cota conocida de antemano para un determinado algoritmo, o una cota teórica como puede ser la función de límite superior que se asocia a cada nodo cuando se calculan los límites superiores en las cotas de Fernández y Bussell [17].

8) BR, es rango deseado dentro del cual consideramos soluciones.

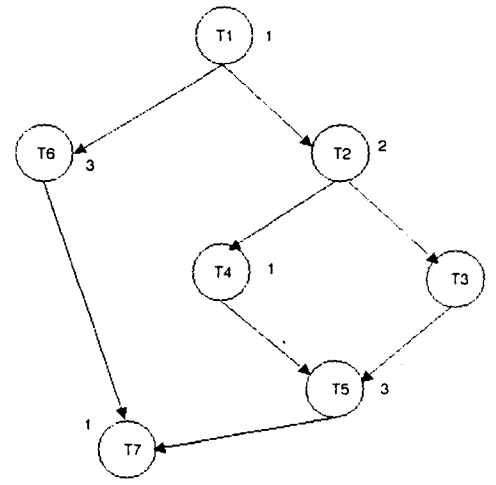


Figura 1.11

9) RB, es el vector límite de recursos cuyos componentes son limites superiores en tiempo, y capacidad de memoria disponibles para solucionar el problema.

Con objeto de ver algunas de las funciones asociadas a los parámetros que se especifican en la tupla anterior pasaremos a presentar un pequeño ejemplo. Siguiendo el trabajo de Schare [8], cada lista de asignación puede ser representado por una permutación p1 =l1,l2,l3, ..., ln. Si consideramos el grafo de la figura 1.11 con los tiempos de ejecución correspondientes, las trazas de Gantt para diferentes asignaciones ("schedules") realizadas se muestran en la figura 1.12. Por ejemplo, las

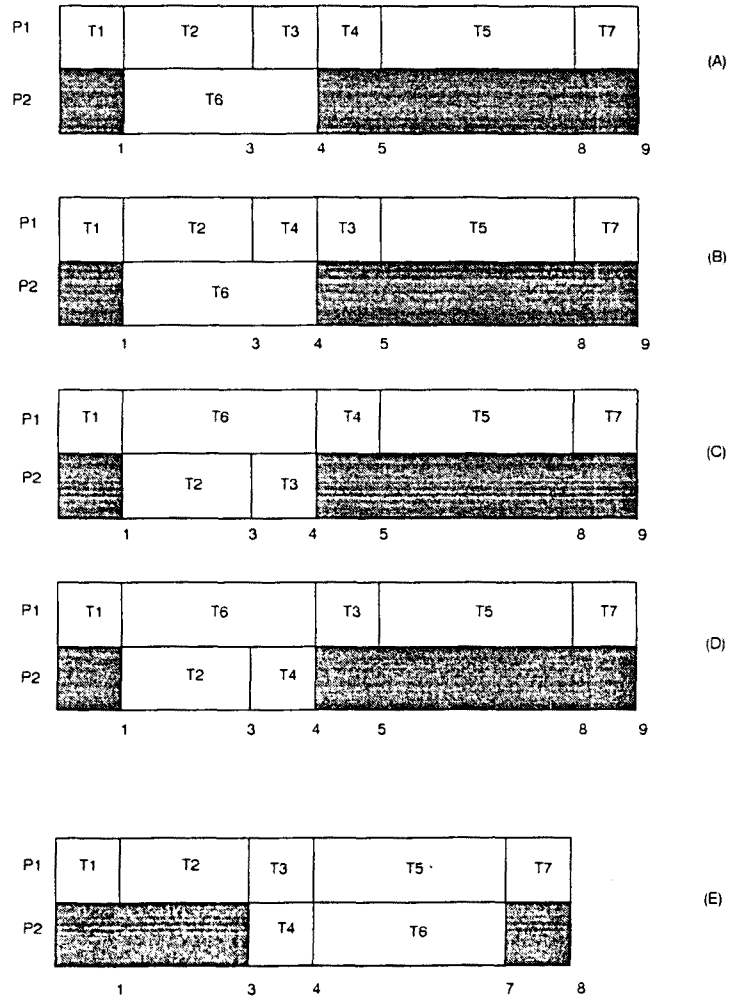


Figura 1.12

trazas de Gantt representadas en la figura 1.12 a,b,c y d, corresponden a las permutaciones 1263457, 1264357, 1623457 y 1624357, respectivamente, siendo el "schedule" óptimo mostrado en la figura 1.12e el correspondiente a la permutación

1234567. Notar que el "schedule" 1263457 y 1623457 son idénticos excepto que la asignación para el procesador para T2, T3, y T6 han sido cambiadas. Ya que el tiempo de comienzo de las tareas es el mismo los dos schedules son redundantes. Estos schedules redundantes pueden ser facilmente eliminados. Como ciertas permutaciones no son soluciones aceptables ya que i no puede aparecer antes que j si la tarea Tj es predecesora de la Ti. El espacio de soluciones es reducido definiendo la función característica F que tiene como valor cero cuando el schedule es redundante o no cumple las restricciones.

La relación de dominancia D es utilizada en unión con la regla de eliminación E para cortar el árbol enumerativo. Una solución parcial &y dominará &z, i.e., (&y,&z) pertenece D, si y sólo si la solución mínima que completa la solución comenzando con &y es al menos tan corta como la solución mínima que se obtiene comenzando con &z. Las reglas de eliminación por tanto son utilizadas para eliminar nodos dominantes y sus subárboles.

La función límite inferior L asigna a cada solución parcial &i un número real L(&y) representando un límite inferior en el tiempo de finalización de todas las soluciones completas que comienzan con la solución parcial &y. Se asume que tiene las siguientes propiedades: 1) si &z es descenciente de &y, entonces L(&z) = L(&y) y 2) para cada solución completa &yN, L(&yN) es el "schedule" de finalización &yN. Ya que el límite inferior es utilizado por una de las reglas de eliminación para cortar el árbol de búsqueda, estos límites pueden contribuir a obtener algoritmos más eficientes.

El límite superior inicial U puede ser tomado como el "schedule" generado por el camino crítico heurístico. Seleccionando BR = 0 (donde la desviación máxima relativa de la solución a aceptar con respecto a la óptima es entonces cero) el algoritmo de "branch and bound" puede ser utilizado para verificar la optimalidad del "schedule" proporcionado por el camino crítico o generar un "schedule" óptimo si el

límite de los recursos no son excedidos. Los resultados obtenidos utilizando esta estrategia muestran la eficiencia de este método.

1.2.e.5.4 Heurísticos

Los modelos heurísticos proporcionan soluciones aproximadas al problema de la asignación de tareas. Se aplican para la resolución de problemas de tipo enumerativo de naturaleza no convergente . Esta técnica requiere mucho menor tiempo de cálculo que el método de programación entera. Por tanto puede aplicarse a problemas de complejidad mucho mayor. Por encontrarse dentro de este último grupo los algoritmos que se presentan en este trabajo el modelo será presentado ampliamente y dejamos su exposición para más adelante.

1.3. Formulación del problema y objetivos del trabajo

Los trabajos publicados referentes al problema del "scheduling" para sistemas multiprocesadores no consideran la existencia de nodos cuyos tiempos puedan variar con objeto de modelar ciertos tipos de lazos y estructuras del tipo condicional (aunque muchos autores hacen especial mención a la necesidad de incorporar estas estructuras en los distintos modelos de resolución del problema del "scheduling" [14,15,16,63]). Este conocimiento en general no es disponible de forma cuantitativa en tiempo de compilación. En el mejor de los casos un compilador únicamente puede proporcionarnos una estimación de los tiempos de ejecución promedio para estos nodos. Nuestro trabajo tiene por objeto relajar la restricción anterior permitiendo que algunos de los nodos puedan variar su tiempo de ejecución dentro de un rango de valores conocidos, con objeto de aproximarnos más a la situación real que presentan los programas modelados por esos grafos.

Por tanto, en los algoritmos de "scheduling" que describiremos en este trabajo, la posibilidad de variación de los tiempos de ejecución de algunos nodos formará parte de la formulación del problema del "scheduling".

Las características básicas de los algoritmos de "scheduling" propuestos se exponen a continuación:

a.- Tipo estático

b.- Naturaleza heurística

c.- No apropiativos

d.- Sin restricciones en el orden de precedencias

e.- Tiempos de ejecución de tareas arbitrario

f.- Tiempos de ejecución variables para algunas tareas

g.- Tiempos de comunicación despreciables frente al volumen de cómputo de las tareas (selección adecuada de la granularidad)

h.- Número de procesadores fijo

A continuación expondremos las asunciones realizadas en este trabajo acerca de los apartados g) y h), así como las consideraciones que nos han llevado a fijar su valor para el desarrollo de los planificadores "schedulers" que vamos a tratar.

1.3.1 Tiempos de comunicación despreciables

El hecho de considerar los tiempos de comunicación entre tareas despreciables supone que la relación entre el volumen de cómputo de las tareas que componen el grafo es mucho mayor que los tiempos de comunicación entre ellas.

Es importante resaltar el hecho de la posible adaptabilidad de las aplicaciones representadas mediante el modelo de DAG frente a posibles tipos de arquitecturas, adecuando en cada caso, dependiendo de las características intrínsecas de la

arquitectura en cuestión, la relación granularidad de la tarea/comunicación y generar para cada tipo de arquitectura siempre que sea posible, un nuevo DAG de tareas que implícitamente ya esté considerando las particularidades en cuanto a la comunicación de la arquitectura subyacente.

1.3.2 Elección del número de procesadores

Nosotros fijaremos el número de procesadores aplicando las técnicas de E.Fernández y B.Bussell [17], que proporcionan, para DAG y unidades homogéneas de procesamiento, el mínimo numero de procesadores que se necesitan para ejecutar el grafo de tareas en un tiempo que no exceda del tiempo de ejecución del camino crítico para el grafo. El camino crítico representa la longitud del camino más largo desde el nodo inicial al nodo final del grafo de tareas, y por tanto, la cota mínima de tiempo que puede ser obtenida para cualquier algoritmo de planificación en el mejor de los casos.

El presente trabajo está organizado de la forma siguiente. El capítulo 2, describirá el mecanismo de planificación basado en lista como método para afrontar el problema del "scheduling" en sistemas multiprocesadores; así como el problema que presentan de sensibilidad frente a la variación de los datos de entrada, igualmente, se presentará un ejemplo para mostrar su funcionamiento y las principales ventajas e inconvenientes que presentan estos métodos en cuanto a su posible aplicación a la hora de generar algoritmos de "scheduling" estáticos o dinámicos, sus límitaciones en cuanto a la bondad de las soluciones obtenidas, y las anomalias que presentan.

En el capítulo 3, se expondrá el método propuesto para resolver el problema de sensibilidad que presentan los algoritmos de "scheduling" basados en lista, cuando se considera que determinadas secciones de código son fuertemente sensibles a los posibles valores de los datos de entrada. Igualmente se describirán los algoritmos utilizados para resolver el problema anterior, cuando dichas secciones de código pueden variar su tiempo de ejecución para dos valores , y se mostrará un ejemplo de funcionamiento. Posteriormente se analizarán tanto la complejidad teórica como la

complejidad real de los algoritmos propuestos para un conjunto de grafos de prueba. Una vez visto el funcionamiento de los algoritmos, se describirán el conjunto de grafos que hemos utilizado para realizar nuestra experimentación así como los índices de rendimiento que utilizaremos para mostrar la bondad de los mismos. Por último expondremos los resultados obtenidos mediante su aplicación y los comentaremos.

El capítulo 4, extenderá los resultados obtenidos en el capítulo anterior, donde el estudio de la sensibilidad de los tiempos de ejecución de las secciones de código se reducía a dos únicos valores, y expondremos los resultados de nuestra experimentación cuando estas secciones pueden tomar tiempos de ejecución dentro un conjunto de valores. Igualmente mostraremos el comportamiento de nuestros algoritmos para el conjunto de grafos utilizados en nuestra experimentación y mediremos la bondad de los mismos cuando queremos representar con la asignación que proporcionan los dos valores extremos, todo el posible rango de valores que pueden tomar los nodos fuertemente sensibles a los datos de entrada. Finalmente resumiremos las conclusiones que se pueden extraer de este trabajo y cerraremos el mismo, comentando algunas de las lineas abiertas que la realización del mismo nos ha suscitado.

El apéndice A, mostrará el entorno de trabajo desarrollado para realizar la experimentación; el apéndice B, recoge de forma exhaustiva, todo el trabajo experimental realizado, los grafos utilizados, tiempos de ejecución para todas las tareas y todos los resultados obtenidos motivo de las conclusiones que se muestran en el presente trabajo.

CAPITULO 2

POLITICAS DE PLANIFICACION "SCHEDULERS" BASADOS EN LISTA, SENSIBILIDAD

- Introducción

El presente capítulo tiene por objeto, mostrar la fuerte dependencia, que presentan determinadas secciones de código en su tiempo de ejecución cuando varian los datos de entrada y se intenta aplicar heurísticas basadas en el método de lista, para resolver el problema de la planificación estática en sistemas multiprocesador formulado el problema en su caso general.

Nuestro objetivo consiste en diseñar algoritmos de "scheduling" que tengan en cuenta para su solución, la fuerte sensibilidad que presenta el modelo anterior de heurísticas, en base a representar de forma menos restrictiva, el comportamiento de los programas. Para este fin, a continuación se mostrará a nivel general, el funcionamiento de los planificadores basados en lista, para posteriormente presentar mediante un ejemplo esta dependencia.

2.1.- Justificación de los "Schedulers" de lista

A la hora de plantearse la elección de un algoritmo de "scheduling", la primera táctica que se nos ocurre es la de ir asignando tareas listas a procesadores libres en el primer instante que se pueda, e ir avanzando en la simulación del grafo de tareas para obtener mediante modelos de prestaciones, como trazas de Gantt, la asignación de tareas a procesadores, el tiempo de ejecución final, la utilización de los recursos, etc. El problema que se observa con esta forma de trabajar es cuando se nos plantea la problemática de tener que decidir qué tareas asignar cuando el número de tareas listas

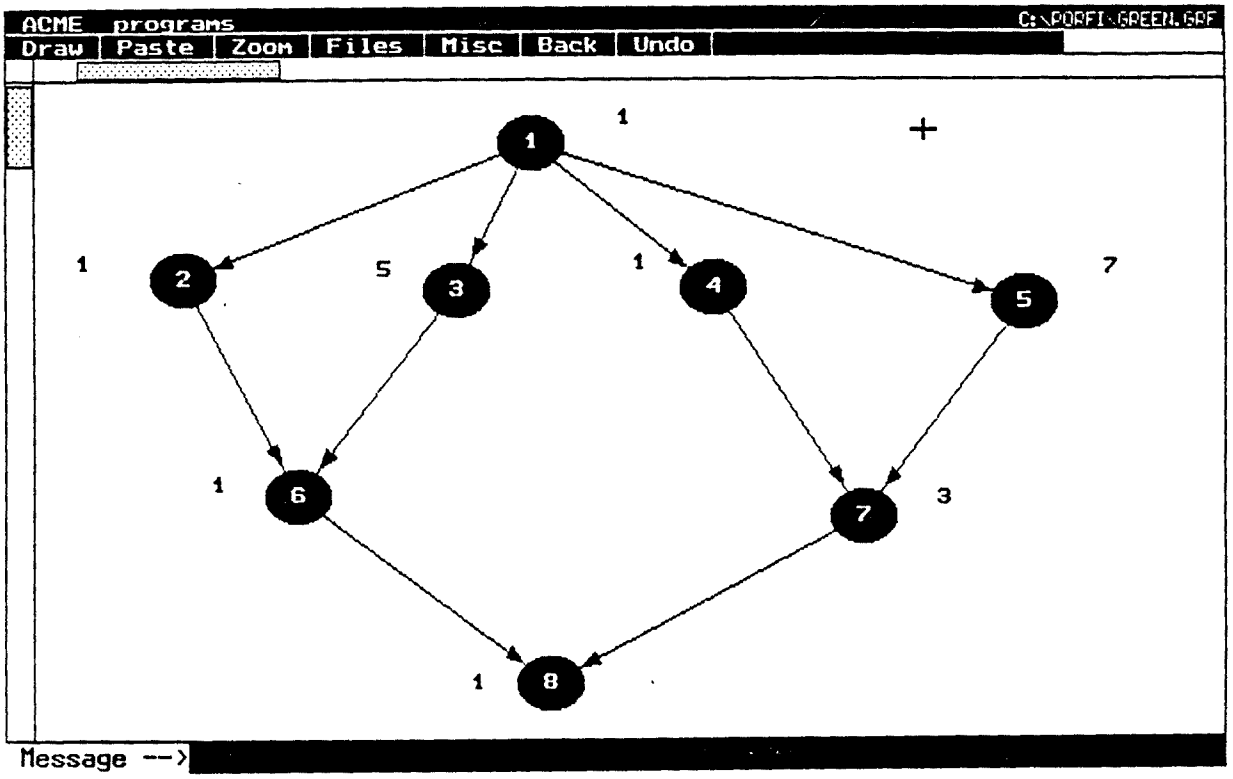


Figura 2.1

es mayor que el número de procesadores, situación por otro lado normal en este tipo de problemas. La importancia de la tarea elegida a asignar puede ser el factor clave

a la hora de obtener un buen rendimiento del planificador. A continuación presentamos un ejemplo que pondrá de manifiesto las afirmaciones expuestas en el punto anterior.

Sea el grafo de tareas representado en la figura 2.1, suponer que la asignación de tareas se va haciendo en el instante que van quedando listas las tareas y hay procesadores libres, además en caso de haber más tareas listas que procesadores libres, elegiremos las tareas a asignar de forma aleatoria. La figura 2.2a, muestra la asignación y las medidas de rendimiento para un sistema de dos procesadores y una

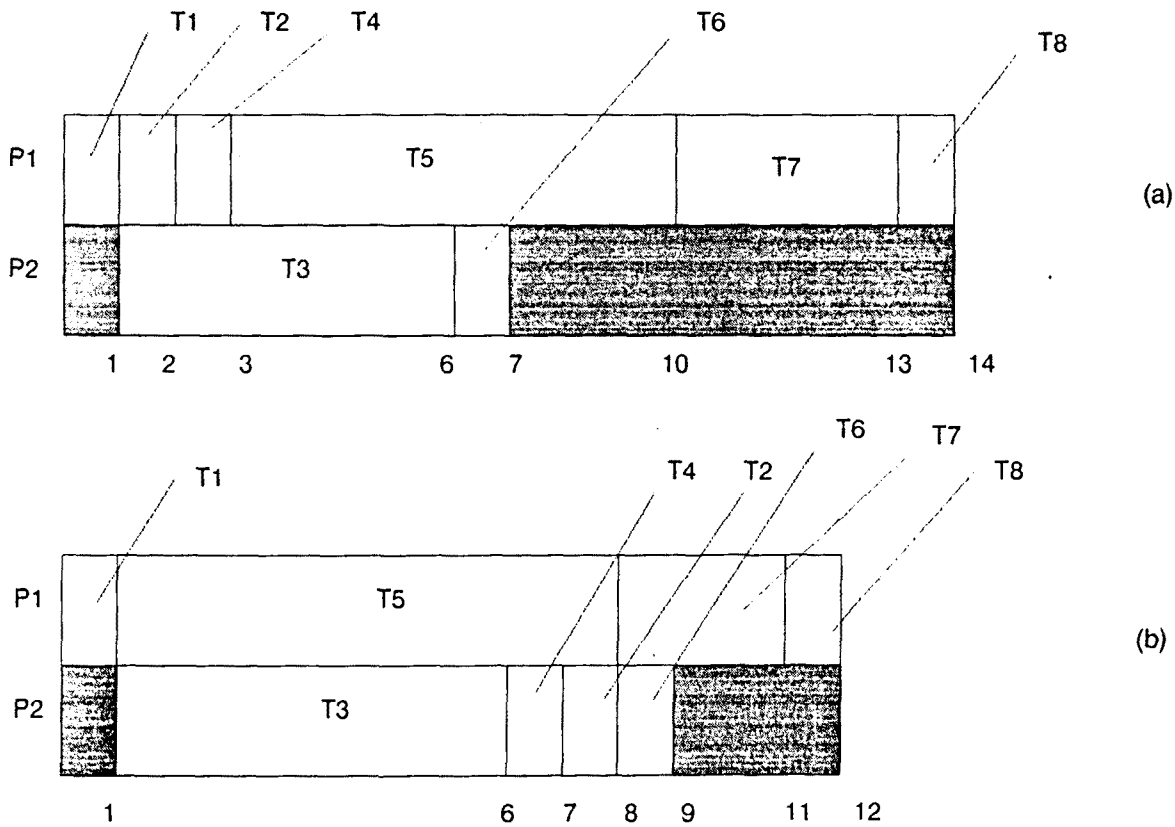


Figura 2.2

lista L = (1,2,3,4,5,6,7,8). Si el objeto de nuestros experimentos es la minimización del tiempo de ejecución del grafo, la medida que nos interesa fundamentalmente es el tiempo de ejecución final del mismo, en estas condiciones igual a 14. Observar la traza de Gantt, para el mismo ejemplo en la figura 2.2b, cuando la lista de tareas es L = (1,5,3,4,2,7,6,8) y dos procesadores, se observa que el tiempo de finalización del grafo es ahora de 12 unidades de tiempo, lo que implica una mejora del 15% respecto a la asignación aleatoria.

2.2.-"Schedulers" basados en lista

Las primeras investigaciones referenciadas en la literatura, acerca de la aparición de los métodos de lista para tratar el problema del "scheduling" en sistemas multiprocesador, se basan en los trabajos de Coffman y Graham referidos al estudio de "schedulers" óptimos de tipo no apropiativo para dos procesadores, y donde el tiempo de ejecución de las tareas es la unidad [4]. La propuesta de Coffman y Graham surge ante el planteamiento formulado por Fujii, Kasami y Ninomiya [11]; este, consiste en dividir el conjunto de tareas en pares de tareas compatibles e incompatibles, donde dos tareas se definen como compatibles si Ti no precede a Tj y Tj no es predecesora de Ti. Para un conjunto de tareas, suponiendo que m representa el máximo número de pares de tareas disjuntas, n-m es un límite inferior para ejecutar todas las tareas. El problema que se plantea entonces, es el de encontrar el máximo número de tareas compatibles y buscar una secuencia óptima de ejecución para el resto de tareas que minimice el tiempo de ejecución del grafo de tareas. Coffman y Graham ante el problema planteado anteriormente, desarrollan un algoritmo basado en la generación de una lista de tareas, y una forma de ejecución que permite generar tanto la asignación de tareas a procesadores como el orden de ejecución de las mismas.

2.2.1.- Funcionamiento

El funcionamiento de los planificadores basados en lista consta de dos fases bien diferenciadas; una primera fase que consiste en la construcción de una lista en base a una determinada función de costo, con objeto de maximizar o minimizar un

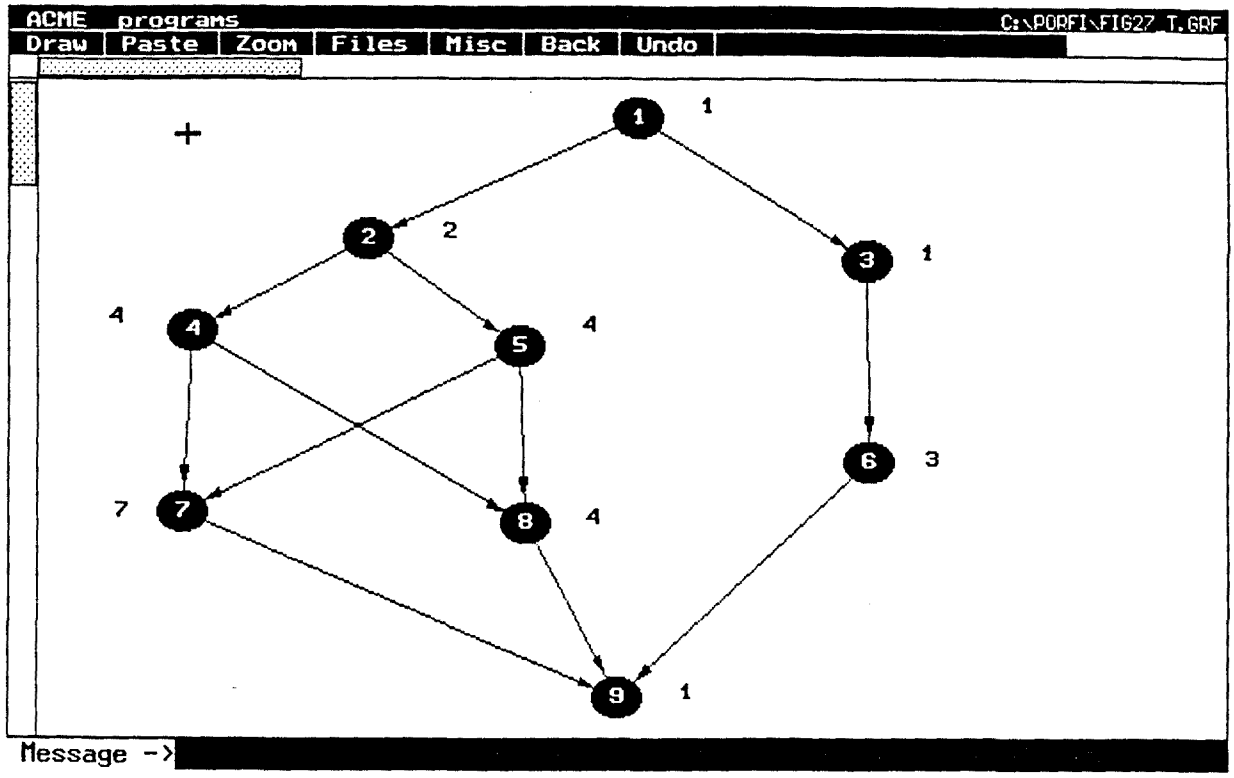


Figura 2.3

determinado índice de rendimiento, y una segunda fase que consiste en la asignación de las tareas a los procesadores [3,4,10,11,12,13,18,37,40].

Una vez que la lista ha sido generada, para proceder a la asignación de las tareas a los procesadores podemos proceder de dos maneras distintas:

a) Se realiza una asignación dinámica de las tareas a los procesadores. Para ello los procesadores, a medida que van quedando libres, podrían acceder al recurso común donde se encuentran las tareas y elegir entre las que están listas para ser ejecutadas, aquella que tuviese mayor prioridad, proceso que se repite hasta que todas las tareas han sido completadas.

b) Se realiza una simulación del proceso descrito anteriormente, con objeto de encontrar una asignación y su posterior carga en los procesadores, todo ello realizado "off- line". Posteriormente a partir de la asignación obtenida anteriormente se puede proceder a la ejecución del grafo de tareas.

Es interesante notar, en base a lo expuesto anteriormente, que este tipo de planificadores podría ser utilizado tanto como planificadores de tipo estático como dinámicos. En este tipo de planificadores es importante notar que la división en dos fases, una primera fase para generar la lista y una segunda fase de ejecución propiamente dicha, obedece más a factores de rendimiento del algoritmo que a la necesidad de dividir el proceso en dos fases. Observar que el proceso más importante a la hora de generar la lista, es el calculo de la función de costo para cada uno de los nodos, pero esta función podría calcularse en tiempo de ejecución, ya que cuando se realiza la asignación de tareas a procesadores, en primer lugar se busca el conjunto de tareas listas, una vez determinadas estas, podría perfectamente calcularse la prioridad de cada una de las tareas listas para determinar la candidata a ser asignada.

La aplicación de estos "schedulers" implica una forma específica a la hora de representar las aplicaciones, ya que presuponen una conocimiento exhaustivo de todos los tiempos de ejecución de las tareas en tiempo de compilación y por otro lado el hecho de eliminar las posibles comunicaciones entre tareas, adecuando previamente la relación, entre los volumenes de cómputo de las mismas y sus comunicaciones, con objeto de incorporarlas en el grafo de tareas.

Es interesante notar a la hora de simular la ejecución del grafo, que siempre que existan tareas listas para su ejecución y procesadores libres, se deberá realizar la asignacion de las mismas a los procesadores, ya que en los métodos de "scheduling" basados en lista, no se considera el hecho de introducir períodos de inactividad en los procesadores mientras existan en el grafo, tareas listas para su ejecución. Esta forma de ejecución, por otra parte la más lógica, en la que podemos pensar, es la causante de que este tipo de "schedulers" no proporcionen por lo general el tiempo óptimo del

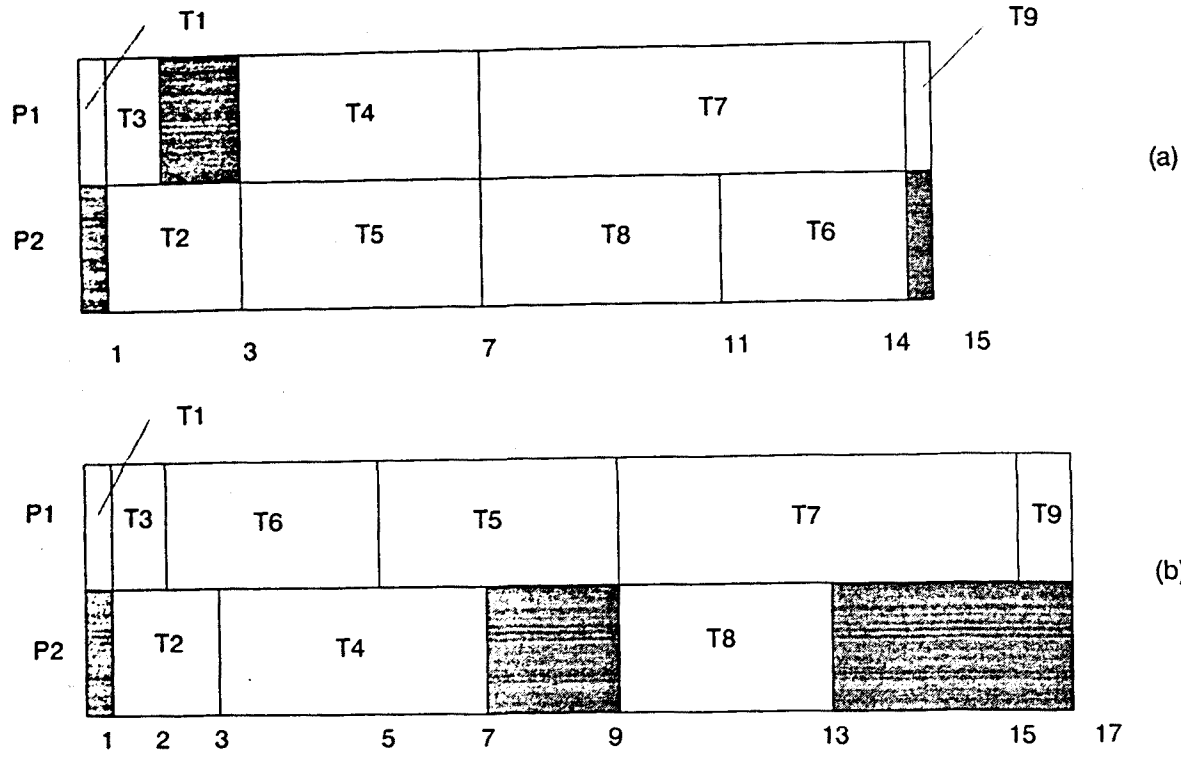


Figura 2.3a,b

algoritmo. La figura 2.3b muestra la traza de Gantt obtenida para el grafo de la figura 2.3, para dos procesadores, cuando se ejecutan las tareas tan pronto como son activadas, proporcionando un tiempo de ejecución final de 17 unidades. Según el grafo de tareas que se muestra en la figura 2.3 el tiempo mínimo (óptimo) de ejecución del grafo es de 15 unidades de tiempo (camino crítico). En la figura 2.3a se observa que la planificación óptima se obtiene introduciendo un período de inactividad (desde tiempo = 2 hasta tiempo = 3) en el procesador 1 retrasando la ejecución de la tarea 6 hasta tiempo = 11, pero generando un "length schedule" de 15 unidades de tiempo, que es el tiempo óptimo de ejecución del grafo.

El grafo ejemplo desarrollado a través de las figuras 2.3 y 2.3a,b nos ha permitido observar, como el hecho de conseguir la optimalidad en el tiempo de ejecución para el grafo de tareas puede implicar la existencia de huecos en ejecución.

2.2.2 Criterios de ordenación de la lista de tareas

El punto fundamental de los "schedulers" basados en lista, es la generación de la lista de prioridad a partir de la cual posteriormente se va a realizar la asignación. Normalmente la función de costo que se asocia a cada nodo, está en relación con el índice de rendimiento que se quiere optimizar; maximar la utilización de los procesadores, minimizar el tiempo de ejecución de la aplicación etc. A continuación se dan algunos de los criterios que se han utilizado en la literatura con objeto de calcular la prioridad de las distintas tareas que componen el grafo de la aplicación.

Hay que tener en cuenta que la elección de una tarea u otra se produce cuando un procesador que ha quedado libre accede a la lista de tareas y encuentra más de una que puede ser ejecutada, por tanto podemos encontrarnos casos en los que soló existe una tarea lista cada vez y por tanto los criterios de construcción de la lista de tareas en base a una determinada función de costo no influye en la asignación obtenida.

Algunos de los criterios más utilizados a la hora de construir la lista de prioridad [11,12] se presentan a continuación:

a) El nivel de los nodos : Se define el nivel de un nodo del grafo, como el camino más largo desde dicho nodo al final del grafo. Representa en tiempo minimo que ha de transcurrir para que pueda ejecutarse la parte del grafo que va desde el nodo en cuestión hasta el final del grafo.

b) El conivel de los nodos: Es el tiempo mínimo que ha de transcurrir desde que se empieza a ejecutar el grafo, para que el nodo en cuestión pueda comenzar su ejecución .

c) El número de sucesores de los nodos.

d) El número de predecesores de los nodos.

Adam, Chandy y Dickson han comparado a través de extensas simulaciones los rendimientos de estos schedules de lista en entornos sin restricciones [11], y demuestran que la función de costo que mejor se aproxima a las soluciones óptimas es aquella que intenta priorizar aquellas tareas que tienen el mayor nivel.

Es usual encontrarse a la hora de utilizar el nivel como función de costo para ordenar la lista de prioridad, que puedan existir varias tareas con el mismo nivel, la elección de la tarea candidata se realiza entonces en base a criterios que se denominan de segundo orden, tarea con mayor tiempo de ejecución , mayor número de sucesores etc. [12,13].

2.3.-Algoritmo CP/MISF (Critical Paht with the Most Inmediatly Sucessors First)

El método del camino crítico ha sido considerado el algoritmo heurístico más eficiente para solucionar el problema del "scheduling"; es una generalización del algoritmo de Hu [3]. Ya que el orden de prioridad no puede ser determinado de forma única cuando existe una gran cantidad de tareas que tienen el mismo nivel, dependiendo de la tarea elegida puede dar lugar al peor tiempo de ejecución.

La razón de elegir el CP/MISF como punto de partida para desarrollar nuestros algoritmos de planificación y problemas que presentan de sensibilidad frente a los datos obedece, a que fue el primer algoritmo de tipo estático basado en lista donde se adecuaron los tamaños de las tareas del DAG con objeto de obtener una granularidad óptima del problema, y por tanto poder considerar las comunicaciones despreciables; y otra parte, a que este planificador ha probado su validez en aplicaciones reales de control del brazo armado de robots con seis y siete grados de libertad [47,48]. No obstante, independientemente del "scheduler" de lista elegido, los problemas de sensibilidad que queremos mostrar, característicos de este tipo de planificadores se pueden observar igualmente partiendo de cualquier otro planificador de la familia.

En este apartado daremos una breve visión del método basado en el camino crítico (CP) eligiendo como tarea más prioritaria, en caso de igualdad del parámetro básico, aquella tarea que tenga un mayor número de sucesores inmediatos (Most Immediately Sucessors First).

Este método se basa en la idea de nivel a la hora de confeccionar la lista de prioridades. El camino crítico de un grafo podemos definirlo como el camino más largo desde el nodo de salida hasta el nodo entrada. El significado del mismo indica la cota mínima que podemos conseguir ya que no es posible obtener mejor tiempo de ejecución independientemente de los recursos disponibles en el sistema.

El algoritmo CP/MISF, consiste básicamente de los siguientes pasos:

Paso 1: Determinar el nivel para cada tarea.

Paso 2: Construir la lista de prioridad en orden descendiente de nivel y del número de tareas sucesivas.

Paso 3: Ejecutar el programa representado por el grafo según la lista de prioridad con objeto de encontrar la asignación y el tiempo de ejecución final.

Este algoritmo se basa en el concepto de nivel, la idea es que se ejecuten en primer lugar aquellas tareas que más tiempo les falta para llegar al final del grafo. Así se intenta que todas las ramas del grafo se vayan ejecutando concurrentemente y no haya casos en los que una rama haya avanzado mucho, pero ahora tenga que esperar la ejecución de otra rama del grafo que ha quedado retrasada, con lo que podría producirse entonces una secuencialización de determinadas partes del grafo que podría haberse ejecutado en paralelo.

2.4.-Sensibilidad de los Algoritmos de lista frente al tiempo de ejecución

Los trabajos desarrollados sobre el tema de planificación estática, no consideran la posible variación en los tiempos de ejecución de las tareas, ya que los grafos que representan los programas son acíclicos, por tanto los lazos de programa y las estructuras condicionales están dentro de los nodos. Para conocer estos tiempos, o bien conocemos el número exacto de iteraciones para los lazos en cada ejecución, o bien en base a estadísticas de ejecución del programa obtenemos una medida estimativa que los represente como tiempo de ejecución medio, ya que la medida exacta sólo la podremos conocer en tiempo de ejecución. Resumiendo, cuando realmente vayamos a ejecutar el grafo, nos encontraremos con que los tiempos de determinados nodos no necesariamente serán los que hemos considerado para confeccionar la lista, sino que pueden variar.

La necesidad de modelos que se aproximen más a la realidad ha sido remarcada por diversos autores en la literatura [13,14,15,41,63] con objeto de poder aplicar las técnicas de planificación estática a un conjunto más ámplio de programas.

Como consecuencia, comenzando de la asignación proporcionada por el CP/MISF, considerando ciertos tiempos de ejecución, cuando el grafo es ejecutado con algunas modificaciones en los tiempos de aquellos nodos que pueden variar, la asignación que nos proporcionará el "scheduler" de lista no será la mejor, y por tanto el tiempo de ejecución que obtengamos puede ser muy ineficiente. Esto puede ser visto en el siguiente ejemplo.

Partiendo del grafo mostrado en la figura 2.4, calculamos el nivel de cada nodo basándonos en el CP/MISF lo cual genera la Tabla 2.1a y a partir de ella elaboramos la lista de prioridades, Tabla 2.1b. Una vez generada esta, pasamos a calcular la asignación. Para ello, deberemos ejecutar de forma simulada la ejecución del grafo de forma que cada vez que un procesador finaliza la tarea en curso, busca una nueva tarea en la lista de prioridades y toma la primera tarea lista que encuentra. Este

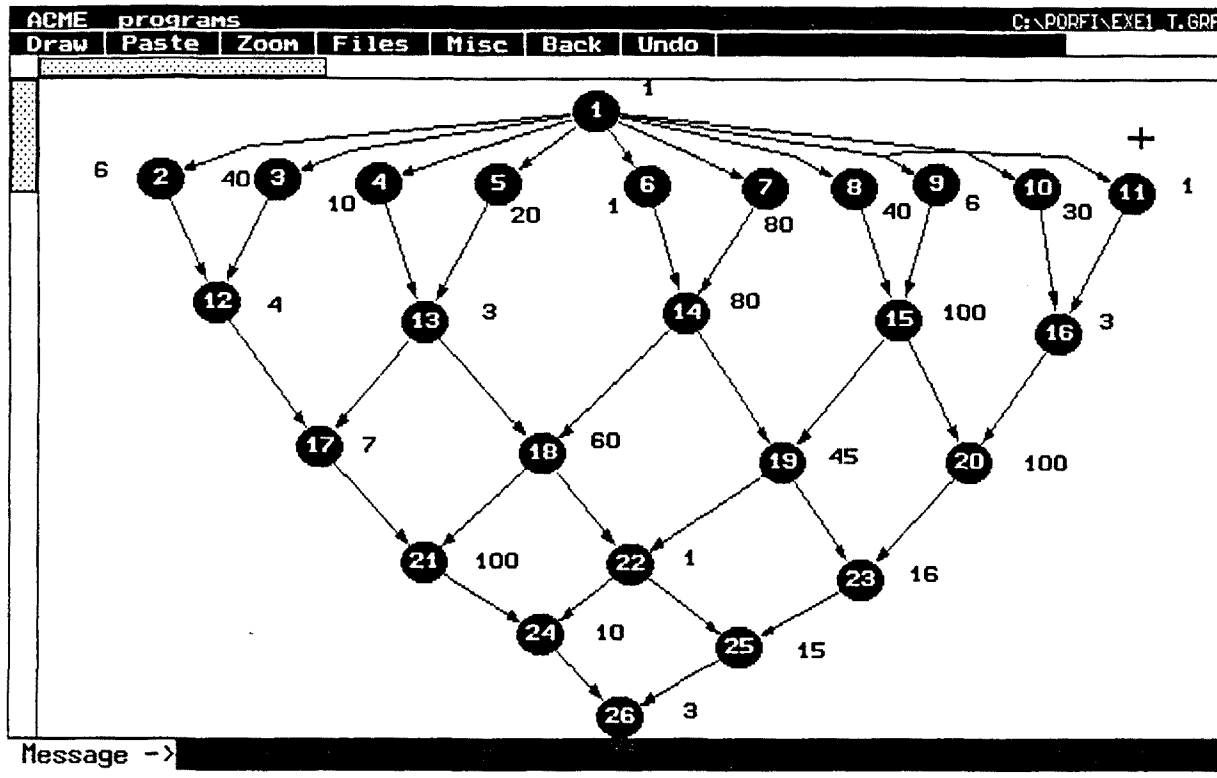


Figura 2.4

proceso se repite hasta que todas las tareas del grafo son asignadas. La Tabla 2.1c muestra la asignación para el ejemplo de la figura 2.4 para tres procesadores.

La traza de Gantt para el ejemplo de la figura 2.4, se muestra en la Tabla 2.3 incluyendo las tareas listas para cada procesador.

Supongamos ahora que partimos de la asignación mostrada en la Tabla 2.1c, obtenida para los tiempos de ejecución (Ti), en el grafo ejemplo mostrado en la figura 2.4. Si ahora para el mismo grafo de entrada, la ejecución real tuviese lugar, habiendo sufrido algunos nodos una modificación en sus tiempos de ejecución (Ti*),

Nodo	Nivel	L. Prioridad	Procesador	Nodo
1	334	1	1	1
2	130	7		7
3	164	8		14
4	186	6		18
5	196	14		21
6	254	9		24
7	333	15		26
8	274	5	2	8
9	240	4		15
10	167	13		20
11	138	18		23
12	124	10		25
13	176	3		
14	253	11	3	6
15	234	16		9
16	137	20		5
17	120	2		4
18	173	12		13
19	79	17		10
20	134	21		3
21	113	19		11
22	19	23		16
23	34	22		2
24	13	25		12
25	18	24		17
26	3	26		19
				22

Tabla : 2.1a 2.1b 2.1c

Tabla 2.1 a,b,c

Nodo	(Ti*)
4	30
5	60
8	20
9	5
13	100
15	10
17	60
18	20
19	100
20	50

(Ti*): Nuevo tiempo de ejecucion

Tabla 2.2

Processor	Node	Time	Start_time	Finish_time	Ready tasks
1	1	1	0	1	7(1)
1	7	80	1	81	14(81)
1	14	80	81	161	18(161)
1	18	60	161	221	21(221)
1	21	100	221	321	24(321)
1	24	10	321	331	26(331)
1	26	3	331	334	
2	idle	1	0	1	8(1)
2	8	40	1	41	15(41)
2	15	100	41	141	20(141)
2	20	100	141	241	23(241)
2	23	16	241	257	25(257)
2	25	15	257	272	
2	idle	62	272	334	
3	idle	1	0	1	[2,3,4,5,6,9,10,11](1)
3	6	1	1	2	[2,3,4,5,9,10,11](1)
3	9	6	2	8	[2,3,4,5,10,11](1)
3	5	20	8	28	[2,3,4,10,11](1)
3	4	10	28	38	[2,3,10,11](1);13(38)
3	13	3	38	41	[2,3,10,11](1)
3	10	30	41	71	[2,3,11](1)
3	3	40	71	111	[2,11](1)
3	11	1	111	112	2(1);16(112)
3	16	3	112	115	2(1)
3	2	6	115	121	12(121)
3	12	4	121	125	17(125)
3	17	7	125	132	
3	idle	29	132	161	19(161)
3	19	45	161	206	
3	idle	15	206	221	22(221)
3	22	1	221	222	
3	idle	112	222	334	

Asignación: Ti
Ejecución: Ti
Traza de Gantt y tareas listas

Tabla 2.3

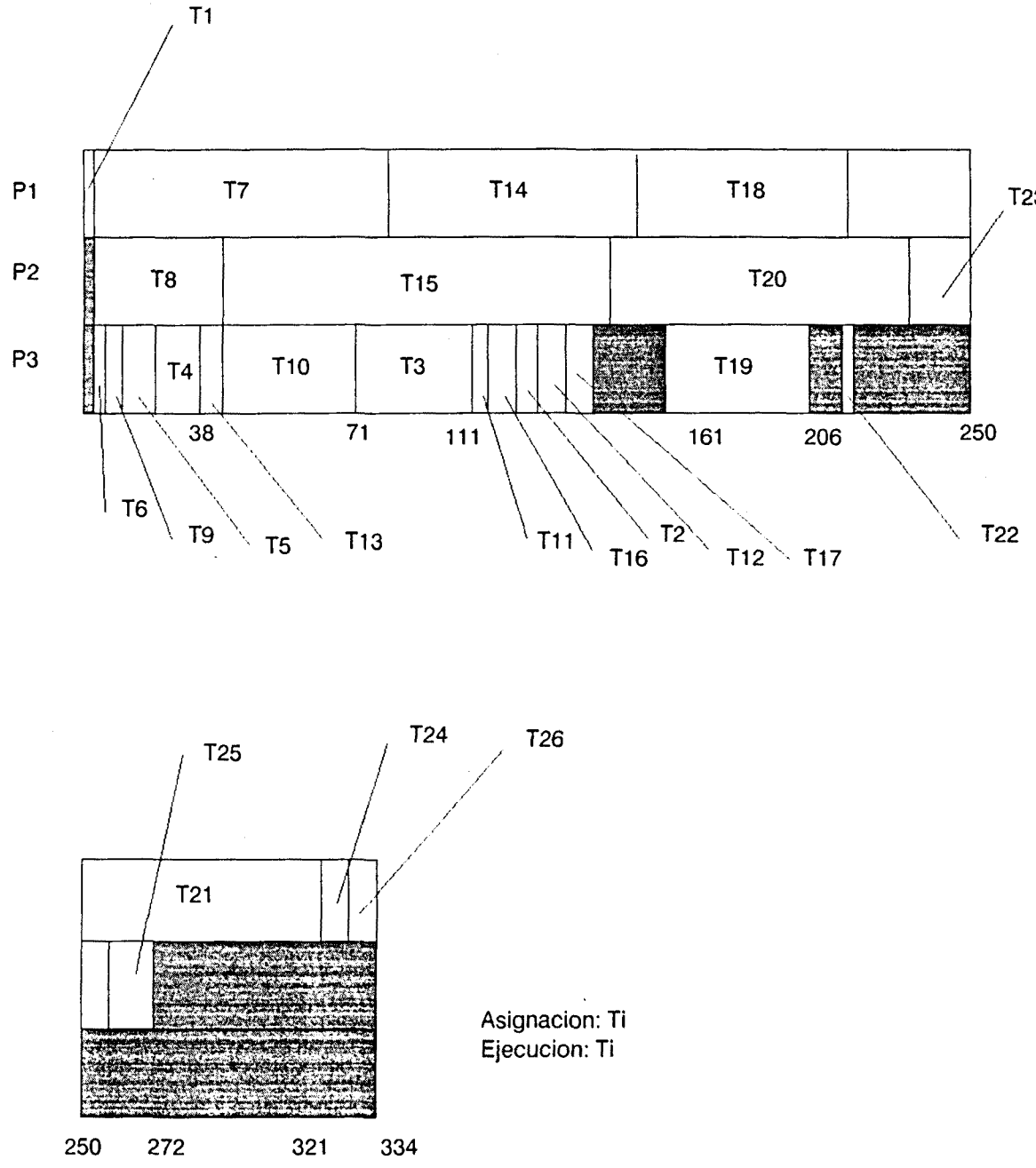

T1
P1
T7
T14
T18
T23
P2
T8
T15
T20
P3
T4
T10
T3
T19
38
71
111
161
206
250
T6
T9
T5
T13
T11
T16
T2
T12
T17
T22
T25
T24
T26
T21
Asignacion: Ti
Ejecucion: Ti
250
272
321
334

Figura 2.5

manteniendo el tiempo de ejecución del resto de nodos, tal como de observa en la Tabla 2.3, se obtiene la traza de Gantt mostrada en la Tabla 2.4, para los nuevos tiempos de ejecución (Ti*). El tiempo de ejecución para esta asignación es 475, pero si los tiempos modificados (Ti*) se considerasen como los tiempos iniciales para obtener la asignación utilizando el CP/MISF, se podria ver que el tiempo de ejecución sería 334. Es decir se ha producido una degradación en el tiempo de ejecución de 475 - 334 = 41 unidades (degradación aproximada del 12%). Este ejemplo muestra que la asignación inicial, evaluada para (Ti) no es eficiente si los tiempos de ejecución son modificados (Ti*).

En las figuras 2.5 y 2.6, se muestra una representación para el ejemplo anterior de las trazas de Gantt generadas a partir del ejemplo de la figura 2.4, donde se puede apreciar que el comportamiento del algoritmo CP/MISF no puede ajustarse a los nuevos valores de tiempos de ejecución de las tareas (Ti*); ya que este algoritmo parte de la base para encontrar la asignación final de tareas a procesadores que los tiempos de ejecución de las tareas son fijos. Es por tanto la sensibilidad de estos "schedulers" frente a los datos de entrada la que pretendemos afrontar y que será motivo de estudio en el capítulo 3; donde mostraremos las técnicas utilizadas y los resultados que proporcionan nuestros algoritmos, introduciendo parámetros nuevos a fin de hacer este tipo de planificadores menos sensibles a la variación del tiempo de ejecución de algunos de los nodos del grafo que represenat el programa.

Es importante notar, la impredecible dependencia que frente al tiempo de ejecución final del grafo, muestran este tipo de "schedulers" cuando variamos los parámetros que intervienen en la formulación del problema; enfrentando nuestra intuición a los datos obtenidos experimentalmente. A tal efecto utilizaremos los ejemplos de Graham modificados [10], observaremos qué sucede con el tiempo de ejecución del grafo modelo cuando cambiamos:

a) La lista de prioridad

b) El número de procesadores

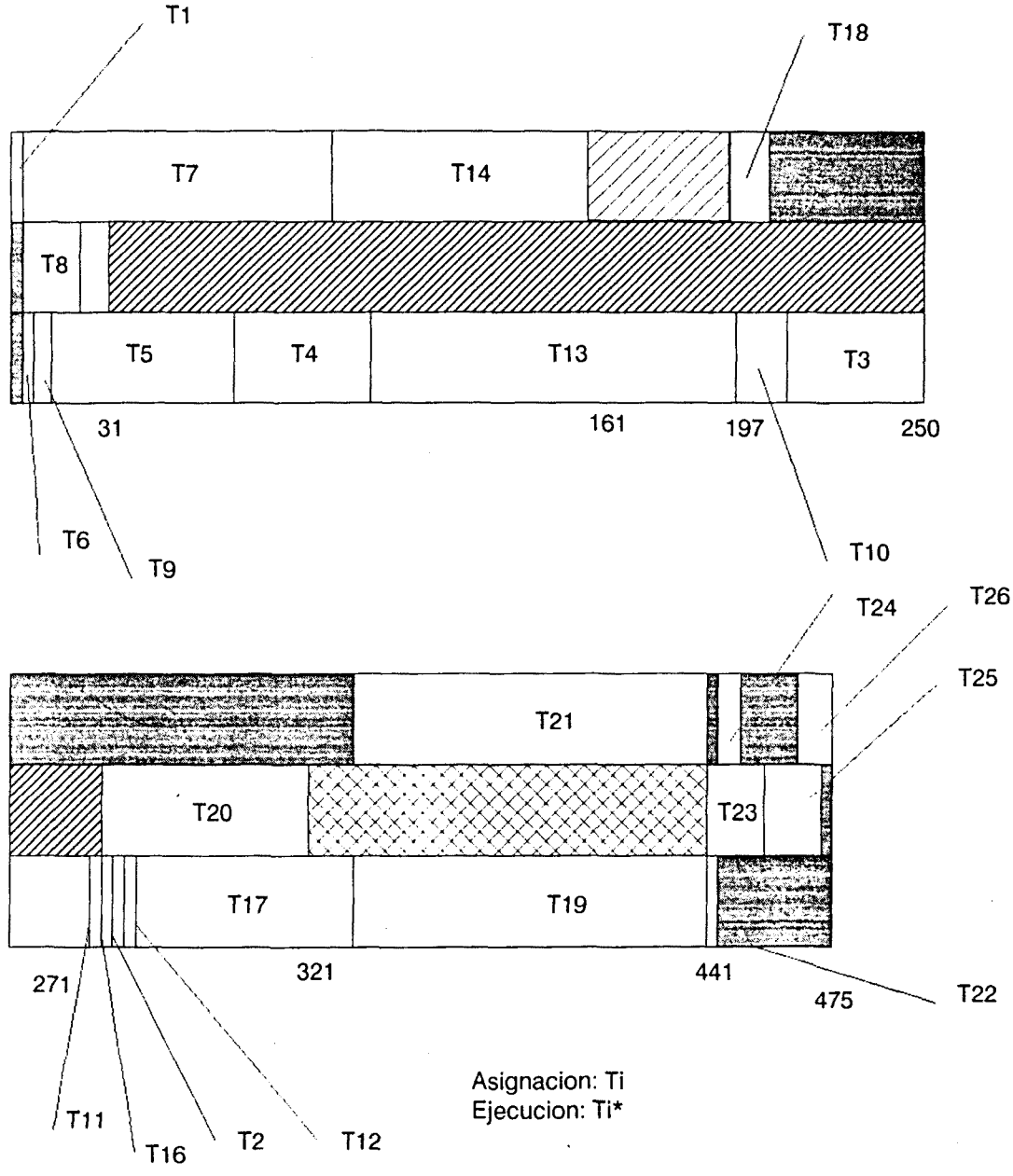


Figura 2.6

c) Se eliminan algunas relaciones de precedencia

El grafo de tareas que nos servirá de ejemplo, aparece en la figura 2.7, la longitud del schedule óptimo que nos servirá de punto de referencia para realizar comparaciones aparece en la figura 2.8a. Observar que en las figuras 2.8b,c y la figura

Processor	Node	Time	Start_time	Finish_time	Ready tasks
1	1	1	0	1	7(1)
1	7	80	1	81	14(81)
1	14	80	81	161	
1	idle	36	161	197	18(197)
1	18	20	197	217	
1	idle	124	217	341	21(341)
1	21	100	341	441	
1	idle	1	441	442	24(442)
1	24	10	442	452	
1	idle	20	452	472	26(472)
1	26	3	472	475	
2	idle	1	0	1	8(1)
2	8	20	1	21	15(21)
2	15	10	21	31	
2	idle	240	31	271	20(271)
2	20	50	271	321	
2	idle	120	321	441	23(441)
2	23	16	441	457	25(457)
2	25	15	457	472	
2	idle	3	472	475	
3	idle	1	0	1	[2,3,4,5,6,9,10,11](1)
3	6	1	1	2	[2,3,4,5,9,10,11](1)
3	9	5	2	7	[2,3,4,5,10,11](1)
3	5	60	7	67	[2,3,4,10,11](1)
3	4	30	67	97	[2,3,10,11](1);13(97)
3	13	100	97	197	[2,3,10,11](1);19(161)
3	10	30	197	227	[2,3,11](1);19(161)
3	3	40	227	267	[2,11](1);19(161)
3	11	1	267	268	2(1);19(161);16(268)
3	16	3	268	271	2(1);19(161)
3	2	6	271	277	19(161);12(277)
3	12	4	277	281	19(161);17(281)
3	17	60	281	341	19(161)
3	19	100	341	441	22(441)
3	22	1	441	442	
3	idle	33	442	475	

Asignación: Ti
Ejecución: Ti*

Tabla 2.4

2.9, el valor que intuitivamente esperamos decrezca (tiempo de ejecución del grafo) cambiando los parámetros que se especifican anteriormente ; a, b y c ; producen un incremento en el tiempo final de ejecución del grafo, lo que contradice nuestra intuición.

Como se demuestra anteriormente, es pues bastante dificil acertar en las prediciones de tiempos de ejecución final, intentando caracterizar grafos en base a número de precedencias, número de procesadores, etc. Los resultados experimentales serán por tanto, los que dirijan nuestra metodología de actuación tanto a la hora de considerar las distintas funciones a la hora de ordenar la lista de prioridades, así como las distintas elecciones que posteriormente deberemos realizar al confeccionar nuestros algoritmos de planificación .

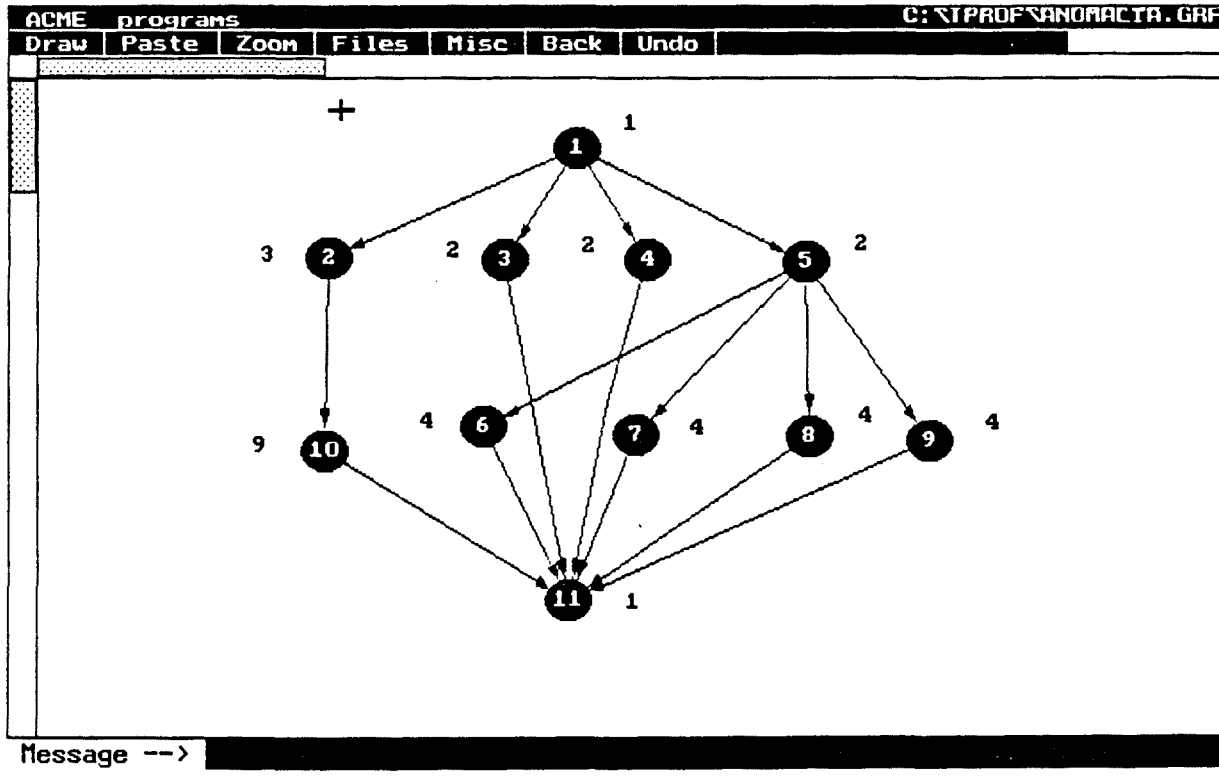
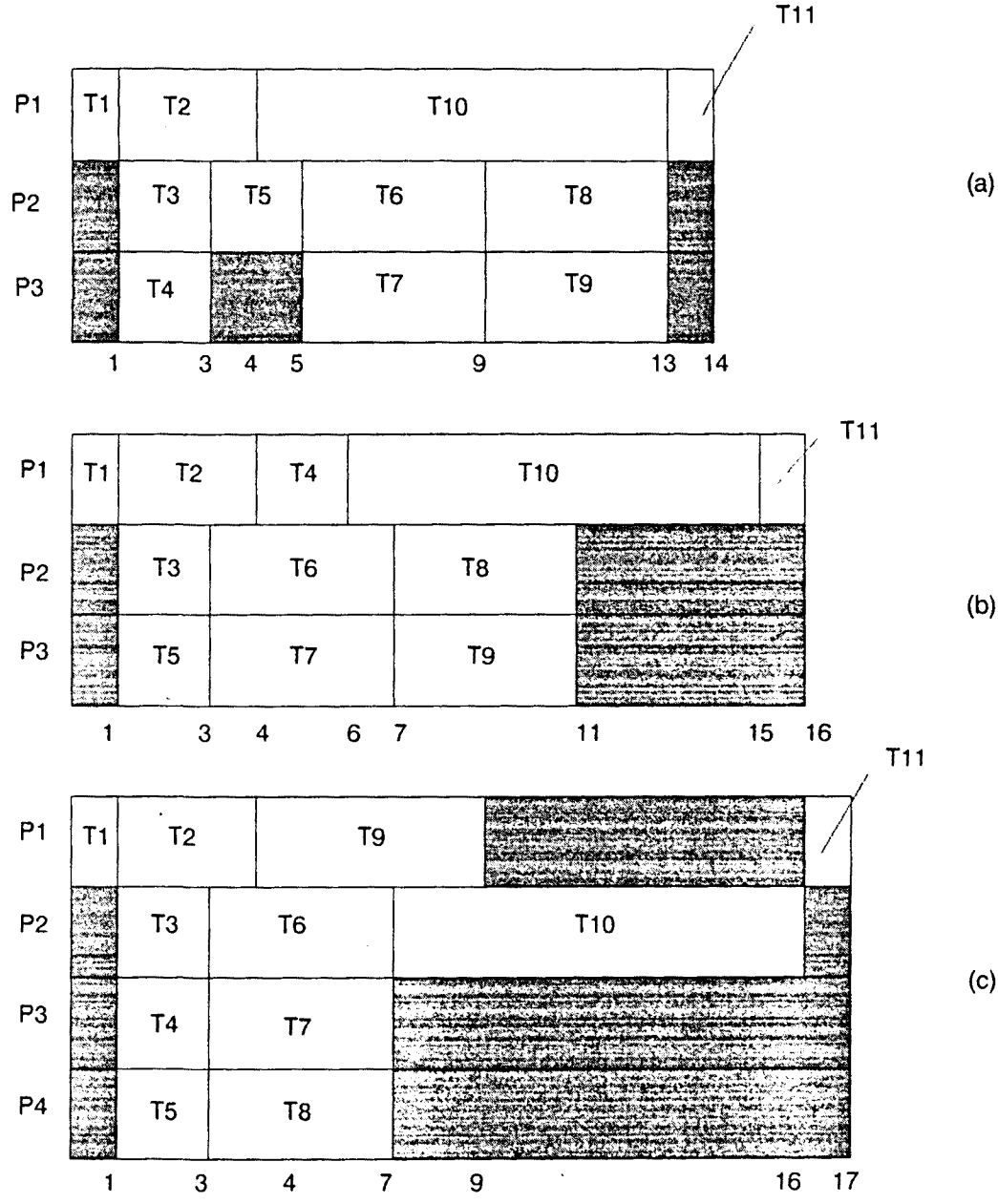


Figura 2.7

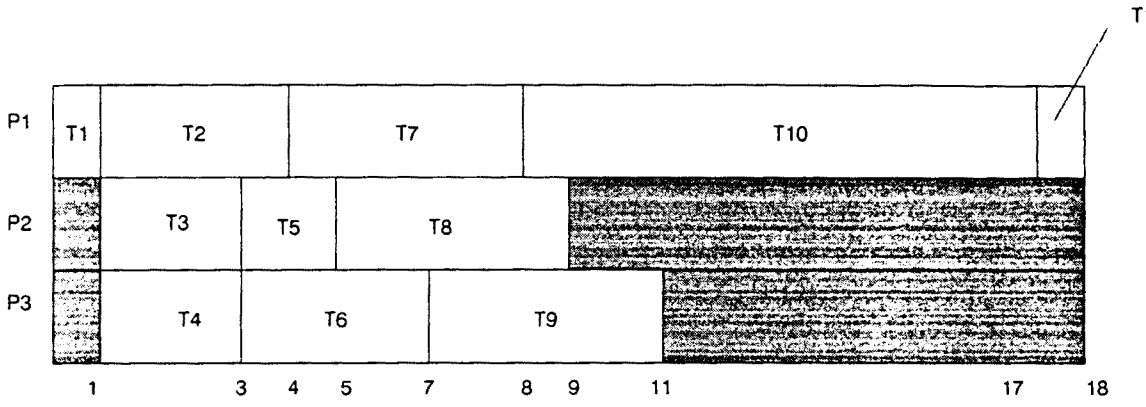


(a).- Tiempo optimo, L = (1,2,3,4,5,6,7,8,9,10,11)

(b).- Cambio de lista, L = (1,2,3,5,6,7,4,10,8,9,10,11)

(c).- Cambio del numero de procesadores

Figura 2.8



Mantener lista L = (1.2.3.4.5.6.7.8.9.10.11)

Eliminadas las relaciones de precedencia (nodo 5 al nodo 6), (nodo 5 al nodo 7)

Figura 2.9

Conclusiones del capítulo

El cambio que se produce en el tiempo de ejecución de un algoritmo (DAG) cuando comparamos:

a) El tiempo obtenido cuando se calcula la asignación para los tiempos Ti y se calcula la ejecución para los mismos tiempos Ti (caso que representa una predicción exacta de los tiempos de ejecución) y b) El tiempo obtenido cuando se calcula la asignación para los tiempos Ti y se realiza la ejecución para los tiempos Ti* (desviación entre el tiempo inicial considerado Ti y el real en ejecución Ti*) precisa de la concepción de políticas de "scheduling" que permitan reducir la desviación cuando los tiempos en ejecución se apartan de aquellos que fueron usados para aplicar una política de "scheduling" determinada.

CAPITULO 3.FUNDAMENTOS DEL METODO PROPUESTO

- Introducción

Una vez analizado el problema de la sensibilidad de los planificadores de lista frente a los datos, nos proponemos la búsqueda de algoritmos que tengan en cuenta en su formulación la posibilidad de que en ciertos nodos puedan variar sus tiempos de ejecución. Nosotros proponemos un método que permite reducir los tiempos muertos de los distintos procesadores duplicando tareas listas pertenecientes a procesadores ocupados, a fin de maximimar la utilización de los procesadores y obtener mejores tiempos de ejecución de las aplicaciones. Con objeto de abordar el problema de sensibilidad planteado en el capítulo II, desarrollaremos dos algoritmos de diferentes complejidades, basados en diferentes criterios y los evaluaremos tomando como índice de rendimiento el tiempo de ejecución del grafo problema cuando algunos de los nodos pueden tomar dos posibles valores en ejecución. Igualmente presentaremos las complejidades de los algoritmos desarrollados, y las compararemos con la complejidad real con objeto de mostrar la viabilidad en cuanto a la posible aplicación de los mismos.

Es interesante resaltar, que la tecnica de la duplicación de tareas ha sido utilizada por diversos autores y así viene reflejado en la literatura [39,40,41,42]; no obstante la finalidad de esas duplicaciones es totalmente distinta de la que se muestra en este trabajo, ya que las tareas duplicadas son ejecutadas además de la original (en el trabajo que presentamos, cada tarea se ejecuta una sola vez), es decir, el aumento

de cómputo global se utiliza para reducir las comunicaciones. Así por ejemplo, los trabajos de Lewis y Kruatrachue [39] se basan en la resolución a través de la aplicación de algoritmos heurísticos del problema del max / min; el objetivo que se pretende es encontrar una solución de compromiso entre balancear la carga de la aplicación (con objeto de maximizar el paralelismo), y minimizar al mismo tiempo las esperas debidas a las comunicaciones entre tareas. Para conseguir esto, almacenan y ejecutan tareas duplicadas en distintos procesadores para así eliminar la necesidad de comunicar a través de la red de interconexión que une el sistema multiprocesador. Un método similar es utilizado por Colin y Chretienne [42], los cuales afirman que en determinadas situaciones es mucho mejor repetir los mismos cálculos en varios procesadores que realizarlos en uno y enviar los resultados a los demás. Markenscoff y Liaw [41], en sus trabajos relacionados con la asignación de tareas en sistemas distribuidos, consideran como función objetivo la minimización del tiempo de respuesta del sistema; a tal efecto ellos proponen no tener comunicación entre procesadores, para ello, si un nodo es asignado a un procesador todos sus inmediatos sucesores deberán ser asignados al mismo procesador; a partir de esta premisa, obtienen un conjunto de procesos independientes donde la misma tarea está asignada a más de un proceso y por tanto a más de un procesador.

3.1.-Método para resolver el problema de la sensibilidad

A partir del ejemplo que hemos empezado a desarrollar para el grafo de tareas de la figura 2.4, pretendemos mostrar de forma detallada, en función de los valores obtenidos para los algoritmos de planificación basados en listas, la variación en el tiempo de ejecución de la aplicación representada en el grafo de tareas, cuando consideramos la asignación proporcionada para un determinado conjunto de posibles valores de los nodos que pueden variar su tiempo de ejecución, y ejecutamos la asignación de tareas para otros valores de tiempo de ejecución de los nodos.

Consideremos que los tiempos de algunos nodos pueden tomar dos valores alternativos $\{Ti, Ti^*\}$, nuestro objetivo será obtener una asignación que proporcione

unos buenos resultados tanto si en la ejecución del grafo, el nodo en cuestión toma el valor Ti o el valor Ti*.

Si analizásemos la traza de Gantt obtenida cuando mantenemos la asignación inicial pero consideramos los tiempos modificados, Tabla 3.0 (igual a la Tabla 2.4 del capítulo II, observaríamos que hay algunos espacios de tiempo donde tenemos procesadores ociosos mientras otros procesadores tienen tareas listas para su ejecución (huecos). En la traza de Gantt de la Tabla 3.0 por ejemplo, en "tiempo = 31 hasta tiempo = 271" el procesador 2 está ocioso mientras el procesador 3 tiene varias tareas listas para ser ejecutadas. En tiempo = 31, el procesador 3 tiene 5 tareas listas (2,3,4,10,11) las cuales deben esperar su ejecución porque el procesador al que han sido asignadas, procesador 3, está ejecutando la tarea 5. Por tanto sería deseable que el procesador 2 ejecutase alguna de estas tareas para reducir el tiempo de ejecución del grafo.

La misma situación aparece varias veces en la traza de Gantt mostrada en la Tabla 3.0, (el procesador 1 está ocioso desde 161 hasta 197 existiendo tareas listas 2,3,10 y 13 en el procesador 3; y el procesador 2 está también ocioso desde 321 hasta 441 existiendo tareas listas, 19, en el procesador 3).

El ejemplo presentado muestra la aparición simultanea de dos fenómenos:

- La existencia de procesadores libres sin tareas listas para su ejecución.

- La existencia de procesadores con varias tareas listas simultaneamente, que implica la secuencialización en la ejecución de las mismas.

El método propuesto se basa en la detección y análisis de ambos fenómenos y su reducción mediante una adecuada duplicación de tareas.

Con objeto de mostrar los resultados proporcionadas por los métodos de duplicación de tareas que se han desarrollado, y antes de pasar a su descripción, vamos

Proc.	Nodo	T_Ejec.	T_inic.	T_final	Tareas Listas
1	1	1	0	1	7(1)
1	7	80	1	81	14(81)
1	14	80	81	161	
1	idle	36	161	197	18(197)
1	18	20	197	217	
1	idle	124	217	341	21(341)
1	21	100	341	441	
1	idle	1	441	442	24(442)
1	24	10	442	452	
1	idle	20	452	472	26(472)
1	26	3	472	475	
2	idle	1	0	1	8(1)
2	8	20	1	21	15(21)
2	15	10	21	31	
2	idle	240	31	271	20(271)
2	20	50	271	321	
2	idle	120	321	441	23(441)
2	23	16	441	457	25(457)
2	25	15	457	472	
2	idle	3	472	475	
3	idle	1	0	1	[2,3,4,5,6,9,10,11](1)
3	6	1	1	2	[2,3,4,5,9,10,11](1)
3	9	5	2	7	[2,3,4,5,10,11](1)
3	5	60	7	67	[2,3,4,10,11](1)
3	4	30	67	97	[2,3,10,11](1);13(97)
3	13	100	97	197	[2,3,10,11](1);19(161)
3	10	30	197	227	[2,3,11](1);19(161)
3	3	40	227	267	[2,11](1);19(161)
3	11	1	267	268	2(1);19(161);16(268)
3	16	3	268	271	2(1);19(161)
3	2	6	271	277	19(161);12(277)
3	12	4	277	281	19(161);17(281)
3	17	60	281	341	19(161)
3	19	100	341	441	22(441)
3	22	1	441	442	
3	idle	33	442	475	

Tabla 3.0

a mostrar sobre el ejemplo de la Tabla 3.0, los resultados que se obtendrían una vez evaluadas las tareas a duplicar e incluidas en la asignación inicial.

En la Tabla 3.1bis se muestra la traza de Gantt obtenida considerando la asignación de la Tabla 3.1 (tiempos de ejecución Ti), a la que se le han añadido las tareas duplicadas evaluadas mediante los algoritmo propuestos. Según podemos observar en la tabla 3.1bis (Ti en planificación con duplicaciones, Ti* en ejecución), el tiempo de ejecución del grafo es ahora 344. Si comparamos este resultado con el obtenido anteriormente (Ti en planificación, Ti* en ejecución sin duplicaciones) que era 475 (Tabla 3.0), observamos una importante reducción en el tiempo de ejecución obtenida a partir de la aplicación de la duplicación de tareas.

Además debe señalarse que si esta nueva asignación con duplicaciones es mantenida (Tabla 3.1bis) pero ahora ejecutasemos con los tiempos (Ti), el resultado que obtendríamos sería exactamente el mismo que fue obtenido cuando consideramos la primera asignación (Tabla 2.1c).

Así, cuando ejecutemos el grafo para la asignación con duplicaciones, los resultados que obtenemos son satisfactorios tanto para el caso de considerar tanto los tiempos (Ti) como para los tiempos (Ti*).

Procesador	Tareas asignadas
P1	1, 7, 14, 18, 2, 21,24, 26
P2	8, 15, 4, 10, 3, 11, 16, 20, 19, 23, 25
P3	6, 9, 5, 4, 13, 10, 3, 11, 16, 2, 12, 17, 19, 22

Las tareas subrayadas son las tareas duplicadas

Tabla 3.1

Es importante notar que utilizando esta metodología de actuación, cada vez que se duplica una tarea, los huecos que aparecen, no son los que aparecían antes de la duplicación, sino que van apareciendo dinamicamente a medida que avanza el proceso de simulación.

Proc.	Nodo	T_Ejec.	T_inic.	T_final	Tareas Listas
1	1	1	0	1	[2,7](1)
1	7	80	1	81	2(1);14(81)
1	14	80	81	161	2(1)
1	2	6	161	167	18(167)
1	18	20	167	187	
1	libre	44	187	231	21(231)
1	21	100	231	331	24(331)
1	24	10	331	341	26(341)
1	26	3	341	344	
2	libre	1	0	1	[3,4,8,10,11](1)
2	8	20	1	21	[3,4,10,11](1);15(21)
2	15	10	21	31	[3,4,10,11](1)
2	4	30	31	61	[3,10,11](1)
2	10	30	61	91	[3,11](1)
2	3	40	91	131	11(1)
2	11	1	131	132	16(132)
2	16	3	132	135	20(135)
2	20	50	135	185	19(161)
2	19	100	185	285	23(285)
2	23	16	285	301	25(301)
2	25	15	301	316	
2	libre	28	316	344	
3	libre	1	0	1	[2,3,4,5,6,9,10,11](1)
3	6	1	1	2	[2,3,4,5,9,10,11](1)
3	9	5	2	7	[2,3,4,5,10,11](1)
3	5	60	7	67	[2,3,11](1);13(67)
3	13	100	67	167	19(161);12(167)
3	12	4	167	171	19(161);17(171)
3	17	60	171	231	
3	libre	54	231	285	22(285)
3	22	1	286	286	
3	libre	58	286	344	

Asignación: Tiempo de ejecución inicial (Ti) con duplicaciones

Ejecución: Nuevos tiempos (Ti*)

Tabla 3.1bis

3.2.- Algoritmos desarrollados

Con objeto de seleccionar el conjunto de tareas que van a ser duplicadas, y los procesadores donde van a ser asignadas, hemos desarrollado dos algoritmos de naturaleza heurística güiados por diferentes criterios para la identificación de las tareas a duplicar y de diferentes complejidades. El primero ellos, basado en el camino critico, donde el factor de segundo orden es la tarea con un mayor número de sucesores, y además incluimos duplicación de tareas (CP/MISF/TD) [45]; el algoritmo considera una visión global del grafo del programa analizando el conjunto de huecos y posibles tareas susceptibles de llenar el hueco, para cada duplicación se calcula el tiempo de ejecución de la aplicación con la tarea duplicada con objeto de comprobar, si la duplicación mejora el tiempo final de la aplicación, el proceso continua intentando duplicar tareas una a una y realizando las comprobaciones pertinentes con objeto de incorporar la tarea duplicada o deshechar la duplicación de la tarea simulada. El límite superior teórico de complejidad de este algoritmo es de O (n^3m).

El segundo algoritmo que se presentará está basado también en la idea de camino crítico y de duplicación, pero de complejidad reducida (CP/MISF/TD/RC). La modificación sustancial con respecto al anterior consiste en el mecanismo de identificar las duplicaciones, en el algoritmo anterior utilizamos la idea de hueco (procesador libre), para duplicar una tarea, en este algoritmo no existe ya la idea de hueco como tal sino la idea de primer instante en el cual un procesador está libre y por tanto es susceptible de comenzar a ejecutar una duplicación. Al igual que en el algoritmo anterior, comenzamos por la primera asignación proporcionada por el CP/MISF con los tiempos de ejecución (Ti), y simulamos la ejecución del grafo para los tiempos de ejecución (Ti*); en este proceso, el primer instante de tiempo en el cual se detecta que un procesador está desocupado, la tarea lista con mayor nivel asignada a otro procesador, es duplicada en el procesador desocupado y continua el proceso de simulación. Si no hay más tareas listas a ser duplicadas, el proceso de simulación avanza, buscando nuevos instantes donde se vuelve a observar la situación

anterior con objeto de duplicar otras tareas. Este proceso continua hasta que la ejecución del grafo en su totalidad ha sido simulada.

Es importante notar que las variaciones que se producen en las trazas de Gantt debido al algoritmo de duplicación son muy sofisticadas y dificil de conocer lo que sucederá cuando se duplica una tarea. Tanto es así que puede suceder que el hecho de realizar una duplicación podemos obtener peores tiempos que si no la hubiesemos realizado, pero es posible que si podemos duplicar otra tarea, el resultado que obtengamos manteniendo la anterior sea mejor que el original, ver para ello [49]. Por tanto, cuando observamos que ninguna tarea puede ser ya duplicada, deberemos recuperar la primera asignación que proporcione el mejor tiempo de ejecución.

Otra consecuencia que se desprende del método de duplicación es la tendencia a reducir el número de procesadores que se necesitan para que el tiempo de ejecución del mismo no supere el tiempo del camino crítico. En el ejemplo que se describe, para el grafo representado en la figura 3.1 el número mínimo de procesadores que necesitamos aplicando las cotas de Fernández y Bussell [17] es de cuatro, si el ejemplo

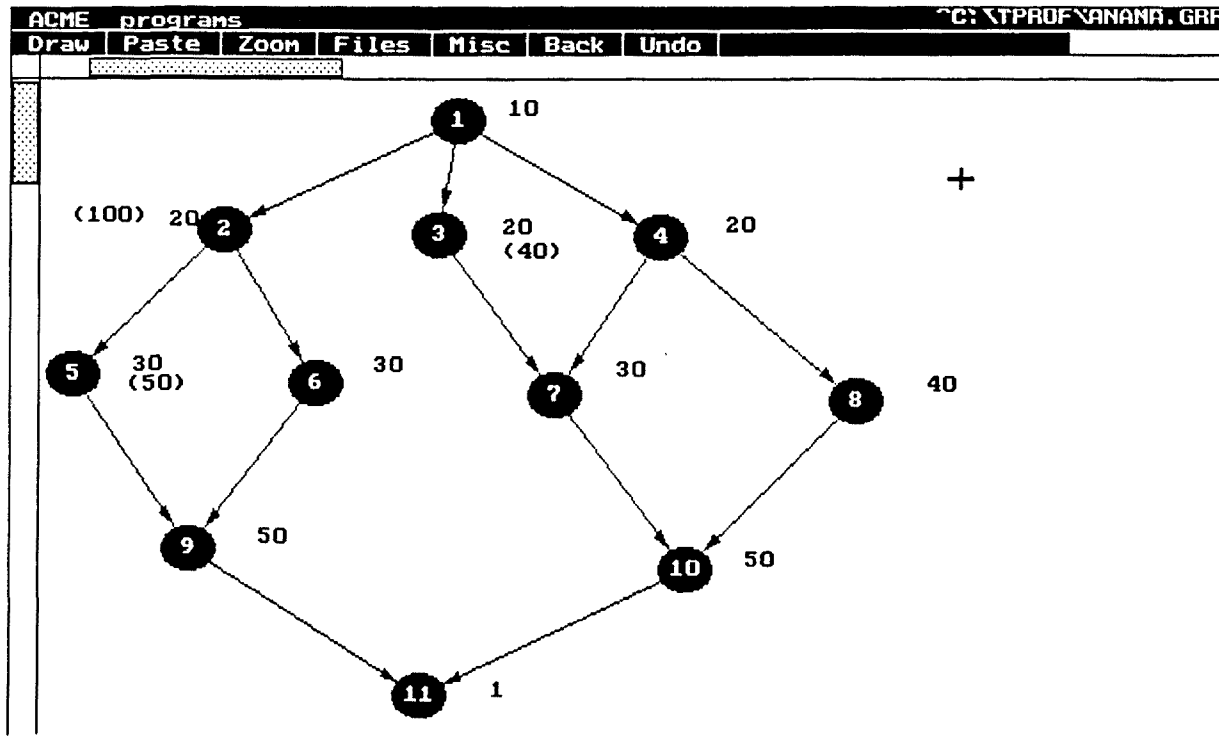


Figura 3.1

de la figura 3.1, lo desarrollamos para cuatro procesadores podríamos observar que el algoritmo de duplicación no proporciona la posibilidad de realizar ninguna duplicación, y el tiempo de ejecución que se obtiene cuando mantenemos la asignación obtenida para los tiempos de ejecución (Ti) y ejecutamos el grafo con los tiempos (Ti*), el tiempo final que obtenemos es de 211 unidades, el mismo valor que obtenemos para un sistema de tres procesadores y una duplicación.

3.2.1.- Descripción del algoritmo CP/MISF/TD

En este apartado describimos el algoritmo desarrollado para evaluar las tareas a ser duplicadas y los procesadores donde deberán ser asignadas, y cuyo diagrama de flujo se muestra en la figura 3.2. La significación de las variables utilizadas se proporcionará a continuación cuando se describan los pasos del algoritmo. Vamos a describir el funcionamiento del algoritmo propuesto, estructurándolo en una serie de pasos cuya correspondencia con la figura 3.2 y figura 3.2bis se puede establecer sobre la base de las operaciones a realizar:

Paso 1: Tomemos la asignación inicial proporcionada por el CP/MISF {Asig_init} considerando los tiempos de ejecución (Ti); y ejecutar considerando los tiempos (Ti*) obteniendo la traza de Gantt. Denominamos { T_ini t} al tiempo de ejecución correspondiente a la asignación inicial, { Asig_MIN } y { T_MIN } serán respectivamente la asignación y el tiempo de ejecución a encontrar para la asignación que proporcione el tiempo de ejecución mínimo, y { Asig_W } y { T_W } las variables intermedias de trabajo.

Paso 2: Sobre la traza de Gantt obtenida, buscar las tareas candidatas a ser duplicadas (llamemos W a la tarea candidata). Buscar el primer hueco (procesador inactivo) en la traza de Gantt obtenida y comprobar si existen tareas listas en otros procesadores y cuyo tiempo de ejecución sea menor o igual que el tamaño del hueco, a)si hay varias tareas candidatas, se elige como W aquella que tenga un mayor nivel, se duplica y vamos al paso tres; b) si no, buscar en el intervalo de tiempo que dura el hueco, el primer instante en el cual existe/n tarea/s lista/s cuyo/s tiempo/s de ejecución

sea/n menore/s o igual/es que la duración del resto del hueco, c) si hay varias tareas candidatas, se elige como W aquella que tenga un mayor nivel, en el caso de que sólo exista una tarea, esta será la tarea candidata { W }, se duplica y vamos al paso tres; si no, d) si no hay tareas a duplicar recuperar la asignación final con su correspondiente tiempo de ejecución y finalizar el algoritmo.

Paso 3: Modificar la asignación (actualizar), cargando la tarea { W }, es importante notar la nueva asignación { Asig_W }.

Paso 4: Simular la ejecución con objeto de generar el nuevo tiempo de ejecución { T_W } y la nueva traza de Gantt.

Paso 5: Si el nuevo tiempo de ejecución { T_W } es menor que que el tiempo de ejecución global { T_MIN } cambiar la asignación y añadir a { Asig_MIN } la nueva modificación alterando tambien el tiempo global de ejecución { T_MIN } e ir al paso 2, sino mirar si hay otras duplicaciones (considerando la nueva duplicación pero sin añadirla en la asignación global { Asig_MIN }) que puede mejorar el tiempo de ejecución global { T_MIN }.

Con objeto de mostrar el funcionamiento del algoritmo desarrollaremos los pasos del mismo en base al grafo de la figura 3.1, que nos servirá como ejemplo.

A partir de la asignación proporcionada por el CP/MISF cuya traza de Gantt es mostrada en la figura 3.3.a , simulamos la ejecución del grafo considerando los tiempos modificados (Ti*). La nueva traza de Gantt para los nuevos valores se presenta en la figura 3.3.b. Cuando el primer hueco (procesador ocioso) es encontrado (procesador 3 de 50 a 110), realizamos un intento para duplicar una tarea lista asignada a otro procesador en ese hueco (tarea 7 asignada al procesador 2, ver figura 3.3bis. Por tanto lo primero que mira el algoritmo es si hay tareas listas que comiencen cuando lo hace el hueco y su tiempo de ejecución no exceda de la longuitud del hueco (tarea 7 tiempo 50). Si hay varias tareas que satisfacen las condiciones anteriores, la tarea candidata es aquella que tiene un mayor nivel. Si no existe ninguna tarea que

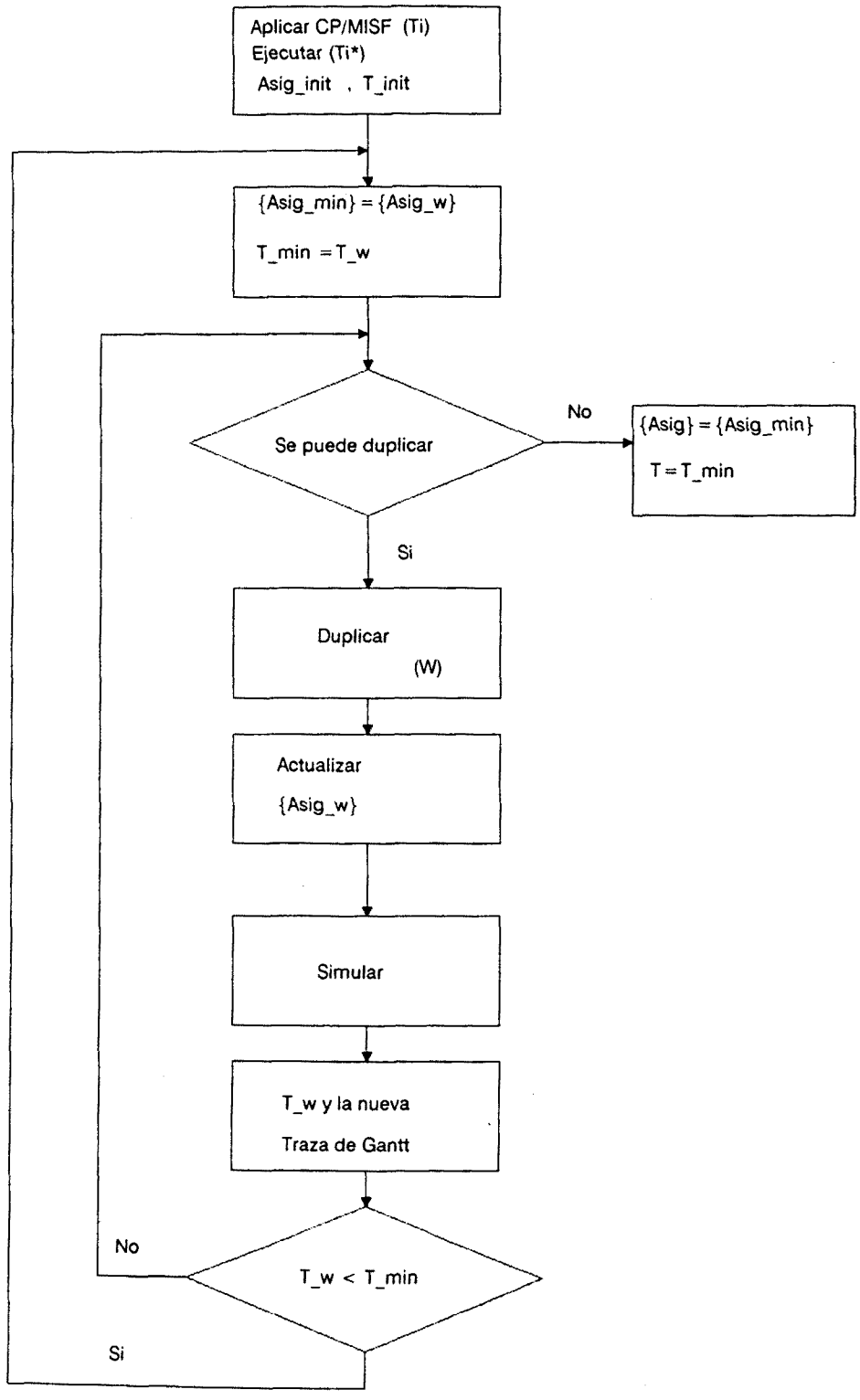

Aplicar CP/MISF (Ti)
Ejecutar (Ti*)
Asig_init , T_init
{Asig_min} = {Asig_w}
T_min = T_w
Se puede duplicar
No
{Asig} = {Asig_min}
T = T_min
Si
Duplicar
(W)
Actualizar
{Asig_w}
Simular
T_w y la nueva
Traza de Gantt
No
T_w < T_min
Si

Figura 3.2

```
BEGIN
    Llamar al procedimiento de asignacion CP/MISF para (Ti);
    asig_min: = asig;
    Simular la ejecucion para (Ti*);
    time_min: = time;
    Min: = Maxint;
    REPEAT
        Buscar el primer empty_slot;
        REPEAT
            level: = 0;
            FOR cada nodo j DO
            BEGIN
              IF nodo[i] cabe en empty_slot[i] THEN
              BEGIN
                IF nodo[j].ready < empty_slot[i].start
                THEN
                  Start: = empty_slot[i].start
                ELSE
                  Start: = nodo[j].ready;
                IF (Start < Min) or
                   (Start = Min) and
                   (nodo[j].level > level) THEN
                BEGIN
                   Min: = Start;
                   candidate: = j;
                   processor: = empty_slot[i].proc;
                   level: = nodo[j].level;
                END;
              END;
            END;
            IF no se han duplicado tareas THEN
               siguiente empty_slot;
          UNTIL se haya duplicado alguna tarea OR
                no hay mas empty_slot;
          IF algun nodo ha sido duplicado THEN
          BEGIN
            Asignar candidate a processor;
            Simular;
            IF time < time_min THEN
            BEGIN
              time_min: = time;
              asig_min: = asig;
            END;
      END;
    UNTIL que ningun nodo ha sido duplicado;
END.
```

asig: Asignacion evaluada
asig_min:Asignacion que ha proporcionado el minimo tiempo de ejecucióon
time_min: minimo tiempo de ejecucion obtenido
candidate: tarea que podria ser duplicada
procesador: procesador en el que la tarea duplicada sera asignada
empty_slot[i]:hueco en la traza de Gantt

Figura 3.2 bis

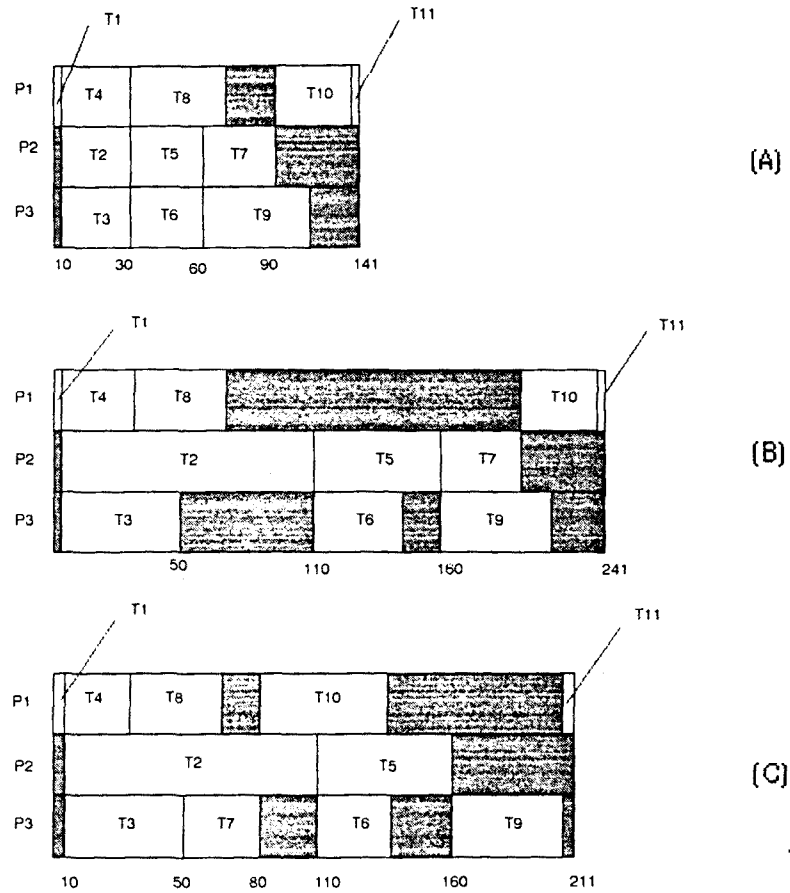


Figura 3.3

satisface las condiciones anteriores, la tarea candidata será aquella que primero este lista durante la duración del hueco. Si no duplicamos ninguna tarea pasaremos a analizar el siguiente hueco. En la figura 3.3bis, se muestra la tabla correspondiente a la traza de Gantt presentada en la figura 3.3b, donde se observa, la identificación de las tareas listas en cada procesador y que están esperando.

La figura 3.3c muestra la ejecución del grafo incluyendo la duplicación de la tarea 7, en el procesador 3. Como se observa, la inclusión de la ejecución de la tarea duplicada ha producido una disminución del tiempo de ejecución final del grafo de 30 unidades de tiempo.

Proc.	Nodo	T_Ejec.	T_inic.	T_final	Tareas Listas
1	1	10	0	10	4(10)
1	4	20	10	30	8(30)
1	8	40	30	70	
1	idle	120	70	190	10(190)
1	10	50	190	240	11(240)
1	11	1	240	241	
2	idle	10	0	10	2(10)
2	2	100	10	110	7(50),5(110)
2	5	50	110	160	7(50)
2	7	30	160	190	
2	idle	51	190	241	
3	idle	10	0	10	3(10)
3	3	40	10	50	
3	idle	60	50	110	6(110)
3	6	30	110	140	
3	idle	20	140	160	9(160)
3	9	50	160	210	
3	idle	31	210	241	

Figura 3.3bis

3.2.1.1 Análisis de la complejidad del CP/MISF/TD

Para evaluar la complejidad temporal en el caso más desfavorable, vamos a determinar el número de análisis que deben realizarse.

El máximo número de huecos que deben ser considerados es m*k, para k = Max {Ki}, donde i = 1 ..m, siendo m = número de procesadores y Ki el número de huecos en el procesador i. En el peor de los casos un procesador podría tener n huecos, donde n es el número de nodos del grafo. Ya que para cada hueco deberemos considerar todas las tareas (n), la complejidad temporal máxima que implica es $O(n^2m)$.

Una vez que el hueco ha sido llenado por una tarea duplicada, la asignación es diferente a la inicial y por tanto no tiene sentido intentar llenar el resto de los huecos porque la nueva asignación puede causar la generación de un nuevo conjunto de huecos, figura 3.b. Por tanto cuando una tarea ha sido duplicada, es necesario resimular la ejecución con esta asignación temporal, con objeto de eliminar duplicaciones innecesarias. Si el nuevo tiempo de ejecución es menor que el mínimo obtenido hasta ahora, salvamos el nuevo tiempo y la asignación; sino lo es, repetimos el proceso anterior hasta que no podamos duplicar ninguna tarea o bien el tiempo de ejecución es reducido por una duplicación, (esto implica mantener las duplicaciones hechas hasta la ejecución en curso.

Por tanto en el peor de los casos, el proceso debería ser repetido n veces. Esto implica que la cota superior para el valor de complejidad temporal sería $O(n^3m)$.

3.2.1.2 Estudio de la complejidad real

Uno de los aspectos importantes a considerar cuando se presentan métodos aproximados, para la resolución del problema del "scheduling" en sistemas multiprocesador, planteado este en el caso general, es el de disponer de alguna medida que nos permita conocer el tiempo de cálculo que necesitarán nuestros

algortimos en la búsqueda de la asignación de tareas a procesadores, así como del tiempo necesitado para la ejecución de la aplicación.

La mayor parte de las veces, la complejidad teórica, nos proporciona una cota temporal, valor por otro lado que puede estar muy distorsionado, ya que, raramente se duplicarán n tareas, o bien, el suponer m*n huecos, son cotas muy alejadas de lo que sucede en la realidad. A tal efecto hemos desarrollado un conjunto de medidas sobre aquel conjunto de rutinas más significativas desde el punto de vista de su tiempo de cálculo, de tal manera que nos van a permitir mostrar, la complejidad real obtenida para un conjunto de grafos de prueba.

A continuación se presentan en forma gráfica, los resultados medidos de las rutinas significativas para el conjunto de grafos que se muestra en la figura 3.4(a-e).

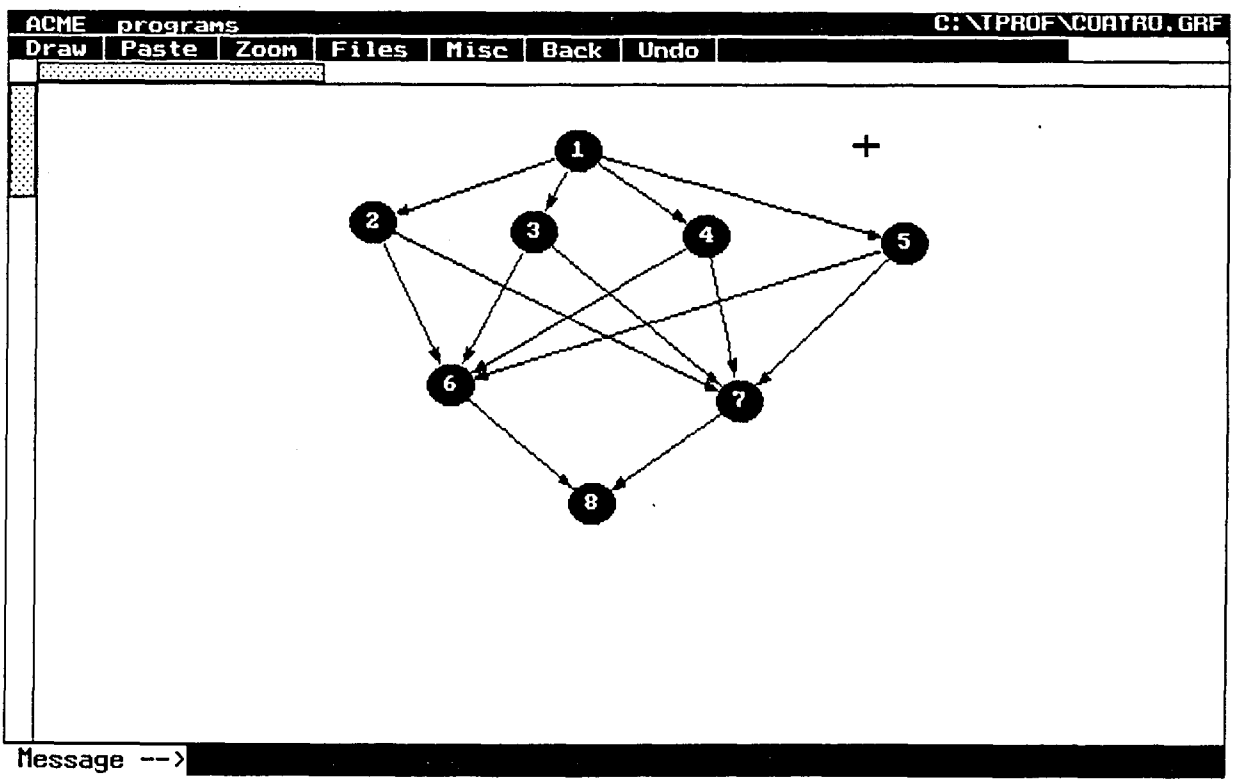


Figura 3.4a

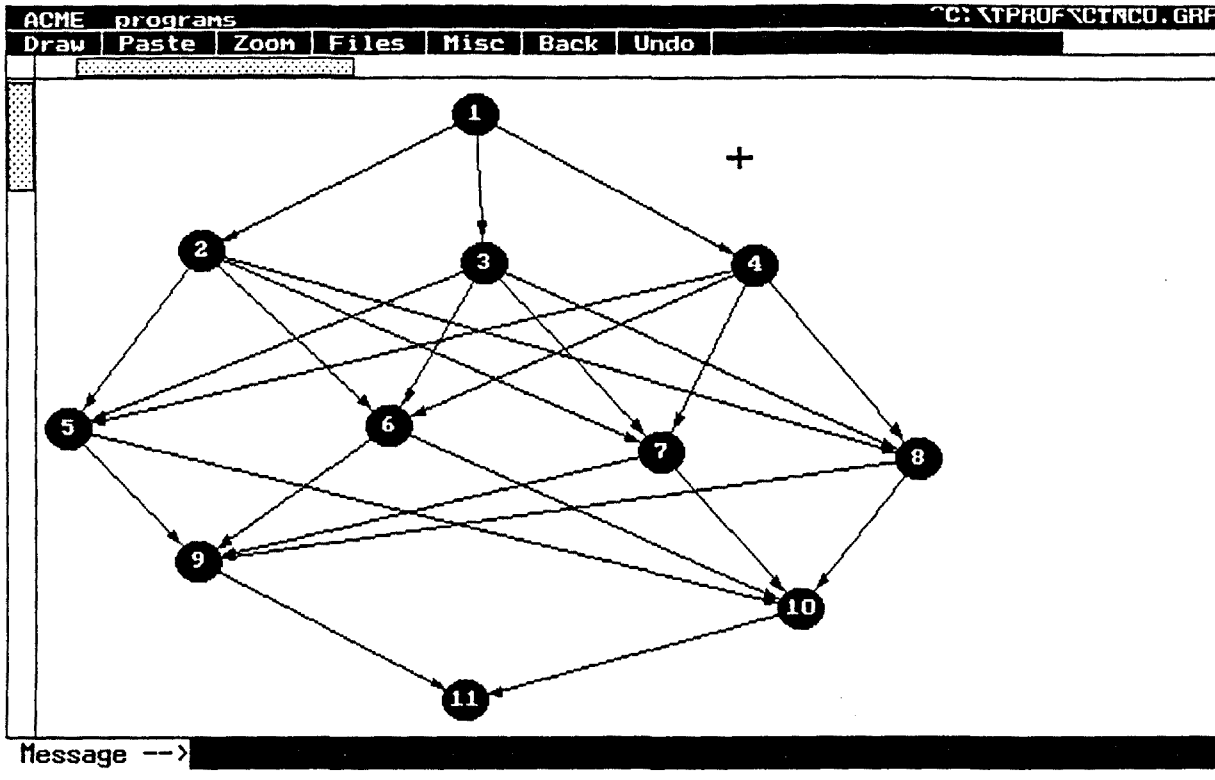


Figura 3.4b

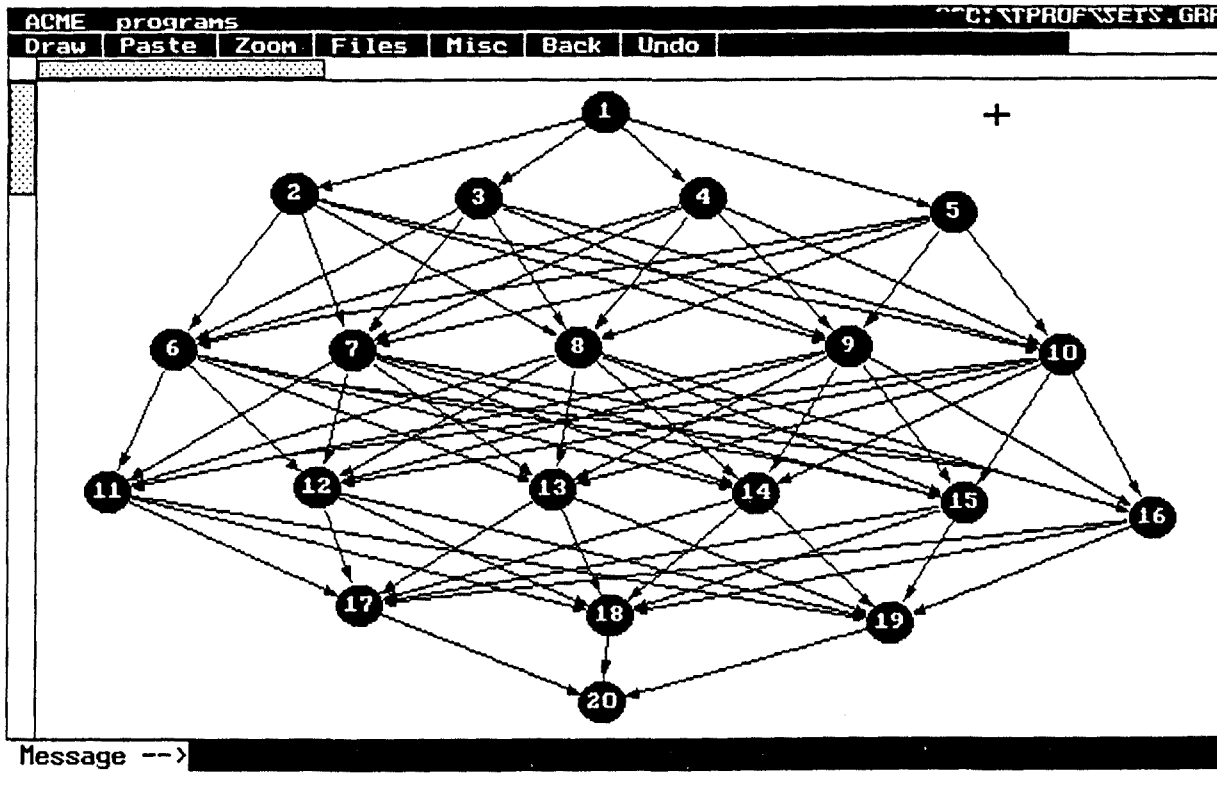


Figura 3.4c

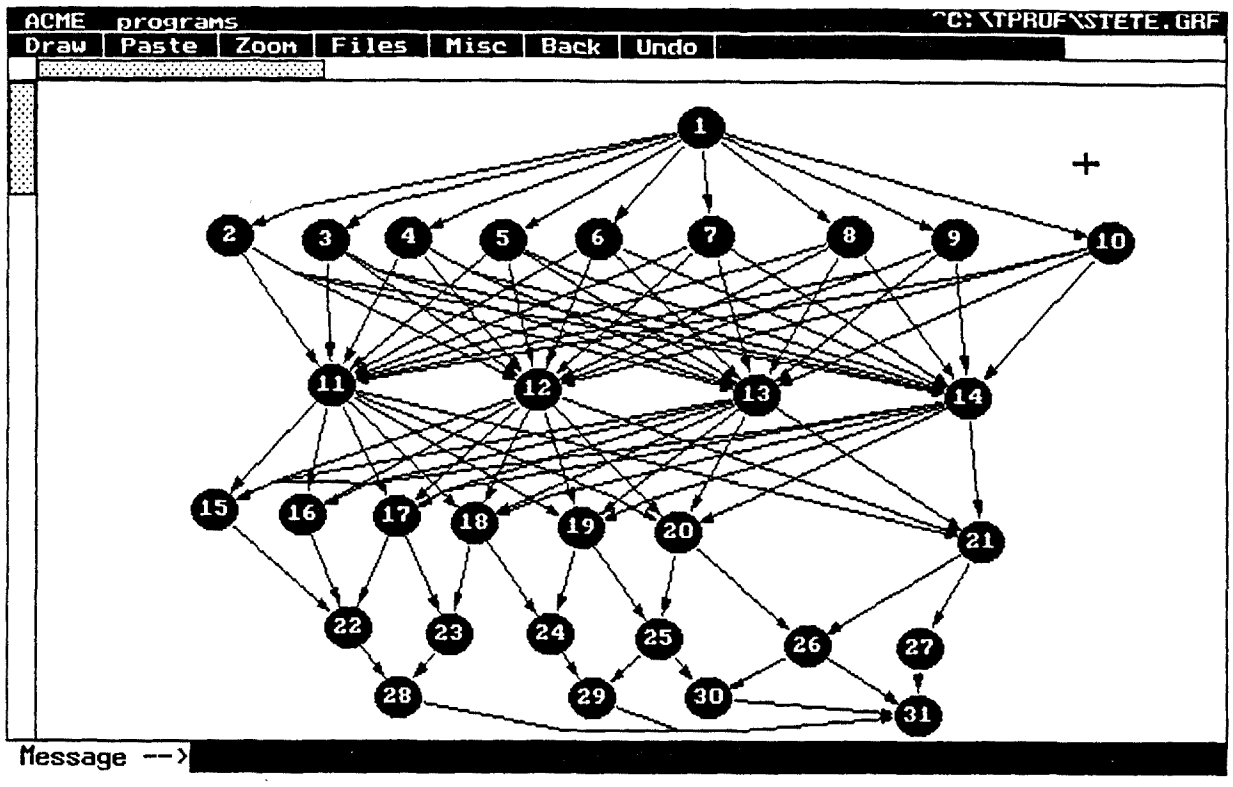


Figura 3.4d

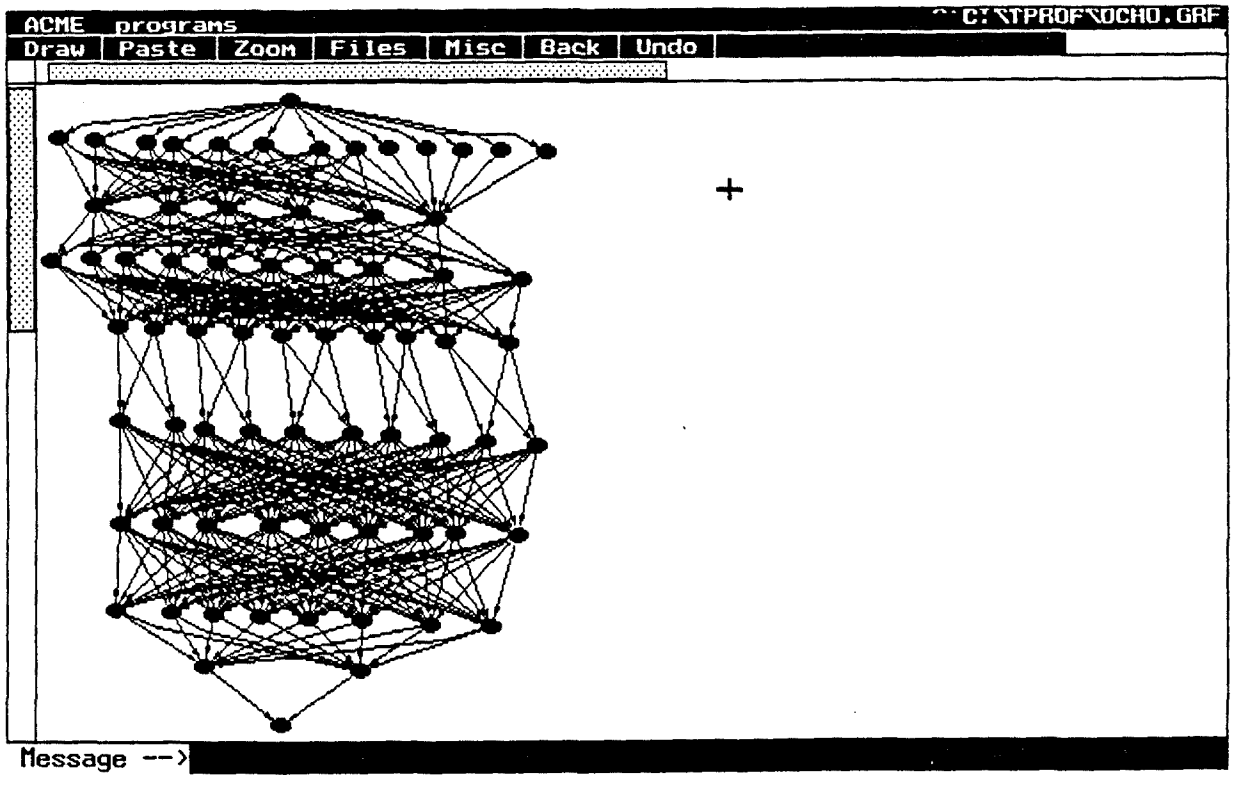


Figura 3.4e

Este conjunto lo integran cinco grafos, cuyo número ded nodos varía desde 8 hasta 70. En primer lugar pretendemos medir la influencia del número de nodos en la complejidad del algortimo. Una vez analizado este punto, mediremos la influencia de las duplicaciones en la complejidad temporal del algoritmo, a este fin, hemos desarrollado un ejemplo que relaciona para la misma topología de grafo, modificando los tiempos de los posibles nodos que pueden variar; el tiempo de ejecución entre las rutinas básicas del algoritmo de "scheduling", en función del número de duplicaciones.

La figura 3.5, muestra la representación gráfica de la complejidad temporal frente al número de nodos para los grafos que se muestran en la figura 3.4(a-e). Aproximadamente, para un número de duplicaciones que oscila entre, 2 y 8, puede observarse, que la complejidad real del algoritmo, no se aproxima a la complejidad teórica, siendo los valores de tiempo de ejecución que proporcionan las rutinas utilizadas para el cálculo de la complejidad real, del orden de $O(n^2)$, frente a la cota teórica máxima calculada $O(n^3*m)$.

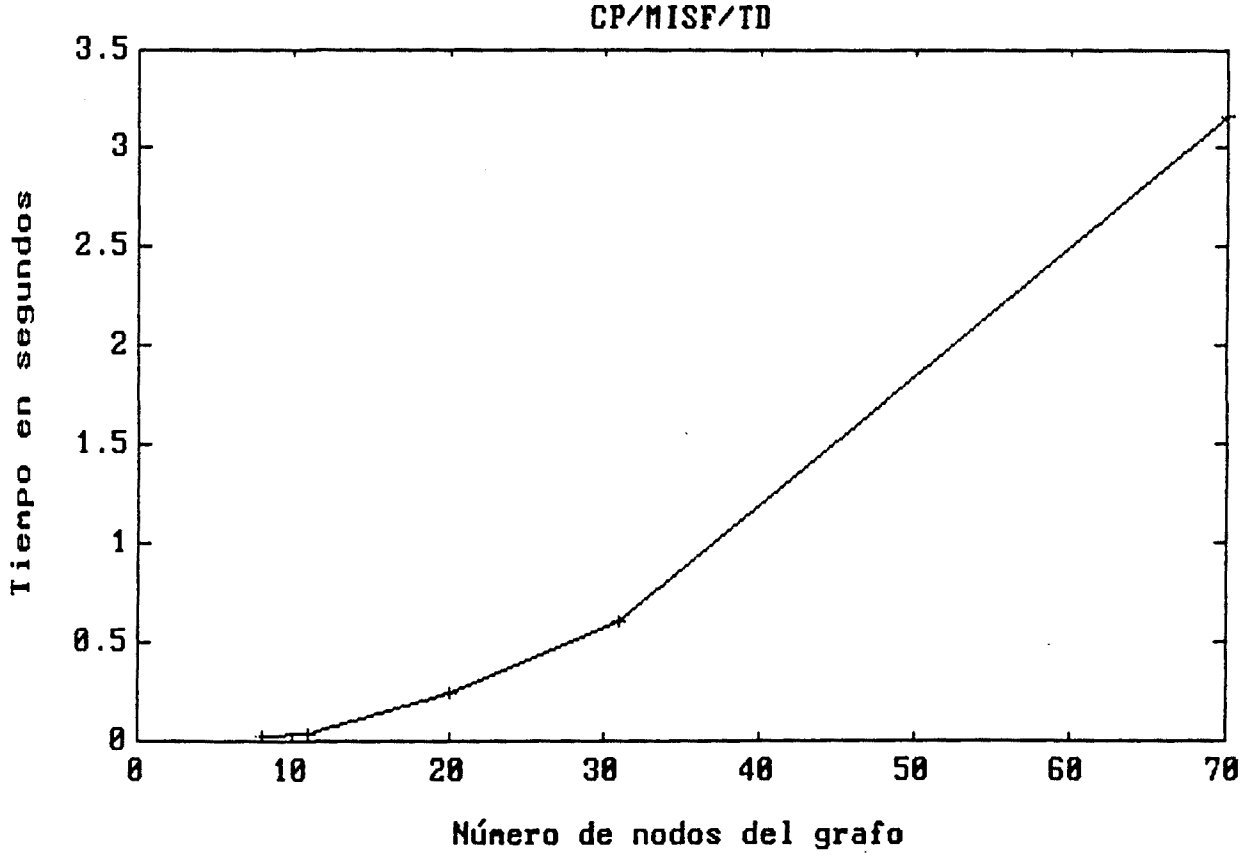


Figura 3.5

La figura 3.6 muestra la relación entre la complejidad temporal y el número de duplicaciones. Esta gráfica se ha obtenido para el grafo de la figura 3.7, manteniendo su topología y variando los tiempos de ejecución de ciertos nodos. Como se observa en la gráfica de la figura 3.6, fijado el número de nodos del grafo (70 en el caso que nos ocupa), la topología del mismo y variando el tiempo de ejecución asociado para cada de los nodos, el número de duplicaciones que se consiguen, tiene una importancia notable en el tiempo final proporcionado por las rutinas destinadas a medir la complejidad real del algoritmo. Como se muestra en la gráfica de la figura 3.6, la complejidad real del algoritmo, no se aproxima al valor máximo teórico calculado $O(n^3)$ como se dedujo anteriormente para el peor de los casos, sino a $K*(n^2*T)$, donde T es el número de tareas duplicadas y K es un factor constante que relaciona el número de huecos analizados frente al número de tareas que realmente se duplican; para los ejemplos presentados, este factor de relación K, es aproximadamente 2/5.

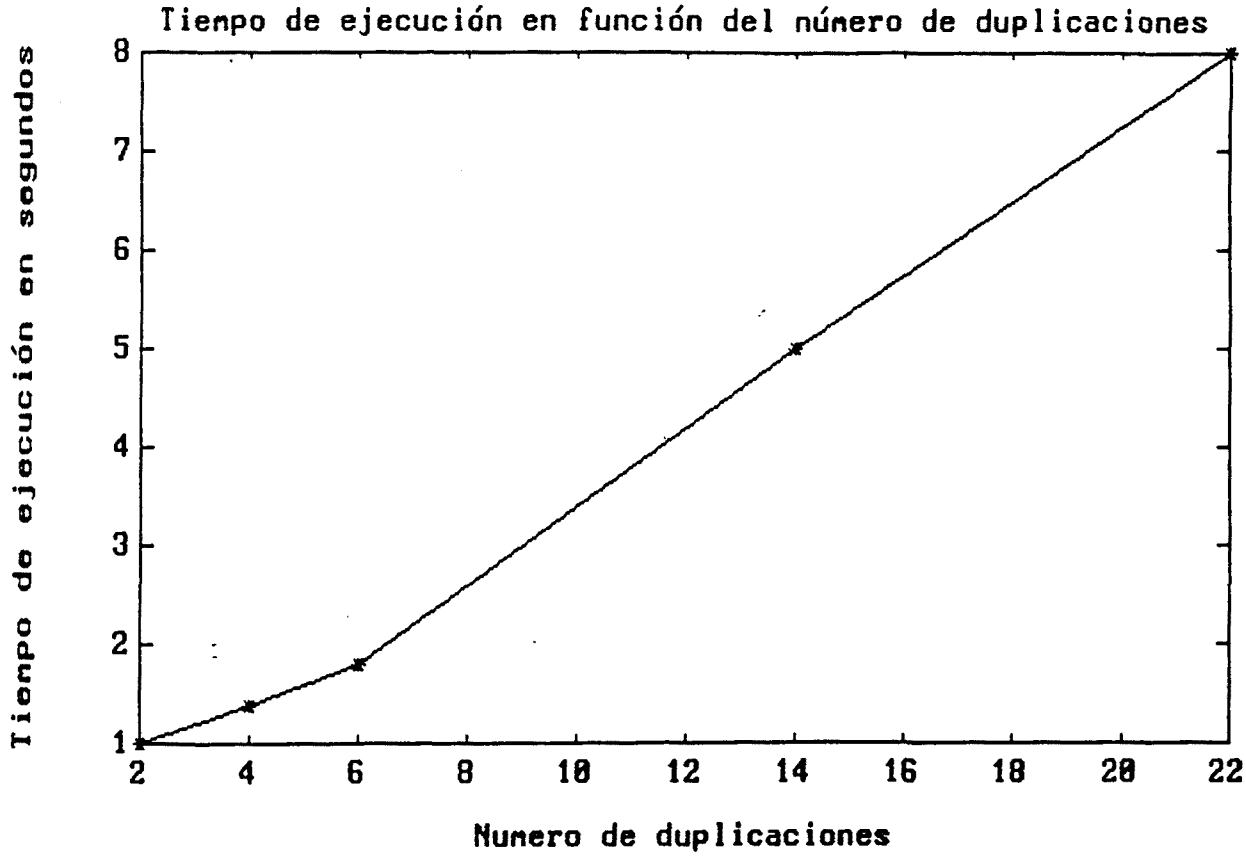


Figura 3.6

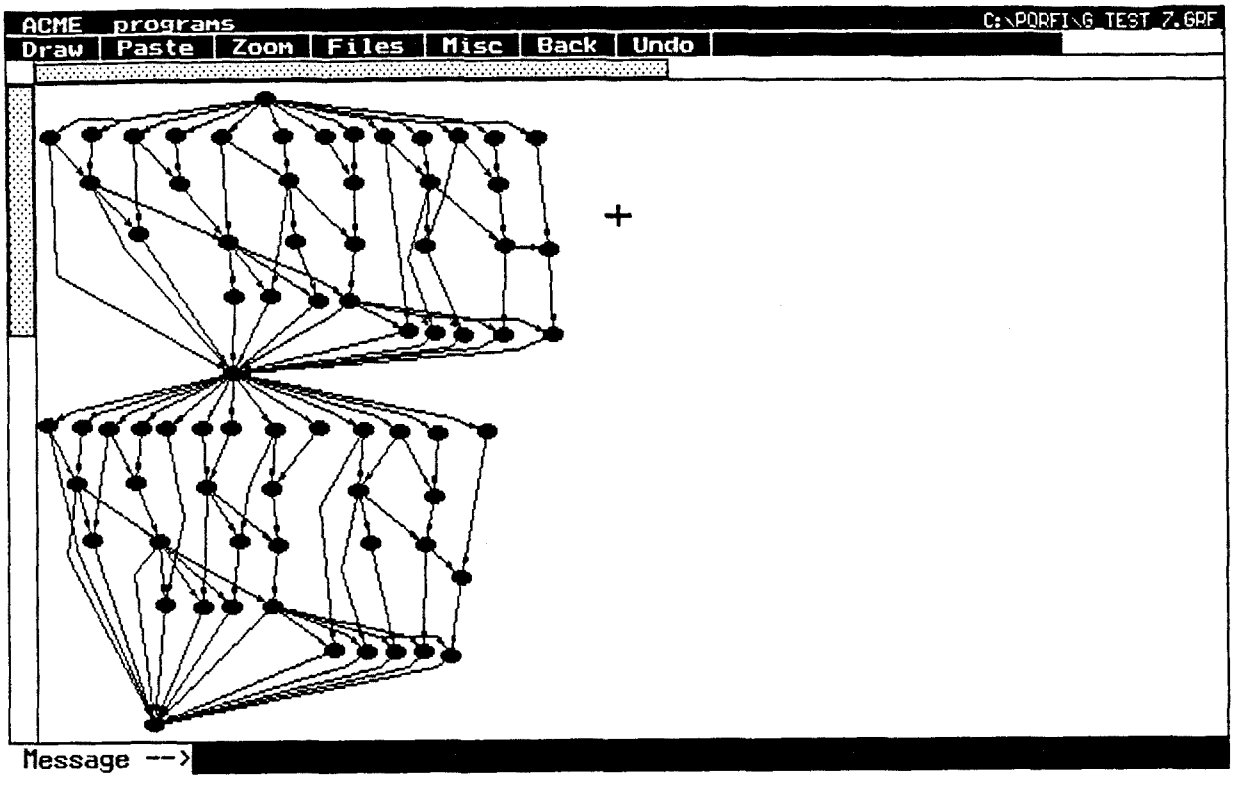


Figura 3.7

3.2.2.- Descripción del algoritmo CP/MISF/TD/RC

En el presente apartado describiremos el funcionamiento del algoritmo propuesto. En las figuras 3.8 y 3.8bis, se presentan el diagrama de flujo y el pseucocódigo correspondiente con objeto de mostrar los pasos más importantes que involucra el algoritmo.

Paso 1: Tomemos la asignación inicial proporcionada por el CP/MISF {Asig_init} obtenida para los tiempos de ejecución (Ti) y ejecutar considerando los tiempos (Ti*). Sea {T_init} el tiempo de ejecución para la asignación inicial obtenido mediante esta ejecución.

Paso 2: Comenzar la generación de la traza de Gantt, para la asignación inicial (Ti), con los nuevos tiempos (Ti*) considerando las siguientes alternativas, a) si existe procesador libre y todas las tareas asignadas de la asignación inicial, continuar el proceso de simulación, sino b) determinar el primer instante libre de un procesador

y mirar si existen tareas listas asignadas a otros procesadores, en caso afirmativo, asignar al procesador libre aquella tarea lista que tenga un mayor nivel, en caso negativo continuar la simulación, e ir al paso 2 a).

Paso 3: Simular la ejecución de la tarea duplicada (TW), e ir avanzando el tiempo de simulación construyendo la correspondiente traza de Gantt para el intervalo considerado. Si no existen periodos de inactividad de los procesadores, ir simulando las tareas asignadas.

Paso 4: Mientras existan instantes libres en los distintos procesadores, ir al paso 2. Si no, ir al paso 5.

Paso 5: Volver a simular el grafo en su totalidad con las tareas que hayan podido ser duplicadas.

Paso 6: Si el tiempo obtenido con duplicaciones (TF) es peor que {T_init}, mantener la configuración inicial {Asig_init} y olvidar las duplicaciones.

Con objeto de mostrar el funcionamiento del algoritmo y exponer algunas de sus características más relevantes, a continuación mostramos el desarrollo del algoritmo para el grafo de la figura 3.1b. A partir de la asignación proporcionada por el CP/MISF mostrada en la figura 3.1bis a), simulamos la ejecución del grafo considerando los tiempos modificados (Ti*). La nueva traza de Gantt para los nuevos valores se presenta en la figura 3.1bis b). Cuando el primer instante de tiempo (procesador ocioso) es encontrado (procesador 1, instante 12) realizamos un intento para duplicar una tarea lista asignada a otro procesador en ese instante (tarea 3 asignada al procesador 2). Por tanto lo primero que mira el algoritmo es si hay tareas listas que comiencen cuando el procesador comienza un periodo de inactividad. Si hay varias tareas que satisfacen la condición de activas en ese instante, la tarea candidata es aquella que tiene un mayor nivel. En caso de no encontrar ninguna tarea lista en ese periodo de inactividad se avanzará al siguiente paso de la simulación del grafo. Como se observa, el hecho de realizar la duplicación de la tarea 3, en el

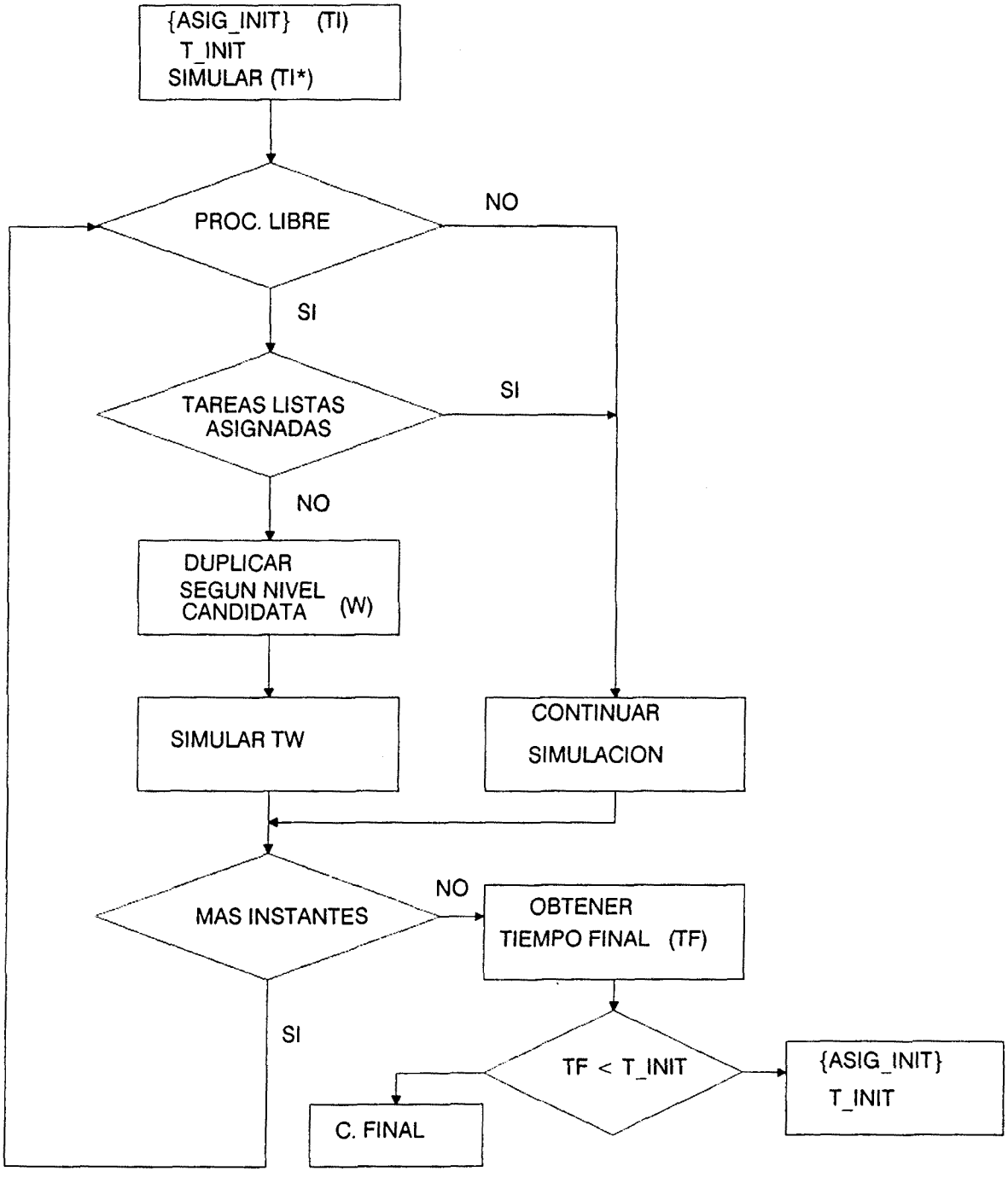

{ASIG_INIT} (TI)
T_INIT
SIMULAR (TI*)
PROC. LIBRE
NO
SI
TAREAS LISTAS ASIGNADAS
SI
NO
DUPLICAR SEGUN NIVEL CANDIDATA (W)
SIMULAR TW
CONTINUAR SIMULACION
MAS INSTANTES
NO
OBTENER TIEMPO FINAL (TF)
SI
TF < T_INIT
{ASIG_INIT}
T_INIT
C. FINAL

Figura 3.8

```
BEGIN

   Comenzar_asignacion CP/MISF (Ti) ;
   Simular_la_ejecucion (Ti*);
   { Asig_init }, T_init ;
   < Proceso de duplicacion >

   REPEAT

      IF (Procesador = libre) THEN

         IF ( No existen_tareas_asignadas_a_este) THEN BEGIN

            Duplicar_tarea candidata_segun_mayor_nivel;
            Simular_tarea_candidata;
            END

      ELSE Continuar_simulacion;

   UNTIL ( Instantes = falso );

   Simular_traza_con_duplicaciones;
   Obtener_tiempo_Final (TF);

   IF (TF > = T_init) THEN Asig_init (T_init);

   ELSE Asignar_con_duplicaciones;

END.
```

Figura 3.8bis

procesador 1, ha generado una disminución en el tiempo de ejecución de 1 unidad de tiempo, y la generación de otra duplicación (tarea 10, en tiempo 14 y procesador 2) como se observa en la figura 3.1bis c).

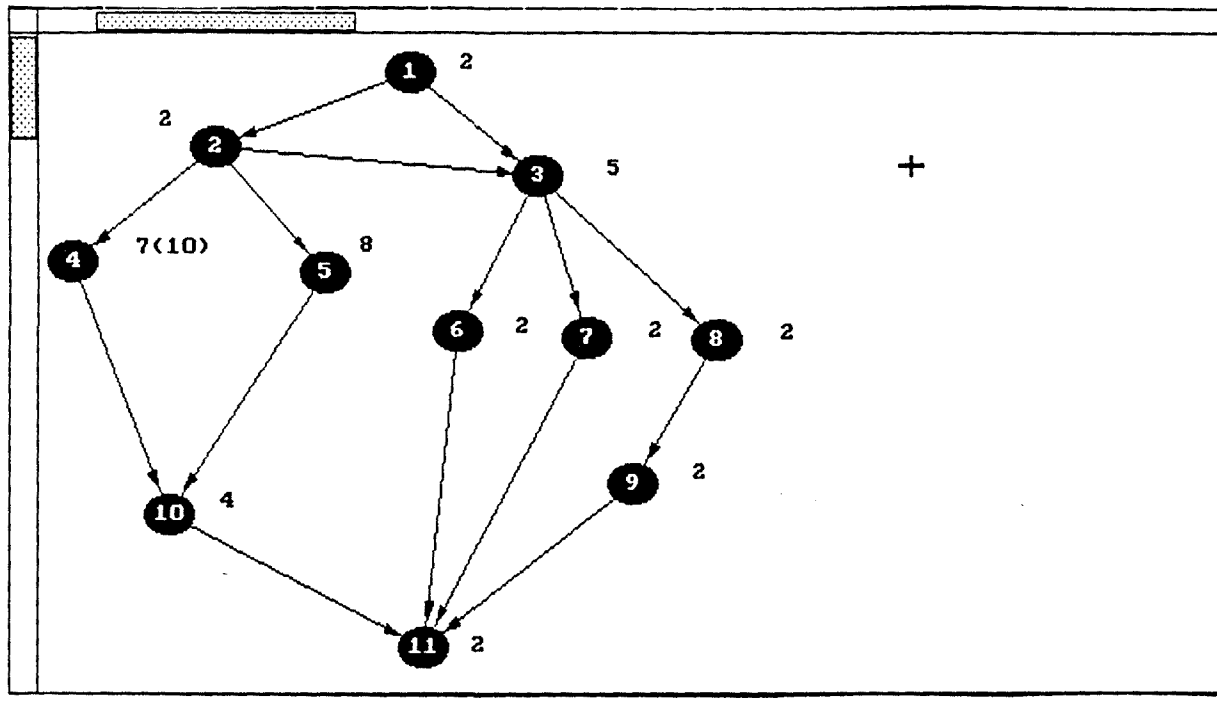


Figura 3.1b

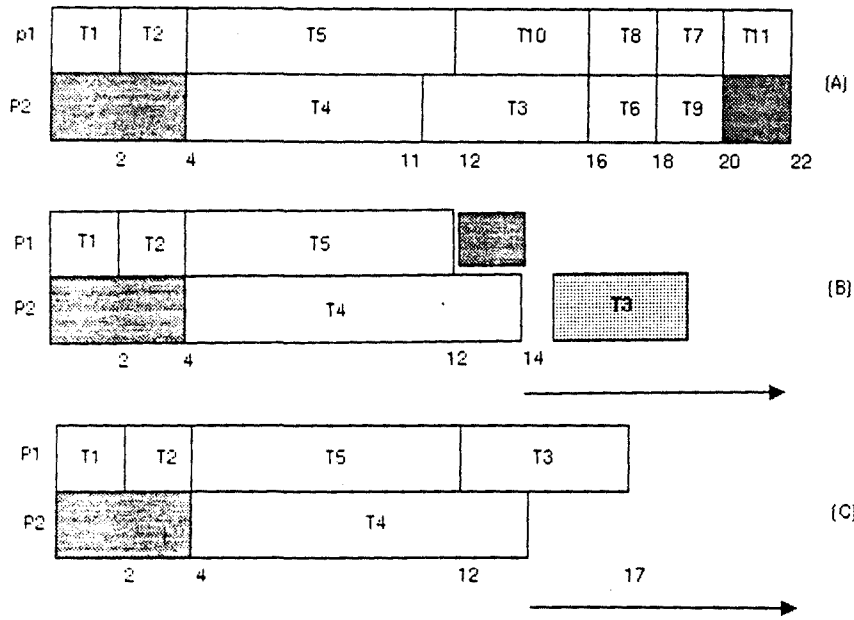


Figura 3.1bis

3.2.2.1.- Análisis de la complejidad más desfavorable del algortimo CP/MISF/TD/RC

El máximo número de instantes de tiempo a analizar por el algoritmo será m * k, donde m es el número de procesadores en el sistema y k = max {Ki}, e i = 1..n, es el número de instantes a analizar en el procesador i. Ya que para cada instante de tiempo hay que comprobar que tareas hay activas en ese momento, como máximo podremos tener n; la cota máxima de complejidad para el algoritmo será $O(n^2m)$.

3.2.2.2.- Análisis de la complejidad real

Al igual que para el algoritmo anterior, la complejidad teórica más desfavorable para el algoritmo que nos ocupa, es una cota de tiempo que practicamente, no se dará nunca en la realidad, con objeto de tener valores concretos de tiempo, y acercarnos más a lo que sucede en la realidad, mostraremos para el conjunto de grafos presentados para el estudio de la complejidad real en el apartado anterior, los valores de tiempo de las rutinas más significativas que implican la ejecución del algortimo.

La figura 3.9, muestra el tiempo de ejecución de las rutinas más significativas del algoritmo, para el conjunto de grafos que aparecen en la figura 3.4.

Con objeto de poder comparar, los tiempos de ejecución de las secciones de código representativas de cada uno de los algoritmos utilizados, en la figura 3.10, se presenta la complejidad real del algortimo CP/MISF, a fin de poder resumir bajo la gráfica que se muestra en la figura 3.11 la complejidad real del algoritmo tratado en este apartado, de complejidad $O(n^2)$, comparándolo igualmente con la complejidad real del algoritmo CP/MIDF/TD. Como se muestra en la gráfica de la figura 3.9, la complejidad real del algoritmo CP/MISF/TD/RC, es practicamente igual a la teórica $O(n^2)$, resultado por otro lado esperado, debido al cambio fundamental en cuanto a actuación del algoritmo CP/MISF/TD/RC respecto al CP/MISF/TD, al realizar el que nos ocupa, el proceso de duplicación, en un único paso, basándose en la idea de primer instante libre, a la hora de duplicar una tarea, sin comprobar el resultado final

en cuanto a tiempo de ejecución que pudiese generar el conjunto de duplicaciones obtenidas.

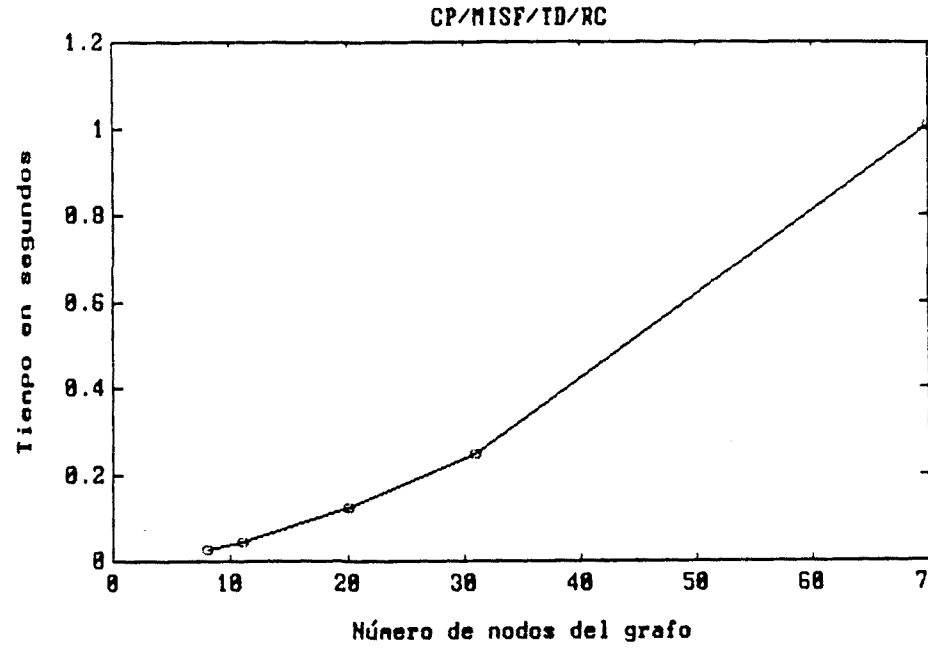


Figura 3.9

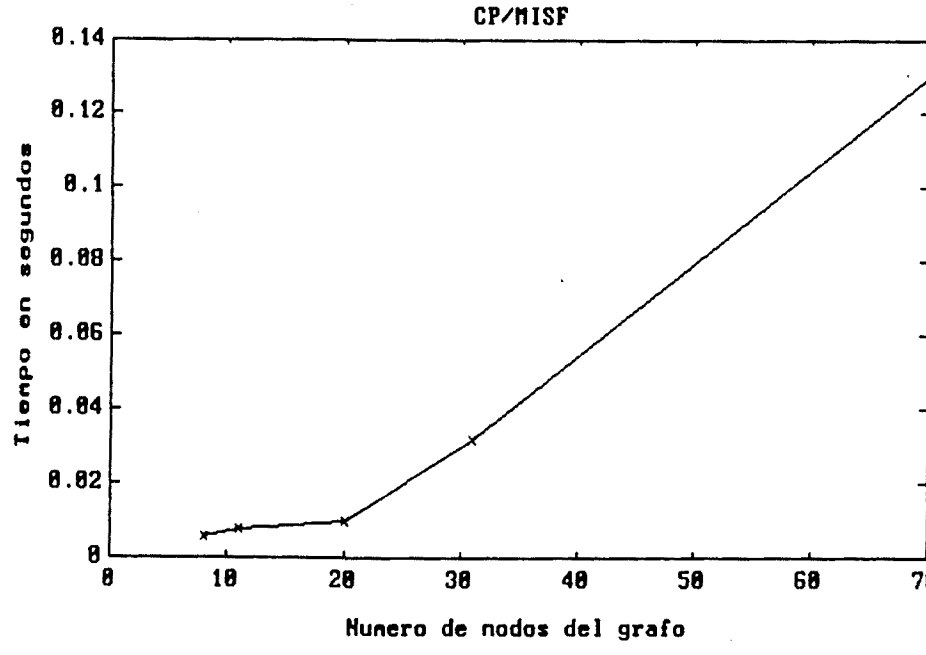


Figura 3.10

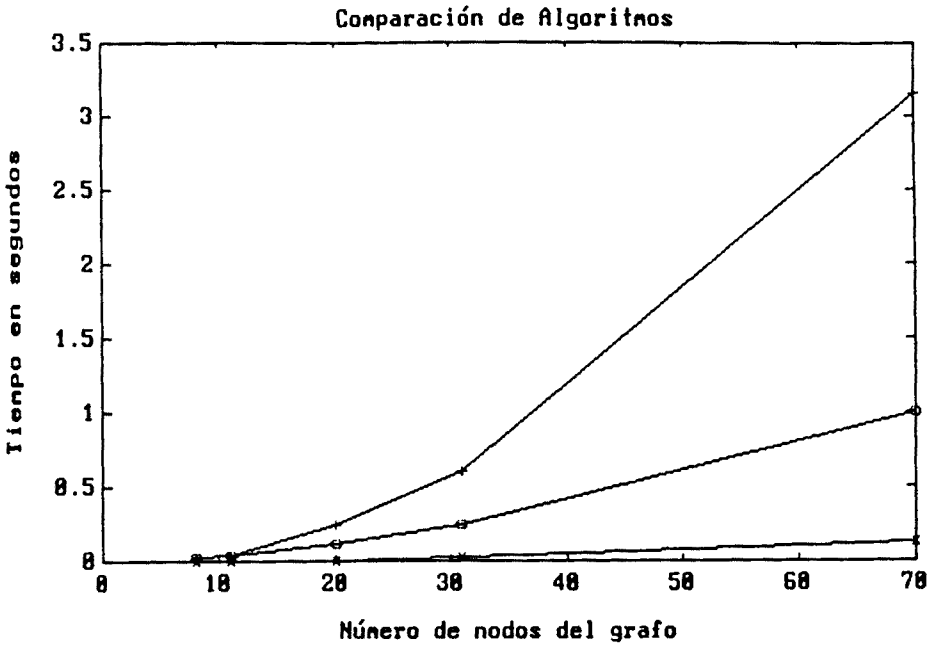


Figura 3.11

Un detalle a tener en cuenta, y que nos puede dar una idea de la importancia de la complejidad temporal real de un algoritmo, es el estudio realizado para los grafos de la figura 3.4 a-e), del tiempo de cálculo para evaluar las cotas de Fernández y Bussell mostradas en la figura 3.12; en los resultados obtenidos para los grafos mencionados anteriormente,se puede observar el crecimiento en cuanto a la complejidad temporal del algoritmo con el cuadrado del camino crítico (observar el tiempo de ejecución obtenido por las rutinas representativas del algoritmo). Para los grafos en cuestión, los caminos críticos son 16, 112, 90, 169 y 524, se puede observar en la gráfica este hecho; para el grafo de nombre "CINCO" en la figura 3.4, el tiempo global de las rutinas representativas es de 6.06 segundos y el número de nodos de 11, el grafo de nombre "SEIS", tiene 20 nodos, un tiempo acumulado de rutinas de 4.3 segundos pero en cambio un camino crítico de 90 unidades de tiempo, de ahí, que la caida que aparece en la gráfica mostrada en la figura 3.12, no sea continua, ya que en las abcisas de la gráfica los grafos utilizados para medir la complejidad real de los algoritmos, no están representados por el tiempo de su camino crítico, sino por el número de nodos del grafo en cuestión.

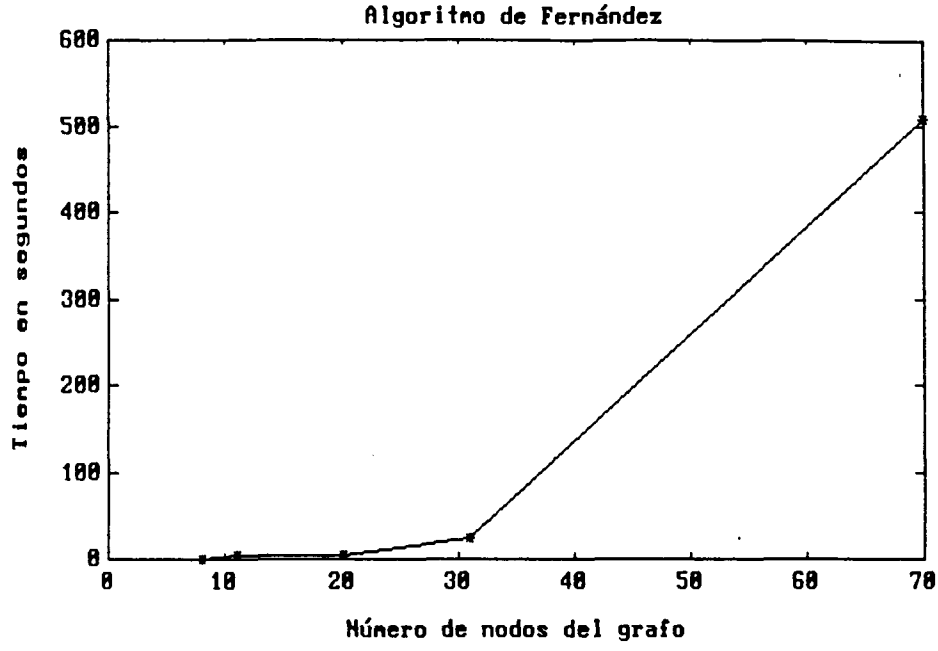


Figura 3.12

3.2.3.- Requerimientos de sincronización

No menos importante que el problema de la planificación de tareas es el problema de la sincronización. Cuando una aplicación podemos descomponerla en un conjunto de tareas que pueden realizarse de forma paralela, y necesitan cooperar para poder ejecutarse, aparece el problema de fijar la ordenación en cada procesador de las tareas asignadas para su ejecución. El problema se complica cuando algunas tareas, necesitan de los resultados obtenidos por otras tareas que están siendo ejecutadas en otros procesadores. La sincronización de tareas en lo referente a fijar la estructura de dependencias y la comunicación entre procesos con objeto de intercambiar información, son aspectos importantes a estudiar en cualquier entorno de cómputo distribuido.

Como hemos expuesto anteriormente, el mecanismo de duplicar tareas, en orden a eliminar los tiempos de inactividad de los procesadores que componen nuestro entorno, tiene implicaciones en el algoritmo de ejecución, ya que deberá asegurarse, que cada tarea sólo se ejecuta una vez. A tal efecto, el algoritmo de

duplicación deberá incluir procedimientos de sincronización, durante la ejecución real del programa, a fin de que garanticen este comportamiento. Cada vez que se ejecuta una de las copias de la tarea duplicada, el procesador que la ha ejecutado deberá informar al otro procesador que tiene la tarea a fin de no volverla a ejecutar otra vez, ya que ambas estaban listas para ser ejecutadas. Estos algoritmos de sincronización corresponderían a la función de exclusión mutua en ejecución para tareas distribuidas [55,56].

El grafo original queda modificado al añadirse por un lado nuevas tareas (duplicaciones) y nuevos arcos de conexión entre tareas que permitan pasar los datos y obtener los resultados desde las duplicadas. Además las tareas predecesoras y sucesoras de una duplicada habrán de modificarse con objeto de a) enviar la información a ambas tareas (original y duplicada) y b) recibir, alternativamente, la información de la tarea ejecutada (original o duplicada).

Otra de las consideraciones a tener en cuenta, es la obligatoriedad de crear, nuevas rutas (caminos) para pasar los datos generados por la tareas duplicadas, así como los caminos de entrada con objeto de ejecutar la tarea duplicada para los datos en cuestión. Un punto importante a tener en cuenta en futuras ampliaciones, será el problema del ruteo de los datos cuando estudiemos arquitecturas que no presentan una estructura de conexión completa, y las relaciones entre el peso de las comunicaciones y los volumenes de cómputo ya no sean despreciables.

3.3.- Aplicación de los algoritmos CP/MISF/TD y CP/MISF/TD/RC sobre un conjunto de grafos

Con objeto de evaluar las prestaciones de los algoritmos desarrollados se ha realizado una experimentación extensiva sobre un amplio conjunto de grafos.

En esta sección presentamos los resultados obtenidos cuando aplicamos los algoritmos CP/MISF/TD y CP/MISF/TD/RC a un conjunto de grafos, así como la

bondad y desviaciones que presentan los algoritmos anteriores frente a las cotas (algoritmos de referencia) que utilizaremos como comparación.

3.3.1.- Elección del tipo de grafos

En el presente apartado se pretende mostrar algunas características de las aplicaciones sobre las cuales hemos desarrollado nuestra experimentación.

Una de las preguntas que se plantean cuando intentamos probar la validez de un algoritmo de planificación, es la elección del conjunto de grafos que consideramos representativos para realizar los tests de rendimiento. Fundamentalmente, disponemos de dos opciones bien diferenciadas; podemos trabajar con grafos que representen aplicaciones reales, o bien diseñar grafos sintéticos que recojan de alguna manera los parámetros significativos que puedan afectar al sistema multiprocesador (múmero de procesadores, rango de variación del tiempo de ejecución de los nodos, número de precedencias y nodos del grafo, grado de paralelismo, tipo de paralelismo etc).

Nuestra experimentación se ha realizado en las dos lineas de actuación expuestas anteriormente, en 3.3.2 analizaremos los grafos sintéticos seleccionados, siendo 3.3.3 el apartado dedicado al estudio de los grafos provenientes de aplicaciones.

3.3.2.- Grafos sintéticos

En una primera etapa, se diseñaron grafos a medida (diferentes tipos de parámetros) para comprobar las prestaciones de nuestros algoritmos respecto al tiempo final de la aplicación, así diseñamos grafos en base a los siguientes parámetros:

- Número de nodos

- Número de arcos

- Tipo de paralelismo, abierto, en el sentido de ir generando paralelismo a partir de cada tarea.

- Cantidad de nodos que pueden variar

Las figuras desde 3.13a, hasta 3.13k muestran la topología del conjunto de grafos utilizados en esta primera fase. En la figura 3.14 los grafos son identificados mediante el nombre que aparece en su figura correspondiente. El orden es el de la secuencia de figuras mostrada. En la figura 3.14 se muestra el número de procesadores calculados a partir de las técnicas de Fernández y Bussell, y que constituirán el número de ellos que utilizaremos para calcular los tiempos de ejecución de la aplicación y que utilizaremos para comparar la bondad de los algortimos presentados anteriormente.

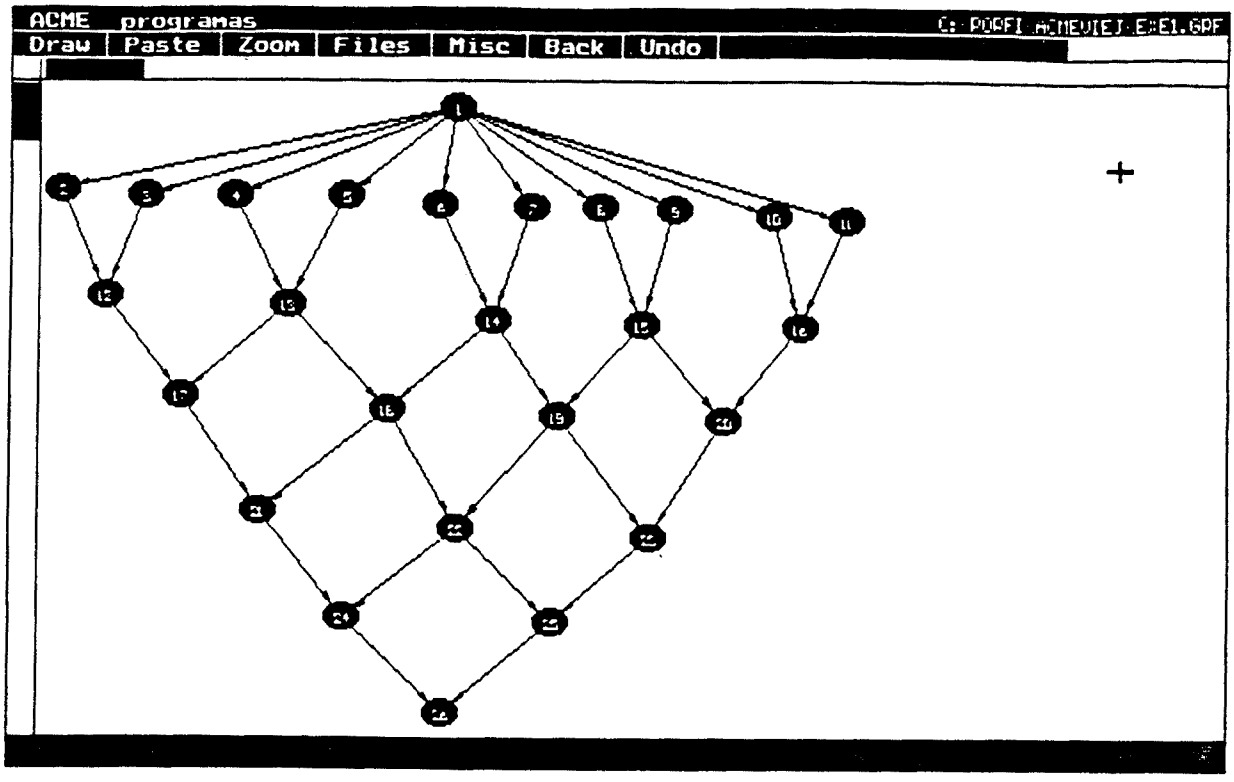


Figura 3.13a

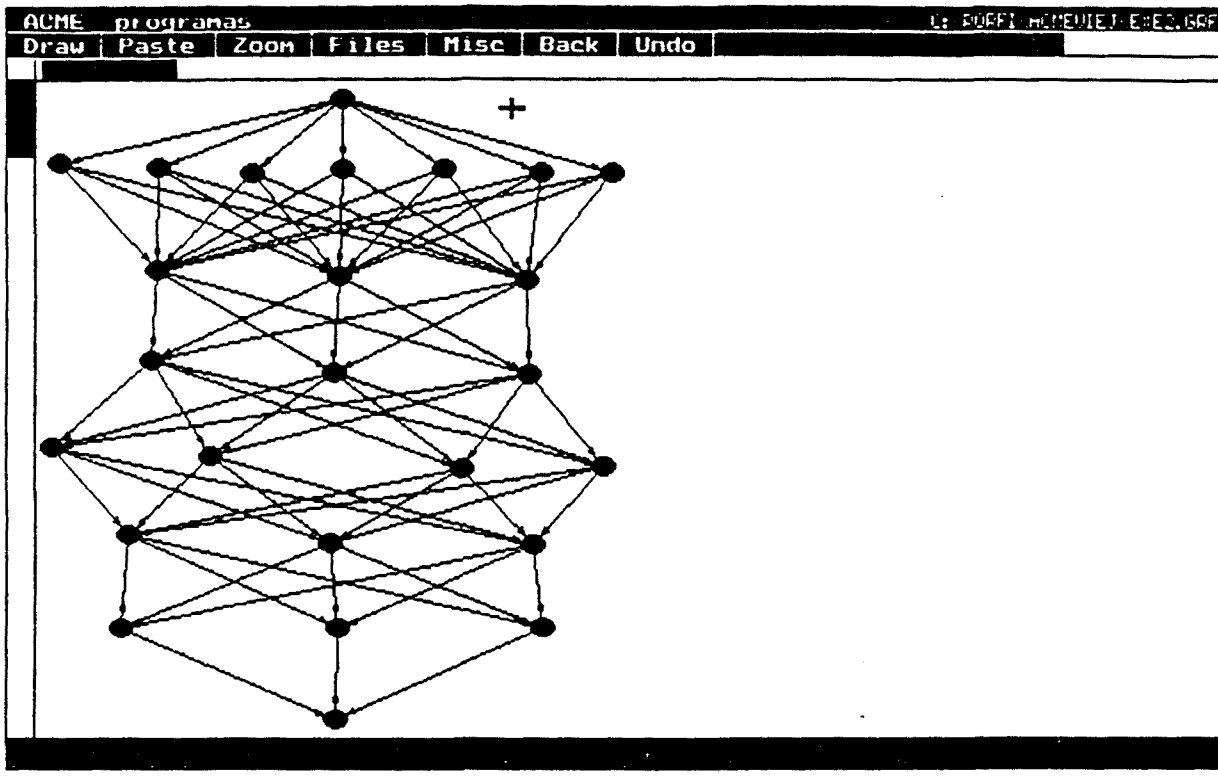


Figura 3.13.b

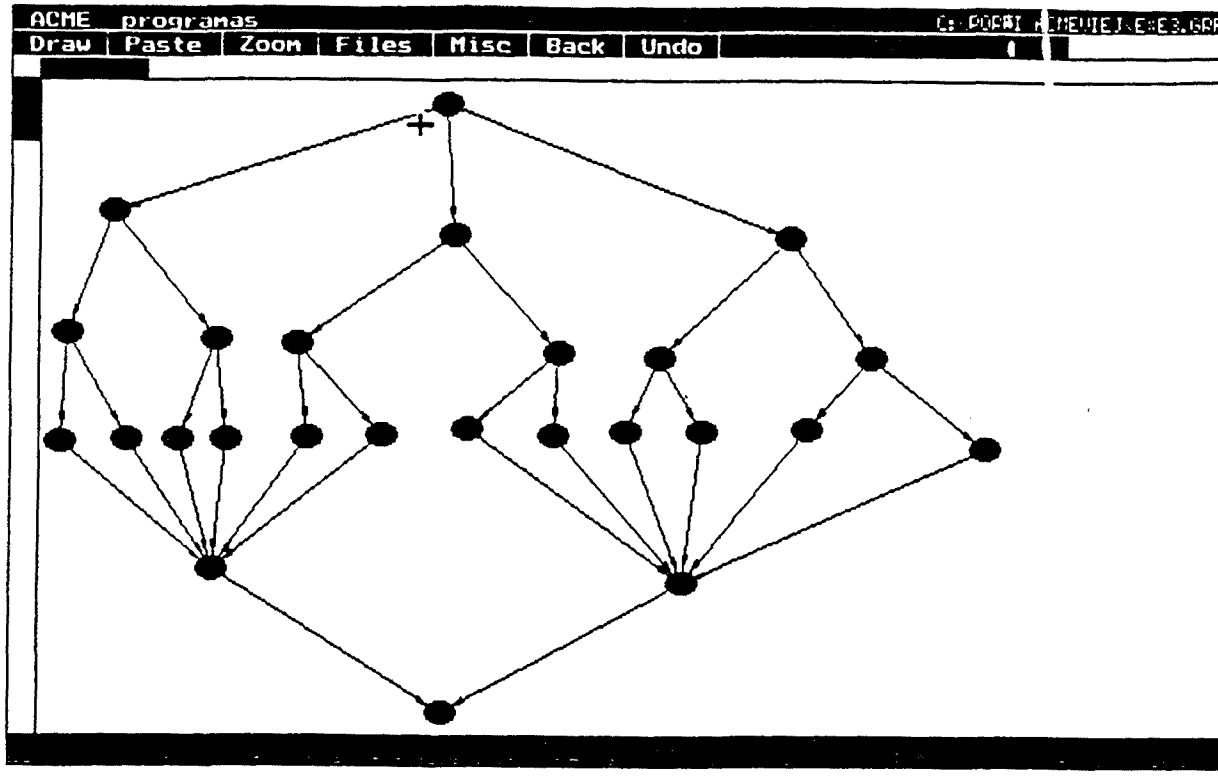


Figura 3.13c

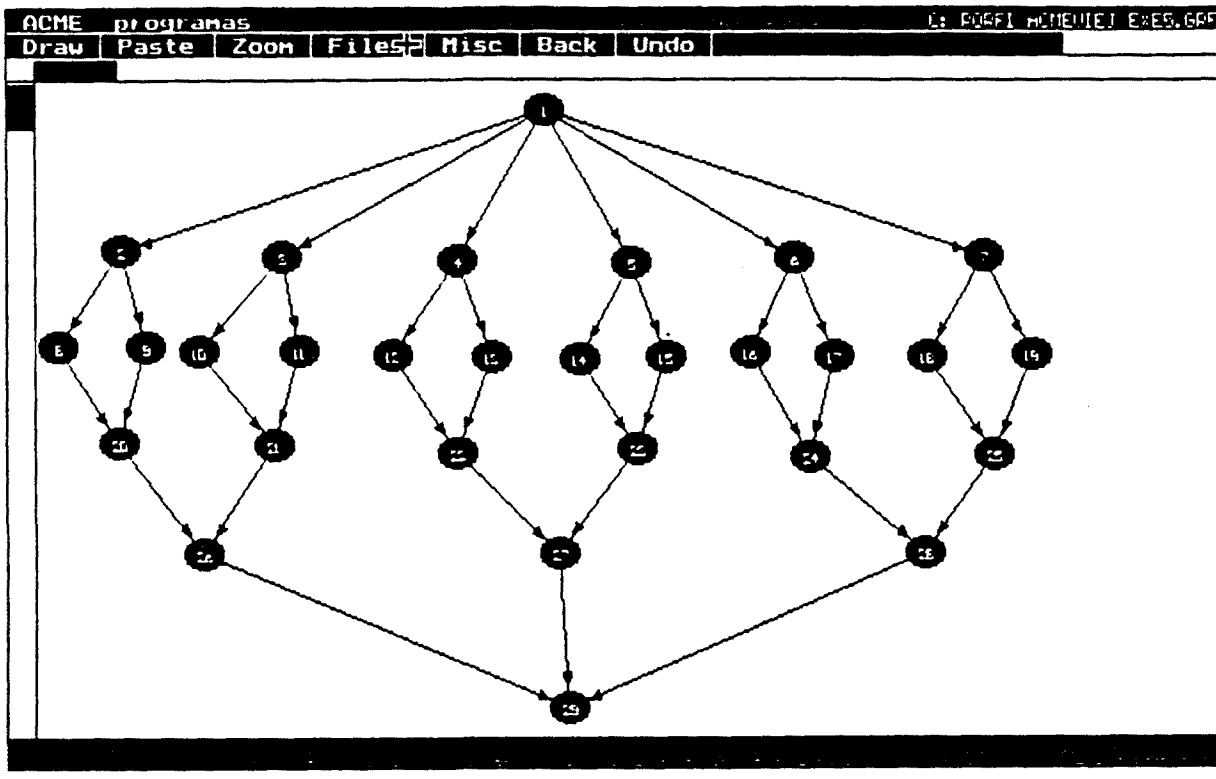


Figura 3.13d

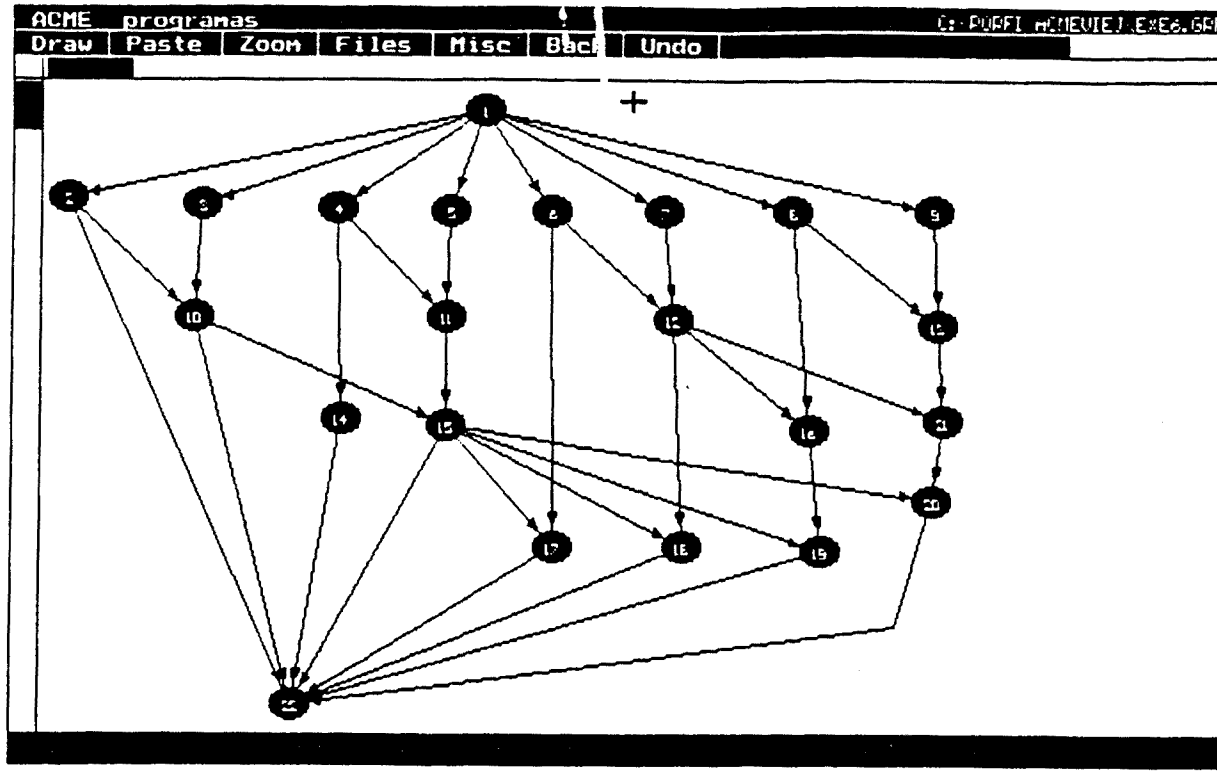


Figura 3.13e

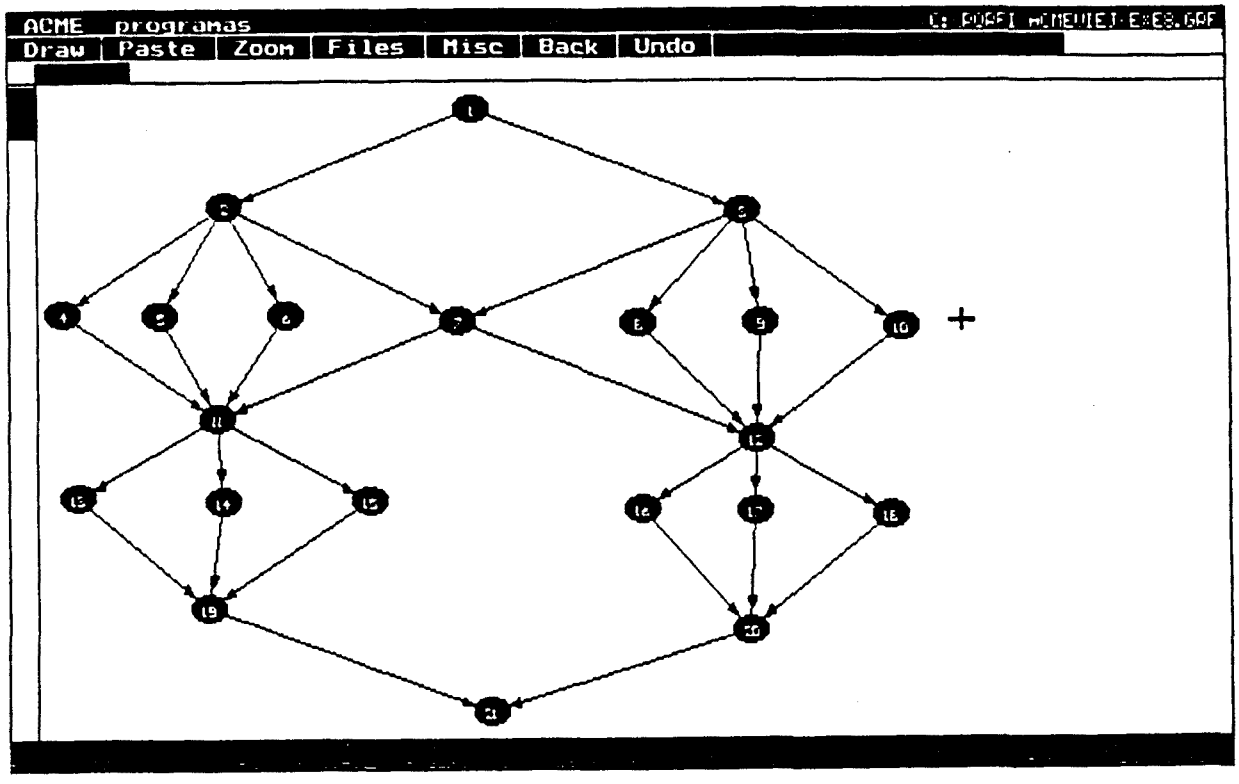


Figura 3.13f

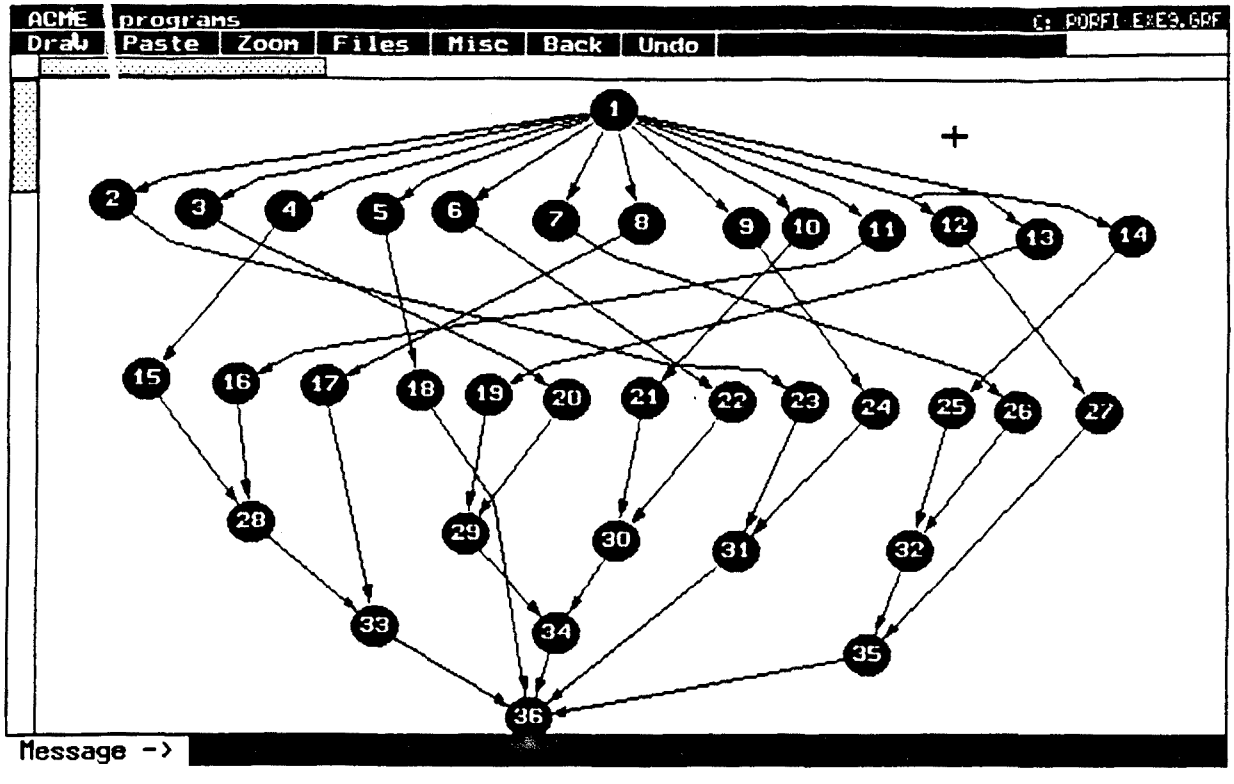


Figura 3.13g

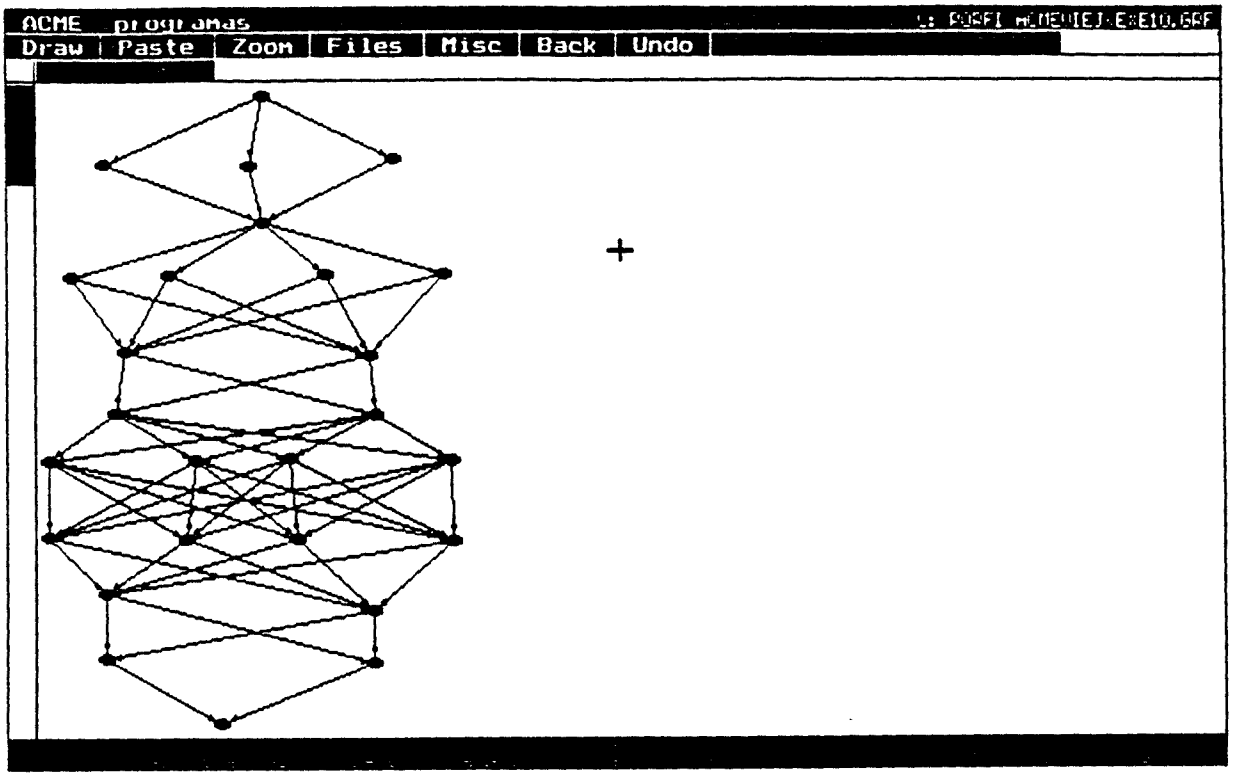


Figura 3.13h

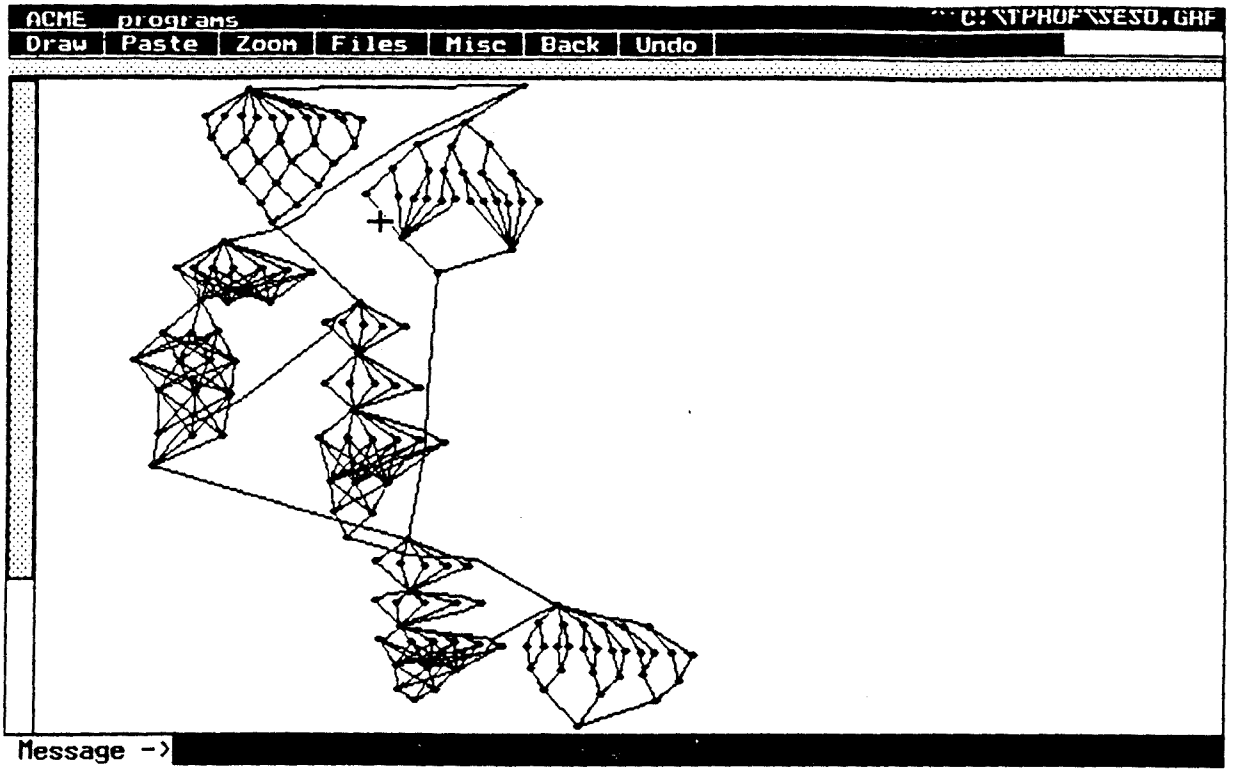


Figura 3.13i

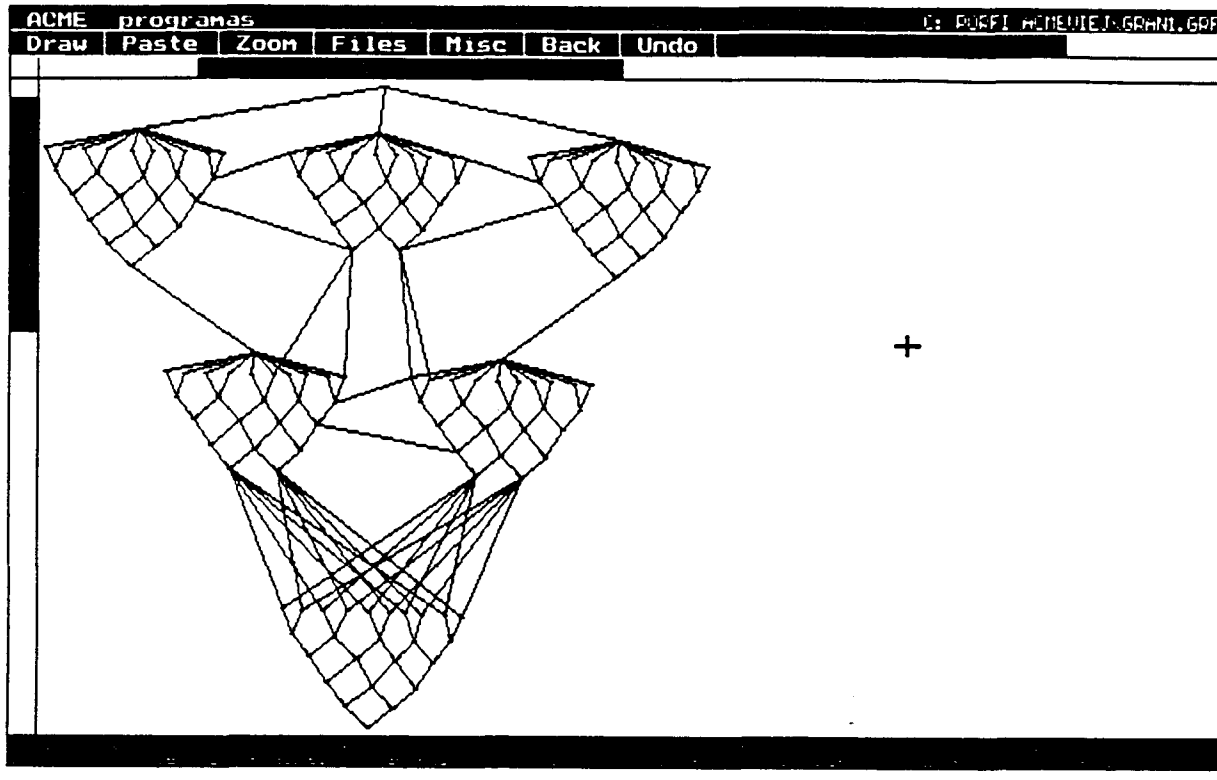


Figura 3.13j

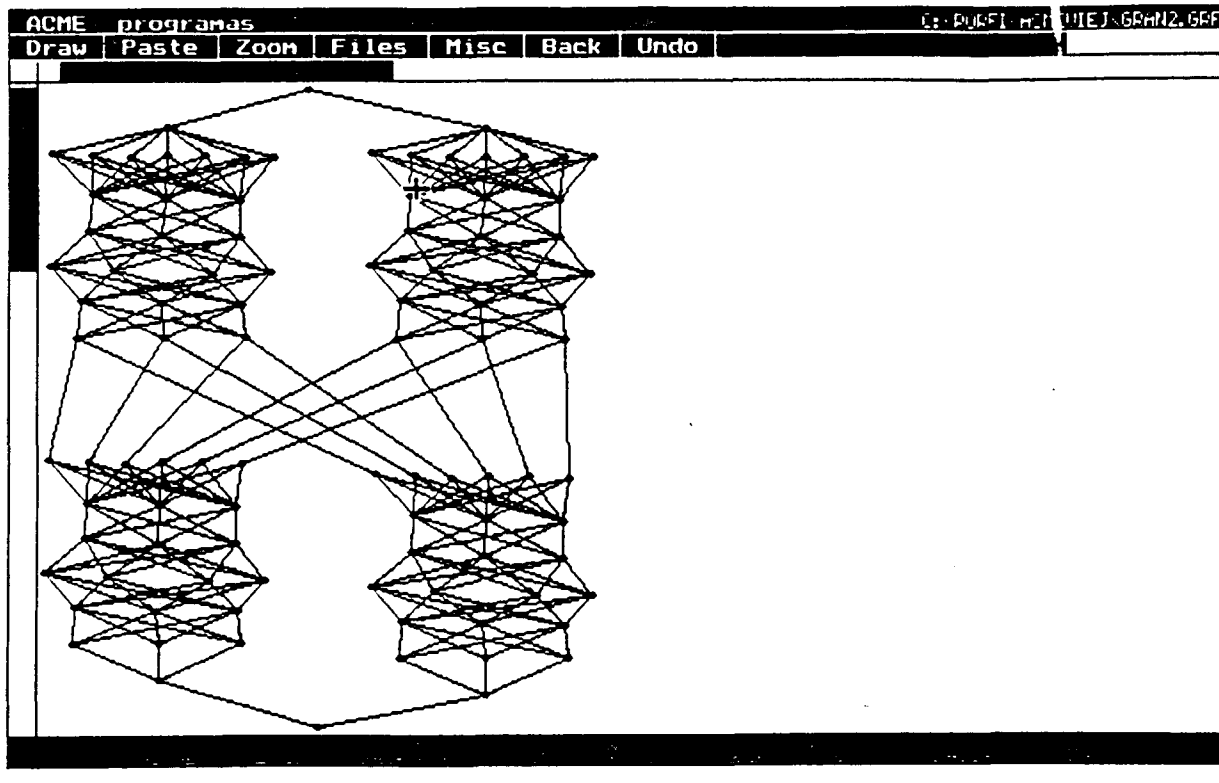


Figura 3.13k

Como se observa en la tabla resumen de la figura 3.14, el conjunto de parámetros significativos de los grafos son los siguientes:

- Número de procesadores, (Num. Proc.), varían entre 3 y 7.

- Número de nodos, (Nodos), variando entre 21 y 156.

- Número de arcos, (Arcos), entre 32 y 306.

- Conjunto de nodos que pueden variar (Var.), entre 6 y 11.

Los valores obtenidos de tiempo de ejecución, correspondientes a cada grafo que son utilizados para medir la prestación de los algoritmos se presentan igualmente en la figura 3.14, son los siguientes:

- El tiempo de ejecución proporcionado por el algoritmo CP/MISF/TD, para la asignación obtenida a partir de los tiempos Ti más duplicaciones, y se ejecuta para los tiempos Ti*, expresado como On3 en la figura 3.14.

- El tiempo de ejecución proporcionado por el algoritmo CP/MISF/TD/RC, denominado On2 en la figura 3.14 , obtenido a partir de la asignación obtenida a partir de los tiempos Ti más duplicaciones, y ejecutando para los valores Ti*.

- El tiempo proporcionado por el algoritmo CP/MISF cuando se calcula la asignación para los Ti, y se ejecuta para el segundo valor Ti*, este tiempo viene expresado por Texec. en la figura 3.14. Este valor correspondería a **la situación real que podría darse de una ejecución con tiempos vatiables, pues la asignación se habría realizado tomando un conjunto de tiempos, y durante la ejecución real, por dependencias de datos, se habrían producido unos tiempos de ejecución para las tareas diferentes.** Las ganancias que podrían generarse mediante los algoritmos de duplicación propuestos deben evaluarse por tanto con mejoras frente a este valor.

- El tiempo proporcionado por el algoritmo CP/MISF cuando se aplica para el nuevo valor Ti*. Se calcula la asignación para Ti* y se ejecuta para los Ti*. Este tiempo viene representado por Cp en la figura 3.14. Este valor sería la cota inferior (teórica) en el tiempo de ejecución (valor óptimo de referencia mínimo) por cuanto la asignación se había realizado con el conjunto de valores que se iban a producir, como tiempo de las tareas variables, en tiempo de ejecución. Ahora bien, este valor sólo sería el óptimo para la ejecución con Ti* y ahora no lo sería para Ti.

Nom.	Num. Proc.	Nodos	Arcos	Var.	Texec.	On3	On2	Cp
Exe1	3	26	39	10	475	344	344	334
Exe2	3	25	72	11	342	330	330	330
Exe3	5	25	35	8	237	184	184	173
Exe5	4	29	39	7	375	321	321	302
Exe6	4	22	40	7	301	301	301	301
Exe8	3	21	32	6	424	343	343	325
Exe9	5	36	46	9	244	226	227	191
Exe10	3	26	60	9	601	601	601	591
Seso	4	156	306	11	1659	1626	1593	1516
Gran1	7	153	253	10	1030	1020	1020	980
Gran2	5	96	285	10	1113	1053	1083	1053

Figura 3.14

En el apéndice B, se detalla para cada grafo, el conjunto de parámetros significativos, topología del mismo, tiempos de ejecución, conjunto de nodos que varían, así como los tiempos de ejecución finales obtenidos para la aplicación de los algoritmos.

Con objeto de facilitar la interpretación de los resultados obtenidos en nuestra experimentación, a continuación presentamos las tablas de rendimiento deducidas de la figura 3.14. Los índices de rendimiento calculados para cada uno de los grafos tabulados en la figura 3.14, han sido los siguientes:

Ganancia máxima obtenible: representa la reducción obtenida por el algoritmo CP/MISF para los valores de ejecución de los nodos (Ti), y ejecutando la asignación para los nuevos valores (Ti*), respecto al valor proporcionado de tiempo de ejecución por el algoritmo CP/MISF calculando la asignación para (Ti*), y simulando la ejecución para los mismos tiempos de ejecución de los nodos (Ti*); cota máxima considerada (Texec - Cp), en la figura 3.14.

Ganancias obtenidas por los algoritmos propuestos: Reducción obtenida en valor absoluto por nuestros algoritmos CP/MISF/TD y CP/MISF/TD/RC respectivamente, respecto al tiempo de ejecución proporcionado por el algoritmo CP/MISF, manteniendo la asignación proporcionada para los (Ti), y simulando la ejecución para los tiempos de ejecución de los nodos (Ti*) . (Texec-On3) y (Texec-On2) respectivamente.

Desviación respecto del valor óptimo: considerando el tiempo de ejecución "óptimo", el proporcionado por el algoritmo CP/MISF, para la asignación obtenida para los tiempos de ejecución (Ti*) y simulando la ejecución para los mismos tiempos (Ti*), el parámetro introducido en este apartado (desviación), nos proporciona una idea de lo alejado o proximo que nos encontramos respecto del "óptimo" considerado. (On3-Cp) para el caso del algoritmo CP/MISF/TD y (On2-Cp) para el algoritmo CP/MISF/TD/RC.

Ganancia % On3, Ganancia % On2: representan respectivamente el % que sobre el valor considerado máximo (Texec-Cp), representan los algoritmos propuestos CP/MISF/TD y CP/MISF/TD/RC.

Ganancia % Oni = ((Texec-Oni)/(Texec-Cp))*100, i=2..3: representa la ganancia relativa de los algoritmos propuestos respecto a la ganancia considerada como maxima.

Desviación % Oni = ((Oni-Cp)/Cp)*100, i = 2..3: representa la desviación sobre el valor considerado como óptimo.

Desviación % máxima = ((Texec-Cp)/Cp)*100: valor de máxima desviación considerada. Promedio: En ambos casos, para los parámetros de ganancia y desviación, representa la media obtenida para los algoritmos tratados, considerando los 11 ejemplos de grafos estudiados.

Como se observa en las tablas 3.14a-c), utilizando distintas representaciones, el algoritmo CP/MISF/TD, es capaz de obtener una ganacia promedio sobre las ganancias individuales del 55.3%, con desviaciones promedio sobre el valor óptimo teórico del 4.7%; igualmente el algoritmo CP/MISF/TD/RC, obtiene una ganancia promedio del 52.6% sobre la ganacia máxima y una desviación promedio del 4.8% sobre el valor óptimo considerado.

Se han obtenido por tanto reduciones, en promedio, del orden del 75% sobre la desviación máxima que se presentaba por la degradación de los tiempos de ejecución al pasarse de una desviación promedio del 17% a una del 4.7 y 4.8 respectivamente.

Para los ejemplos mostrados, se observa, que el algoritmo, CP/MISF/TD, proporciona en media, mejores resultados que el algoritmo CP/MISF/TD/RC, resultado bastante lógico, ya que a la hora de realizar las duplicaciones, en este último

no se realiza ninguna comprobación por simulación del tiempo de ejecución proporcionado, por cada una de las tareas duplicadas.

Probada la validez de los algoritmos frente a un conjunto de grafos sintéticos, consideramos y decidimos a enfrentar nuestros algoritmos a estructuras de grafos que se correspondieran con aplicaciones reales que se presentan en la literatura [46,47,48,50], así por ejemplo aplicamos los algoritmos a estructuras de grafo que modelaban problemas de ordenación, resoluciones de sistemas de ecuaciones lineales, problemas de predicción metereológica etc. El conjunto de nucleos utilizado, fueron los lazos de Livermore [46], a partir de ellos y mediante la unión de las estructuras representativas de estos nucleos se forman los grafos representativos de las aplicaciones reales. A continuación, se describen algunos aspectos de este tipo de nucleos utilizados en parte de la experimentación que se mostrará en este trabajo.

Nombre	Ganancia Texec-Cp	Gan. On3 Texec-On3	Gan. On2 Texec-On2	Desviacion On3-Cp	Desviacion On2-Cp
Exe1	141	131	131	10	10
Exe2	12	12	12	0	0
Exe3	64	53	53	11	11
Exe5	73	54	54	19	19
Exe6	0	0	0	0	0
Exe8	99	81	81	18	18
Exe9	53	18	17	35	36
Exe10	10	0	0	10	10
Seso	143	33	66	110	77
Gran1	50	10	10	40	40
Gran2	60	60	30	0	30

Tabla 3.14a

Nom.	Ganancia % On3	Ganancia % On2	D% On3	D% On2	D% Max Texec	Cp
Exe1	92.9	92.9	2.99	2.99	42.21	334
Exe2	100	100	0	0	3.63	330
Exe3	82.8	82.8	6.35	6.35	36.99	173
Exe5	73.9	73.9	6.29	6.29	24.17	302
Exe6	0	0	0	0	0	301
Exe8	81.8	81.8	5.53	5.53	30.46	325
Exe9	33.9	32	18.32	18.84	27.74	191
Exe10	0	0	1.69	1.69	1.69	591
Seso	23	46.1	7.25	5.07	9.43	1516
Gran1	20	20	4.08	4.08	5.10	980
Gran2	100	50	0	2.84	5.69	1053
Prom.	55.3	52.6	4.77	4.88	17.01	

D% On3: Desviacion relativa del algoritmo On3 respecto del Cp
D% On2: Desviacion relativa del algoritmo On2 respecto del Cp
D% Max: Desviacion maxima relativa respecto al Cp
On3: Algoritmo CP/MISF/TD
On2: Añlgoritmo CP/MISF/TD/RC

Tabla 3.14b

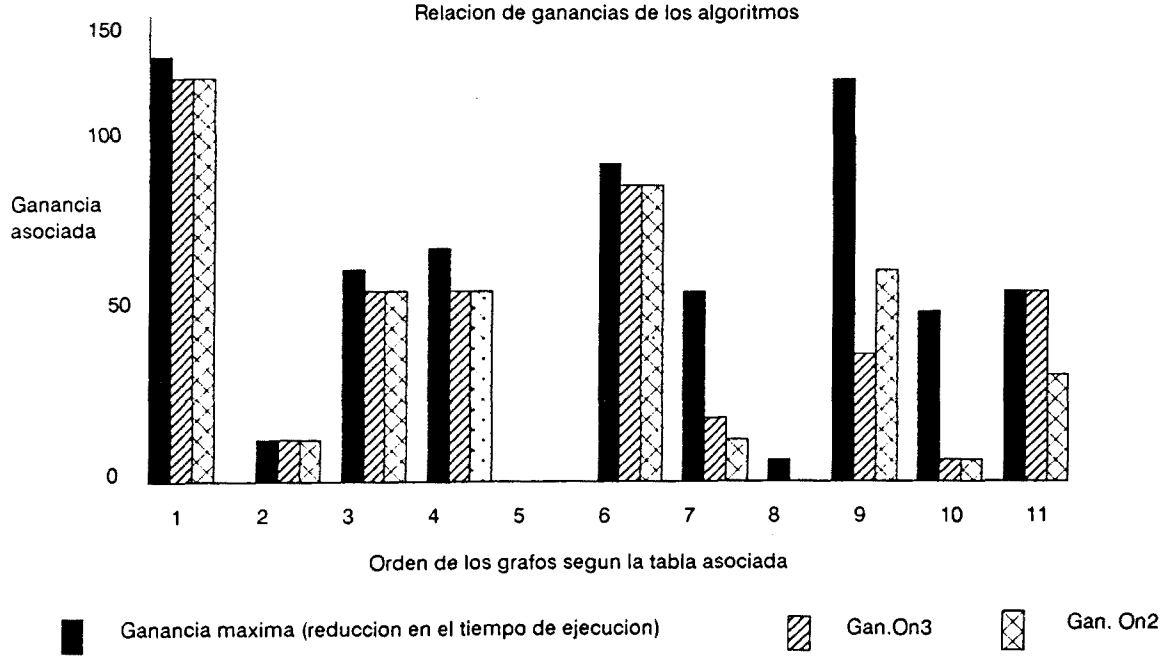


Tabla 3.14c

3.3.3.- Lazos de Livermore

Los lazos de Livermore son un conjunto de fragmentos de código escritos en lenguaje Fortran, y que contienen al menos un lazo o bucle. Estos lazos fueron diseñados como programas "kernel" de algunos de los programas utilizados en el Lawrence Livermore National Laboratory y posteriormente utilizados como benchmarks en otras instituciones [46]. El objetivo propuesto al definir este tipo de nucleos fue el estudio de dependencias entre iteraciones que pertenecían al mismo lazo, mas que ver las dependencias existentes entre iteraciones. A este nivel solamente se necesitan cinco estructuras para ser capaces de clasificar los 24 lazos de Livermore. Las estructuras reconocidas en el conjunto de lazos anteriores son:

a) Estructura de dependencia de datos independiente

b) Estructura de dependencia de datos en árbol

c) Estructura de dependencia de datos recursión lineal

d) Estructura de dependencia de datos conjunto de equivalencia

e) Estructura de dependencia de datos secuencial

A continuación describimos brevemente las secciones de código que representan así como las estructuras de grafo que se obtienen.

a) Independente

Este tipo de estructura se caracteriza por ser cada iteración del lazo independiente de las demás. Por tanto ninguna iteración necesita información de las otras para poder comenzar a ejecutarse.

La codificación en Fortran de este tipo de estructura es la siguiente:

A (K) = B (K) + C (K)

La representación en DAG, puede verse en la figura 3.15. Una tercera parte de los lazos de Livermore ("Kernel" 1, 7, 8, 9, 10, 12, 15, 25) tienen esta estructura.

b) Arbol

Originalmente no ocurre en los lazos de Livermore tal como estaban codificados. No obstante, podemos aplicar alguna modificación para llegar a la misma.

La codificación en Fortran de este tipo de estructura es la siguiente:

```
Q = 0

DO 10 K = 1, 8

10 Q = Q + X (K)
```

Observar que cada iteración es la entrada de la siguiente, suponer un ejemplo que suma ocho elementos de un array; aprovechando la asociatividad de la operación, obtenemos el lazo funcionalmente equivalente transformado. De esta forma el problema puede ser dividido en trozos de tipo secuencial que pueden combinarse utilizando la estructura en árbol.

Q =((((((A + B)+ C)+ D)+ E)+ F)+ G)+ H

Sea: X (1) = A, X (2) = B ... X (8) = H

((A + B) + (C + D)) + ((E + F) + (G + H))

La representación en DAG, aparece en la figura 3.15, los "kernel" 3 y 24 pueden ser transformados en estructuras de árbol. Los "kernel" 4 y 21, pueden ser transformados como una combinación de la estructura independiente y en árbol.

c) Recurrencia lineal

Esta estructura pertenece a las denominadas no vectorizables, un ejemplo de código que la representa se muestra en las siguientes lineas:

A (1) = B (1) y

A (K) = B (K) + A (K - 1) para k 1

En la figura 3.16 se presenta esta estructura en DAG, los "kernel" 5 y 11 tienen esta estructura, también se da secuencialmente en el "kernel" 19 y cinco veces en el kernel 23.

d) Clase de equivalencia

La siguiente estructura se le denomina clase de equivalencia, como la anterior no depende de los valores numéricos de los datos, de hay su definición de estáticas. Cada iteración del lazo utiliza el valor del dato para determinar el miembro correspondiente de una clase de equivalencia. El dato para esa iteración es combinado con el resto de el dato de la misma clase de equivalencia .

Esta estructura viene representada en la figura 3.17, donde la combinación de cada clase de equivalencia tiene la estructura de árbol. Los "kernel" 13 y 14 tienen esta clase de estructura.

e) Secuencial

La última estructura de dependencia de datos para los lazos de Livermore es la estructura de depencia de datos secuencial, la codificación en lenguaje Fortran podría representarse como sigue:

IF X (K) 0

THEN X (K) = F (X (K - 1))

ELSE X (K) = G (X (K - 1))

La función realizada para determinar X (k), es dependiente del valor calculado para X (K - 1), por tanto X (K - 1) necesita ser determinado antes que X (K). Esta estructura de datos es representada en el DAG de la figura 3.18, en general es de las estructuras definidas como no paralelizables. Los "kernel" 16, 17 y 20 tienen esta estructura, los nodos que aparecen en la parte superior e inferior de la figura 3.18 representan nodos de pre/post procesamiento.

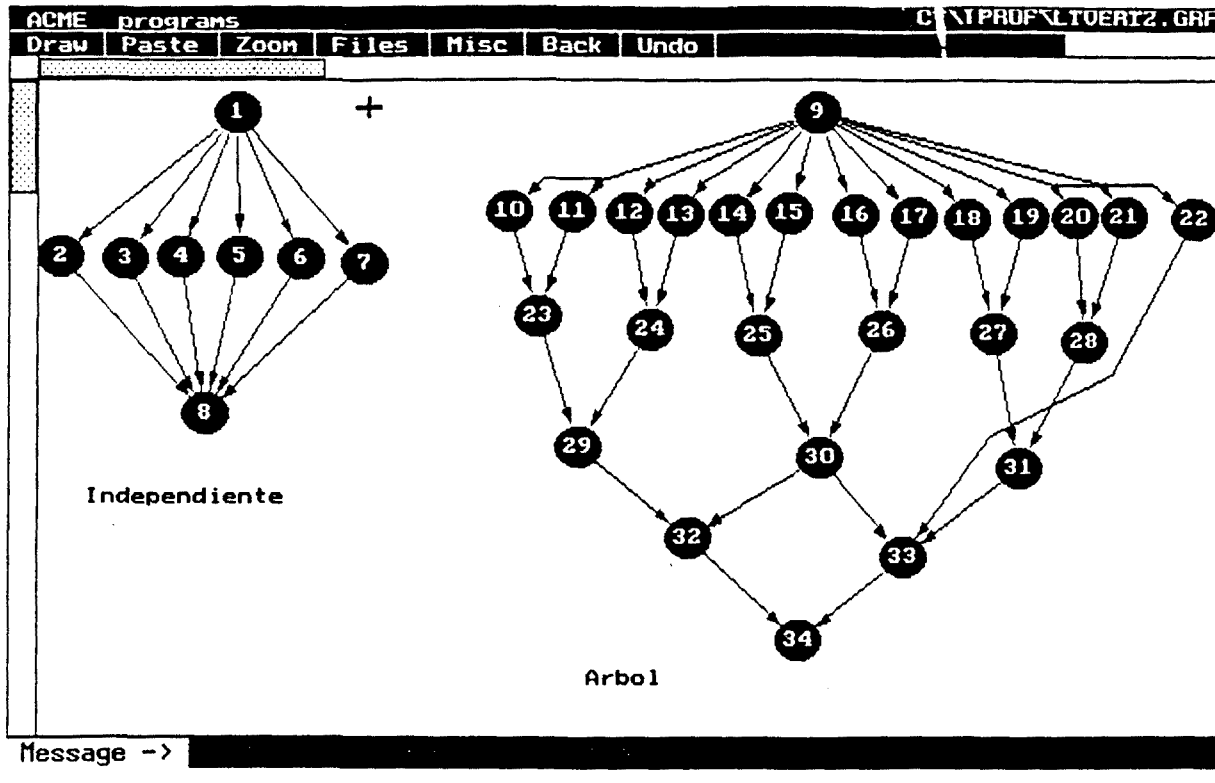


Figura 3.15

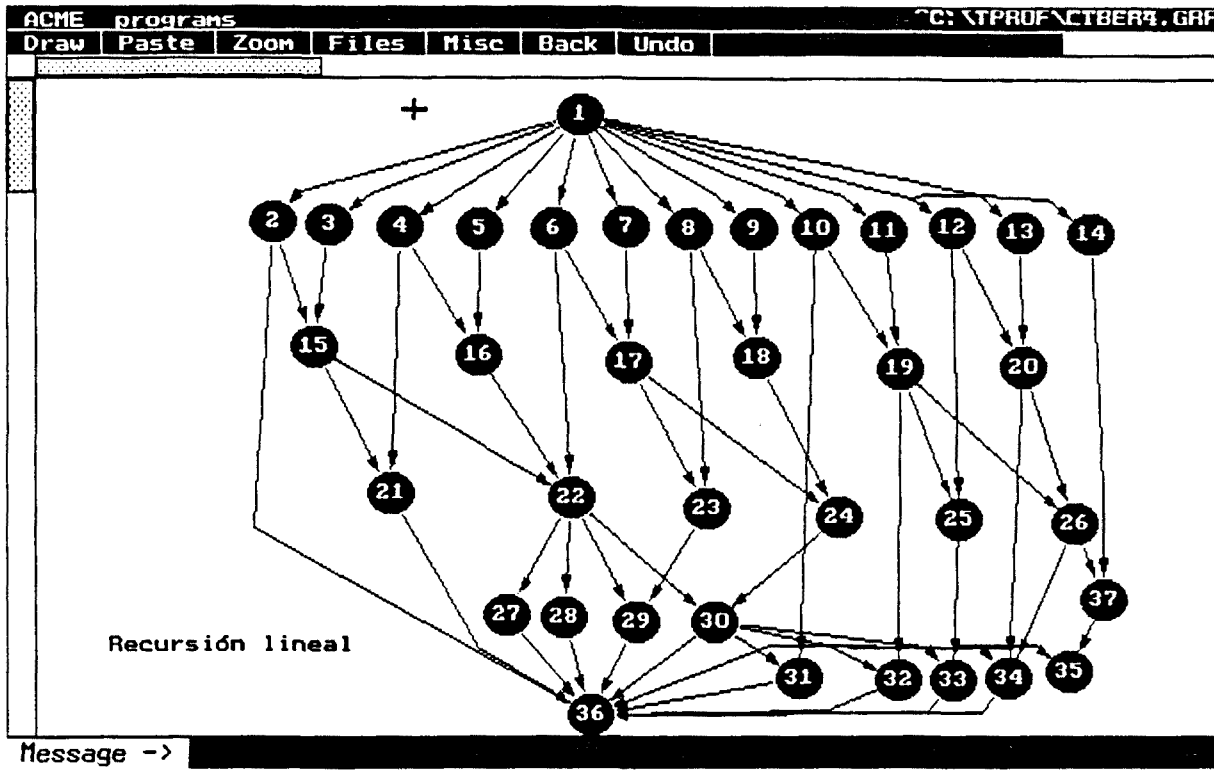


Figura 3.16

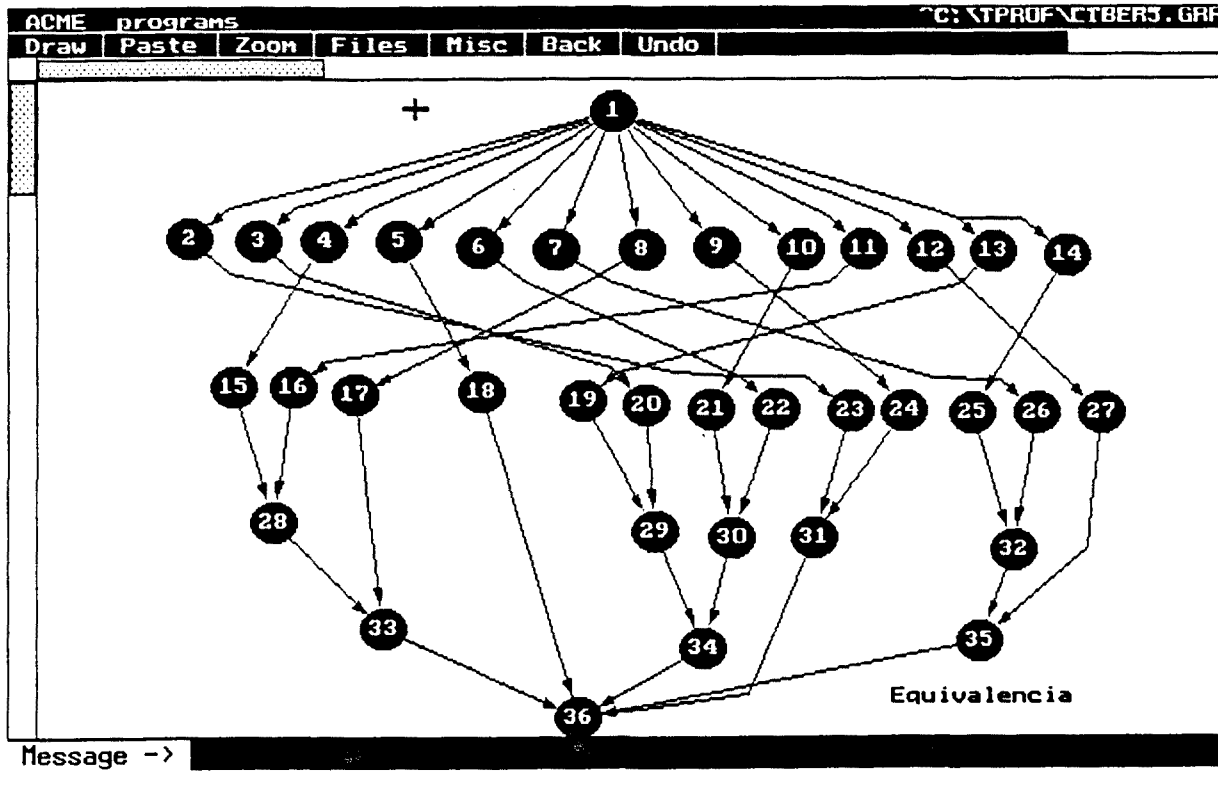


Figura 3.17

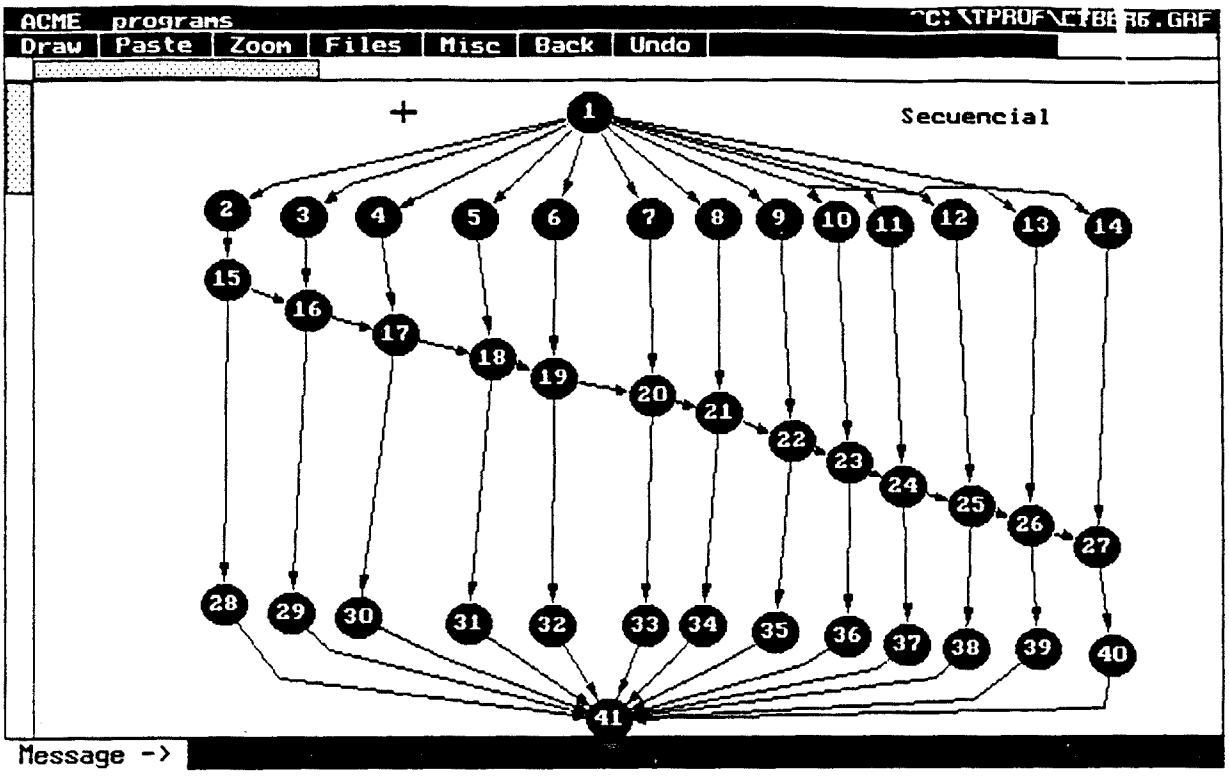


Figura 3.18

3.3.3.1.- Estudio experimental. Análisis de resultados para los grafos construidos a partir de las estructuras de Livermore

Una vez presentados los tipos de grafos sobre los cuales vamos a desarrollar nuestro proceso de evaluación, pasaremos a definir los rangos de valores donde se van a mover los parámetros significativos que constituyen los DAG de las aplicaciones evaluadas. Hecha esta exposición, definiremos el conjunto de medidas de rendimiento que nos servirán como referencia, para finalmente poder extraer de la tabla global de resultados, datos acerca de la bondad de nuestros algoritmos

3.3.3.2.- Parámetros significativos para los DAG analizados

Los resultados experimentales se han obtenido a partir de 100 ejecuciones sobre 10 grafos distintos.

Los parámetros que definen el conjunto de grafos sobre los cuales basaremos nuestras discusiones posteriores se exponen a continuación:

-El conjunto de nodos para todos los grafos es en media de 100 nodos.

-El tiempo de ejecución de las tareas varia de 1 a 100 unidades de tiempo.

-El número de nodos del grafo que pueden variar (incertidumbre considerada) es del 30% del total de los nodos del grafo.

-Los tiempos de variación de estos entre un 10% y un 1000% de el valor inicial.

-El número de procesadores para los cuales se van a estudiar las asignaciones, pueden variar entre 4 y 18.

La topología de los grafos motivo de estudio, se presenta en la figura 3.19, donde se puede identificar las distintas estructuras de Livermore utilizadas para formar el conjunto de grafos.

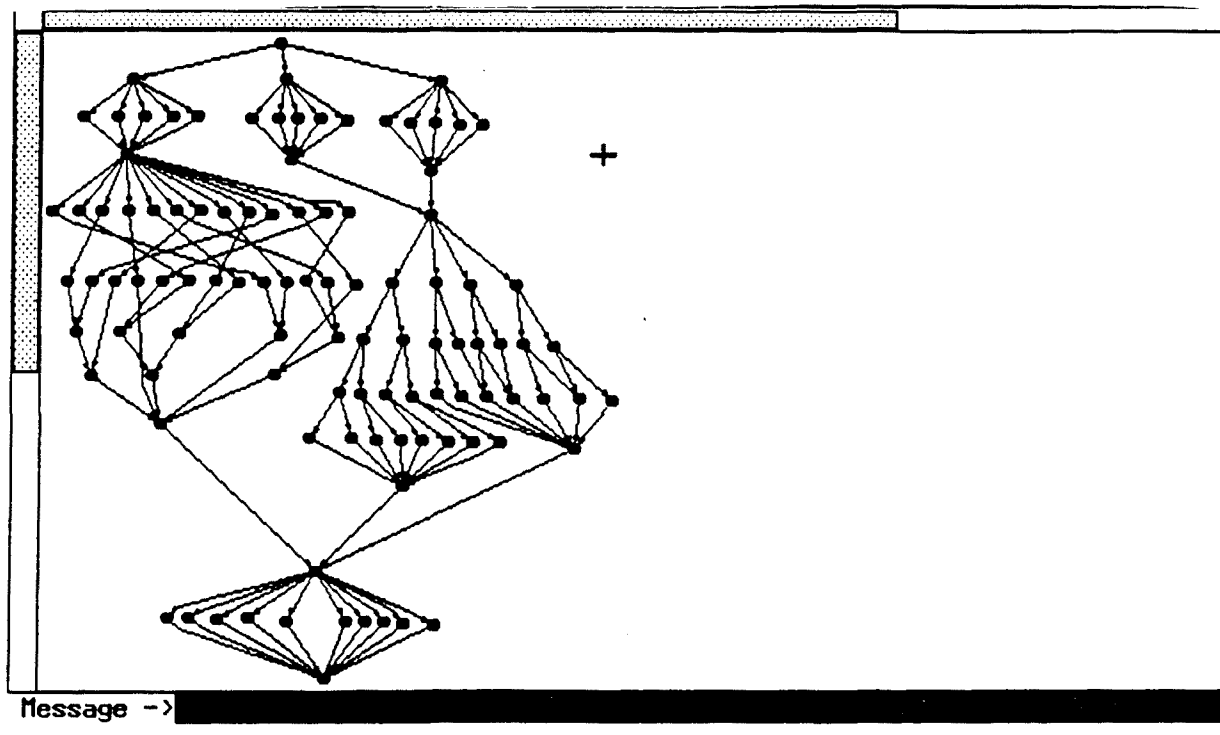


Figura 3.19a

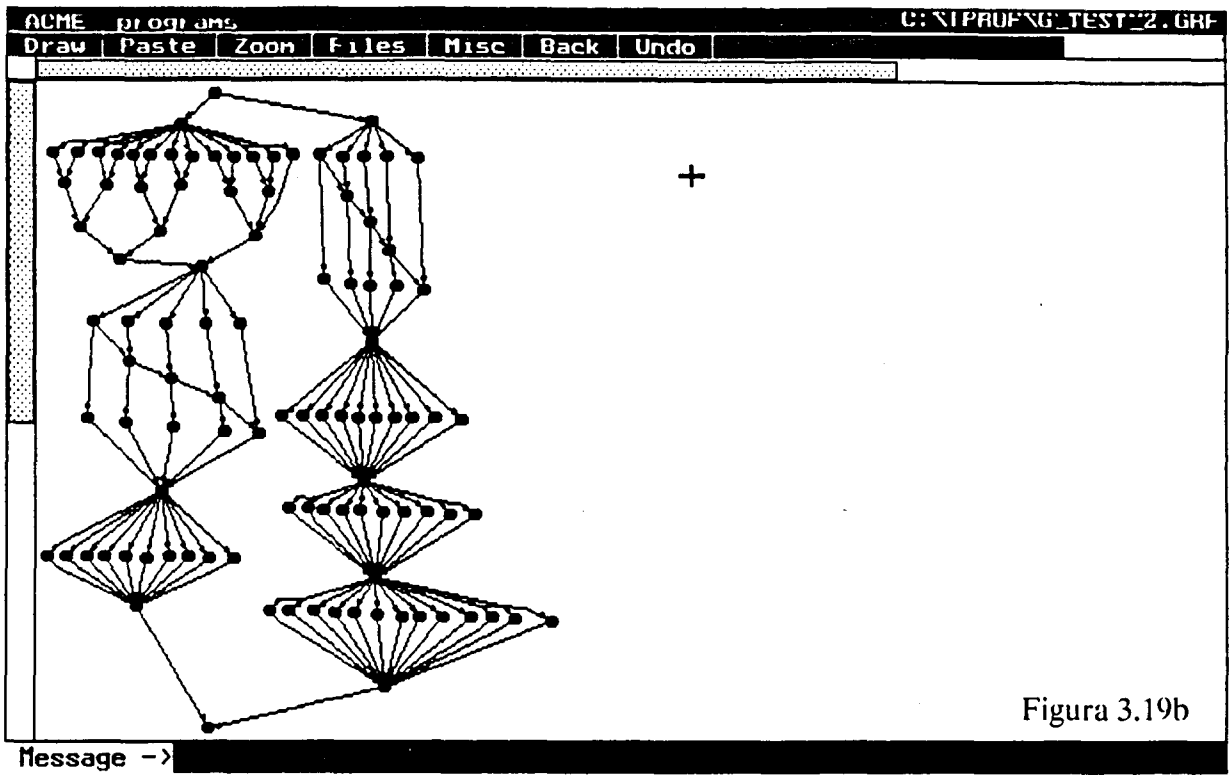


Figura 3.19b

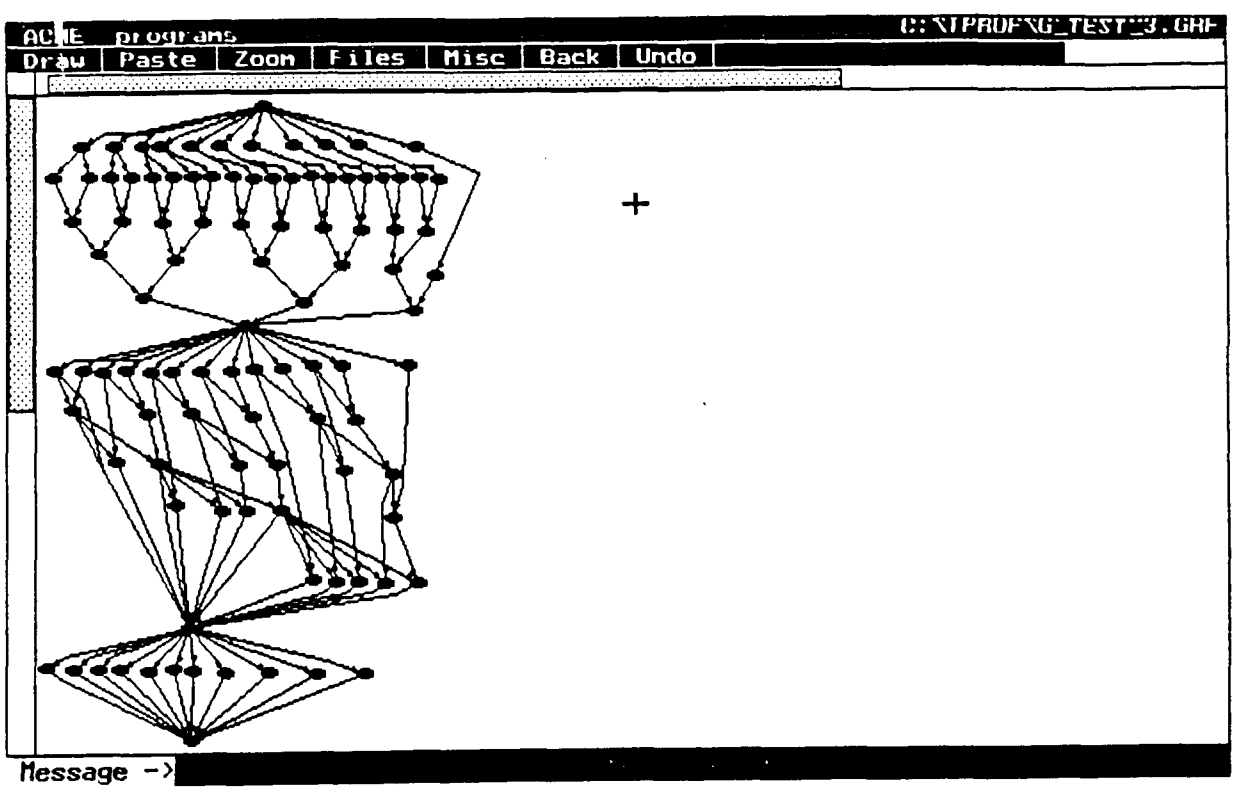


Figura 3.19c

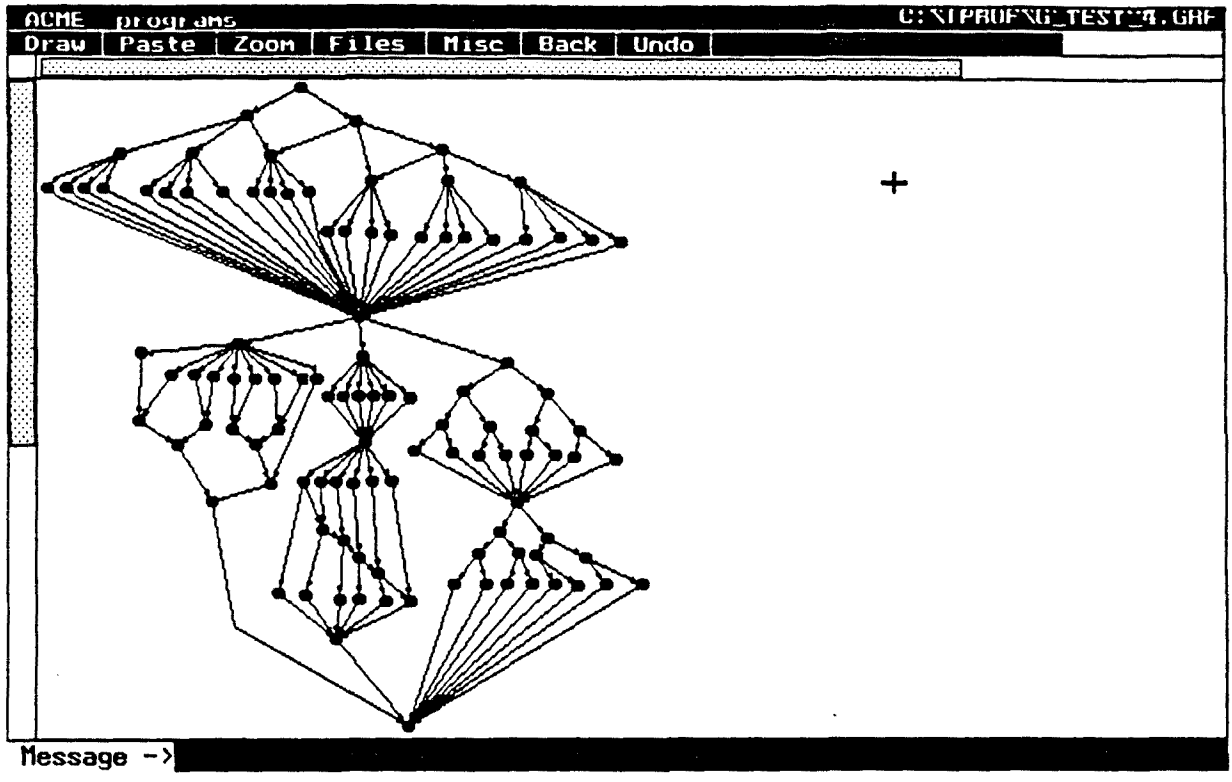


Figura 3.19d

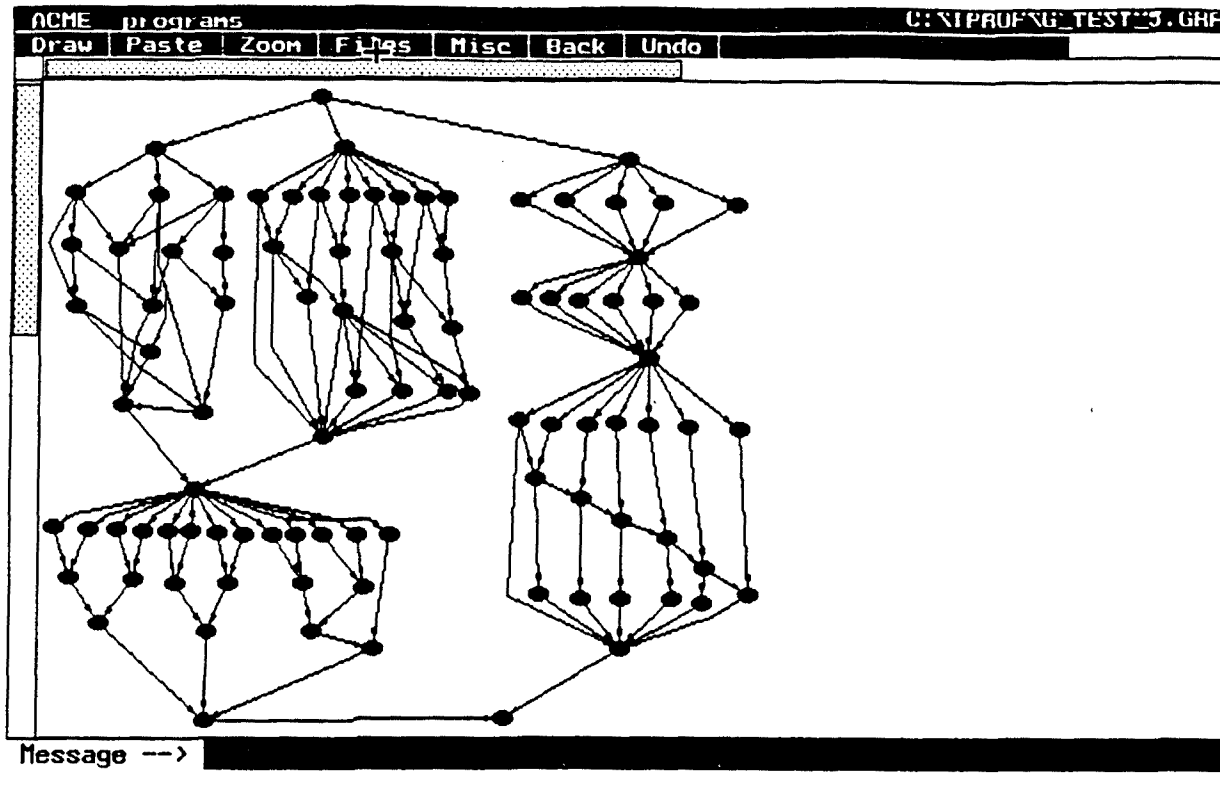


Figura 3.19e

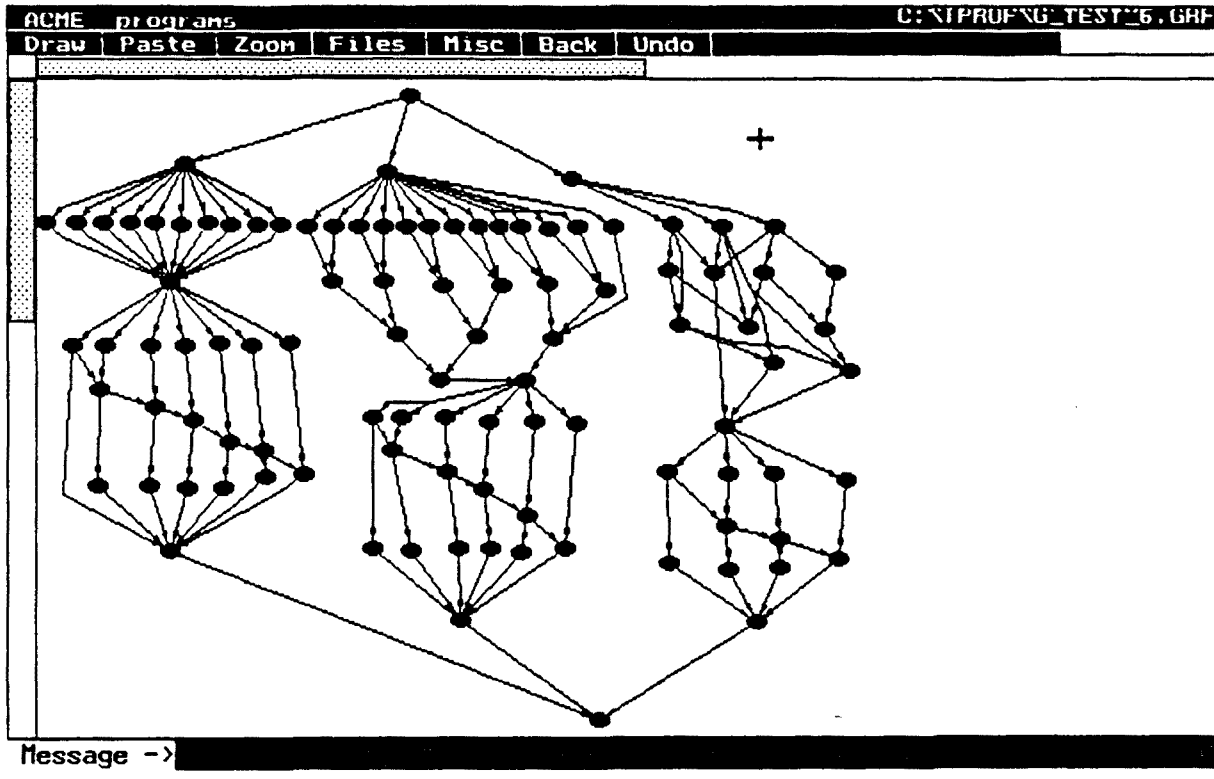


Figura 3.19f

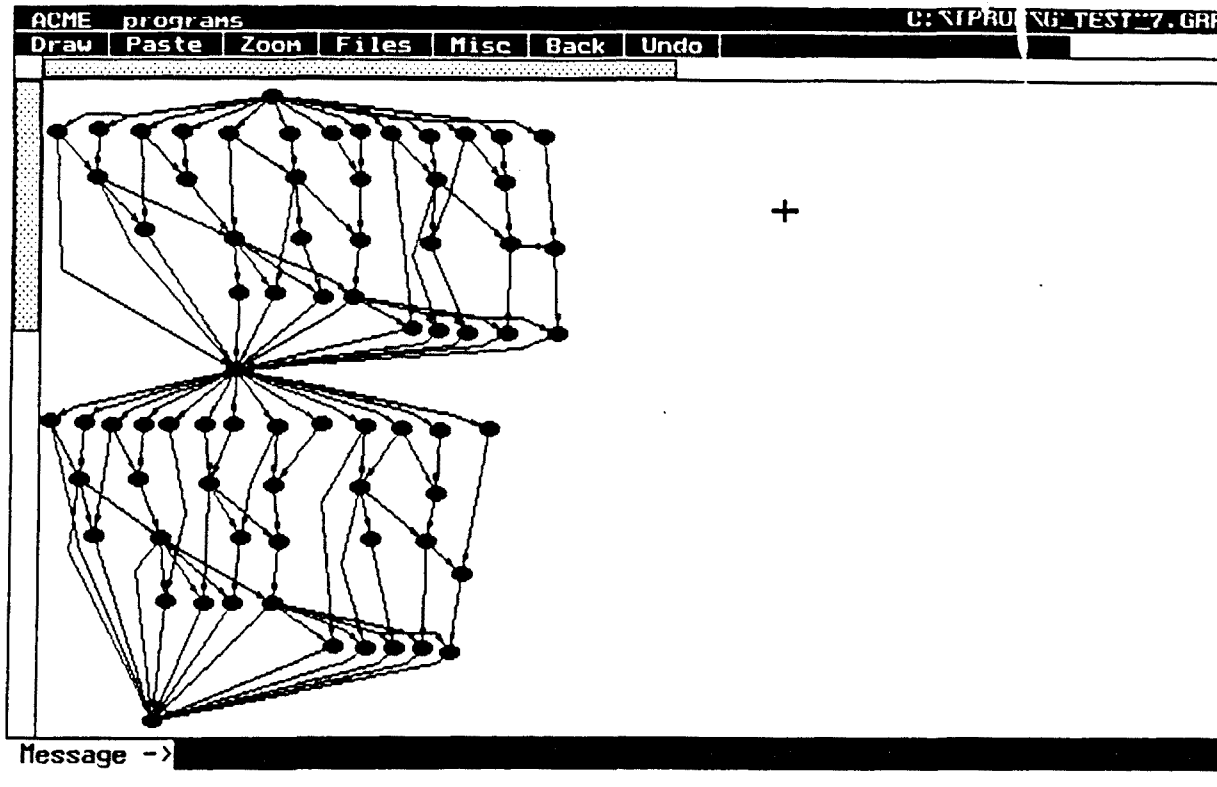


Figura 3.19g

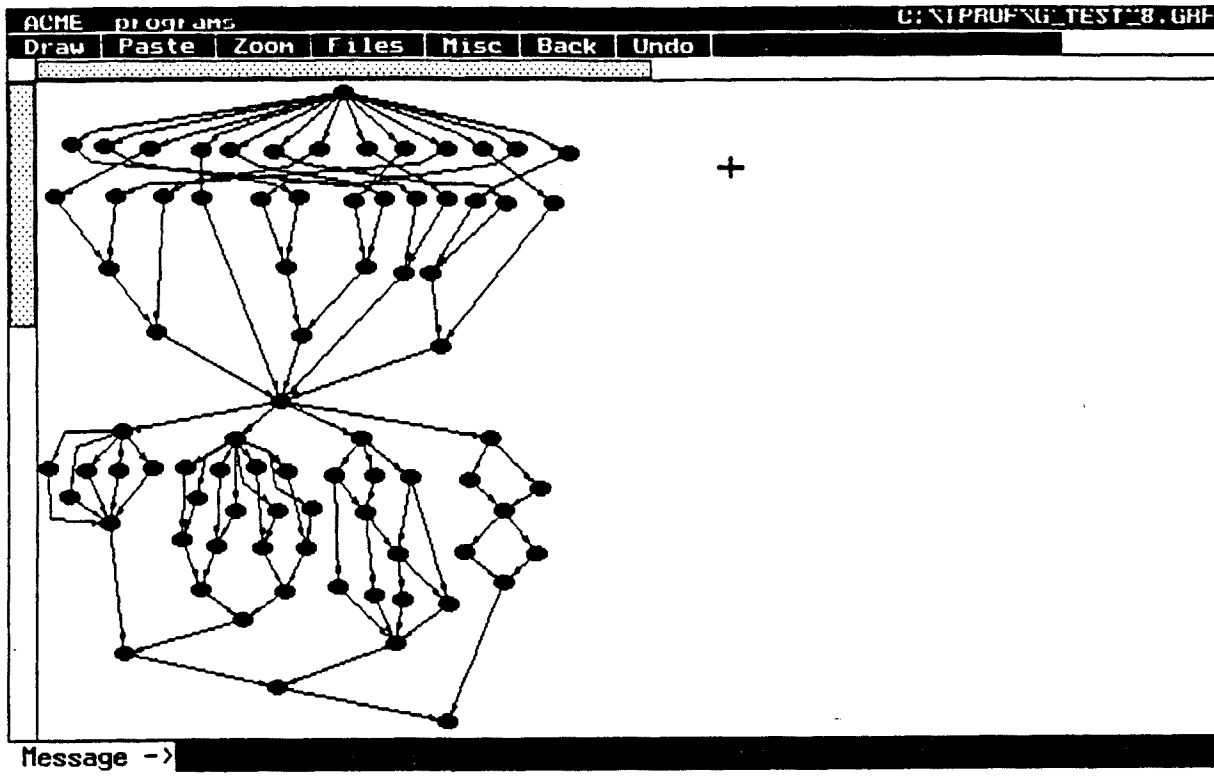


Figura 3.19h

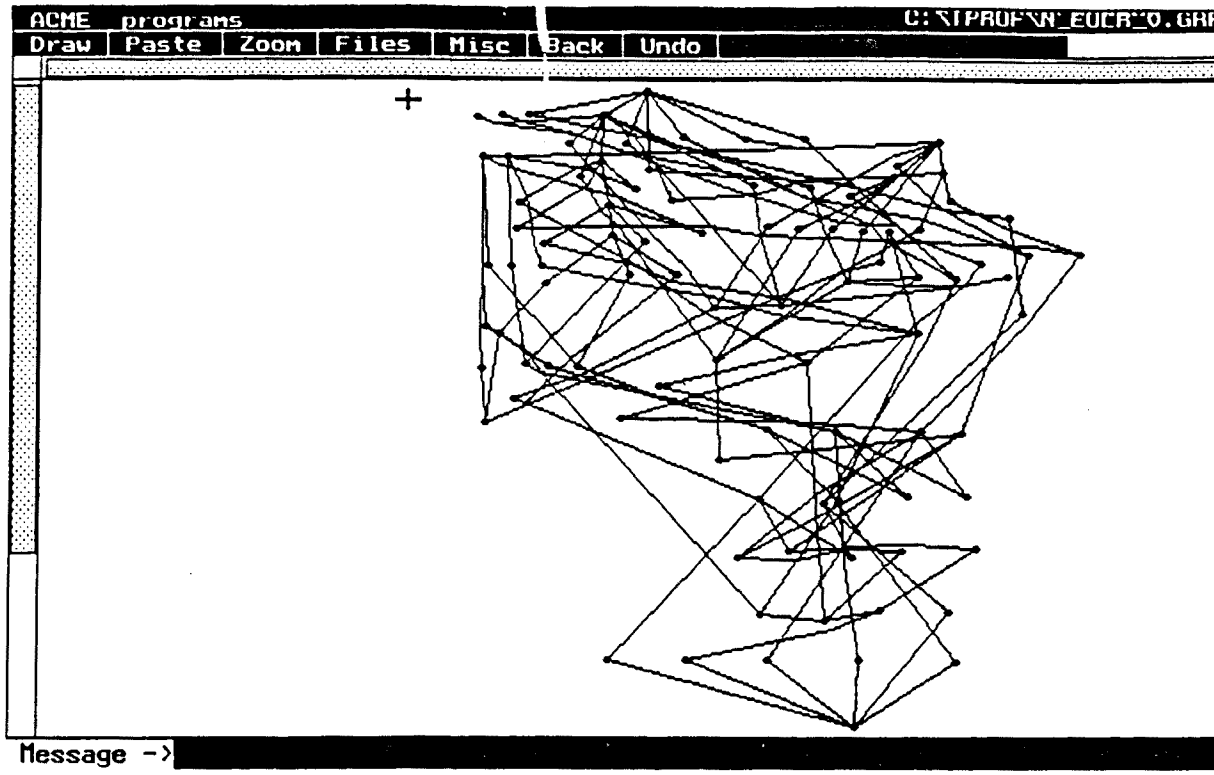


Figura 3.19i

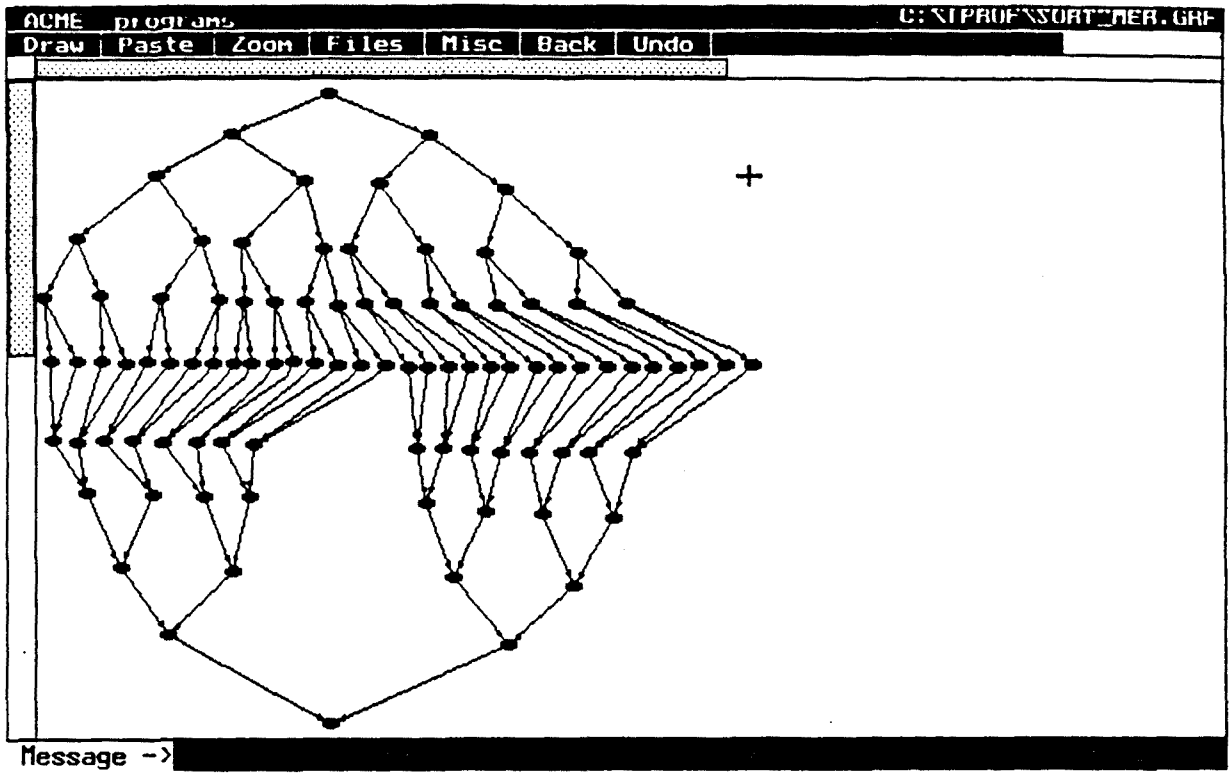


Figura 3.19j

3.4.- Asunciones y planteamientos a la hora de medir

Un problema que se plantea a la hora de establecer la bondad de los algoritmos de planificación desarrollados es, encontrar un punto de referencia válido, ya que no es posible encontrar los valores de referencia óptimos en tiempo polinomial.

Una posibilidad consistiría en comparar los distintos algoritmos entre si y determinar posteriormente cual es el mejor, técnica bastante deficiente ya que como medida relativa podría proporcionarnos para un conjunto de algoritmos, cual es el mejor, pero no nos proporciona la suficiente información para saber si con otro algoritmo distinto podríamos obtener mejores resultados.

Otra posibilidad consiste en evaluar unos determinados índices que representen el máximo o el mínimo que puede tomar una medida, así por ejemplo, el tiempo mínimo que ha de transcurrir para poder ejecutar el grafo con un número de procesadores dados. Una primera cota podría ser, el tiempo del camino crítico como medida del mínimo tiempo de ejecución en el cual es posible ejecutar el grafo, no

obstante gran parte de las veces, esta cota no va a ser alcanzable por no disponer de un número adecuado de procesadores.

A la hora de comparar la bondad de nuestros algoritmos, elegimos compararnos con el CP/MISF, algoritmo que ha probado su validez y ha sido implementado en aplicaciones reales. La literatura muestra que en un 67% de los casos testeados se encontraron soluciones óptimas, el 87% de las soluciones estaban dentro de el margen de error menor que el 5%, y si se relajaba el error al 10%, el 98.5% de las soluciones estaba incluido dentro del margen [12].

3.5.- Mecánica de experimentación

En el presente apartado, pretendemos mostrar la mecánica experimental que hemos seguido para analizar cada grafo, con objeto de obtener las medidas de rendimiento que posteriormente tabularemos y comentaremos. A tal efecto, presentaremos para uno de los grafos utilizados en nuestra investigación, los parámetros evaluados, valores asignados a cada nodo, la sensibilidad introducida y los valores de prestaciones obtenidos.

En la figura 3.20 se muestra la estructura del grafo utilizado, en el cual se pueden identificar algunas estructuras de Livermore utilizadas para su composición; 5 estructuras independientes, una estructura de dependencia de datos en relación de equivalencia y una estructura que representa paralelismo abierto. Los parámetros que caracterizan el grafo y el entorno de computación van a ser los siguientes:

El número de procesadores obtenidos para realizar las medidas, obtenido al aplicar las técnicas de Fernández y Busell [17] ha sido de cuatro. Los tiempos de ejecución utilizados son los Ti, considerados como punto de partida para obtener la asignación inicial proporcionada por el CP/MISF.

El número de nodos del grafo es de 103.

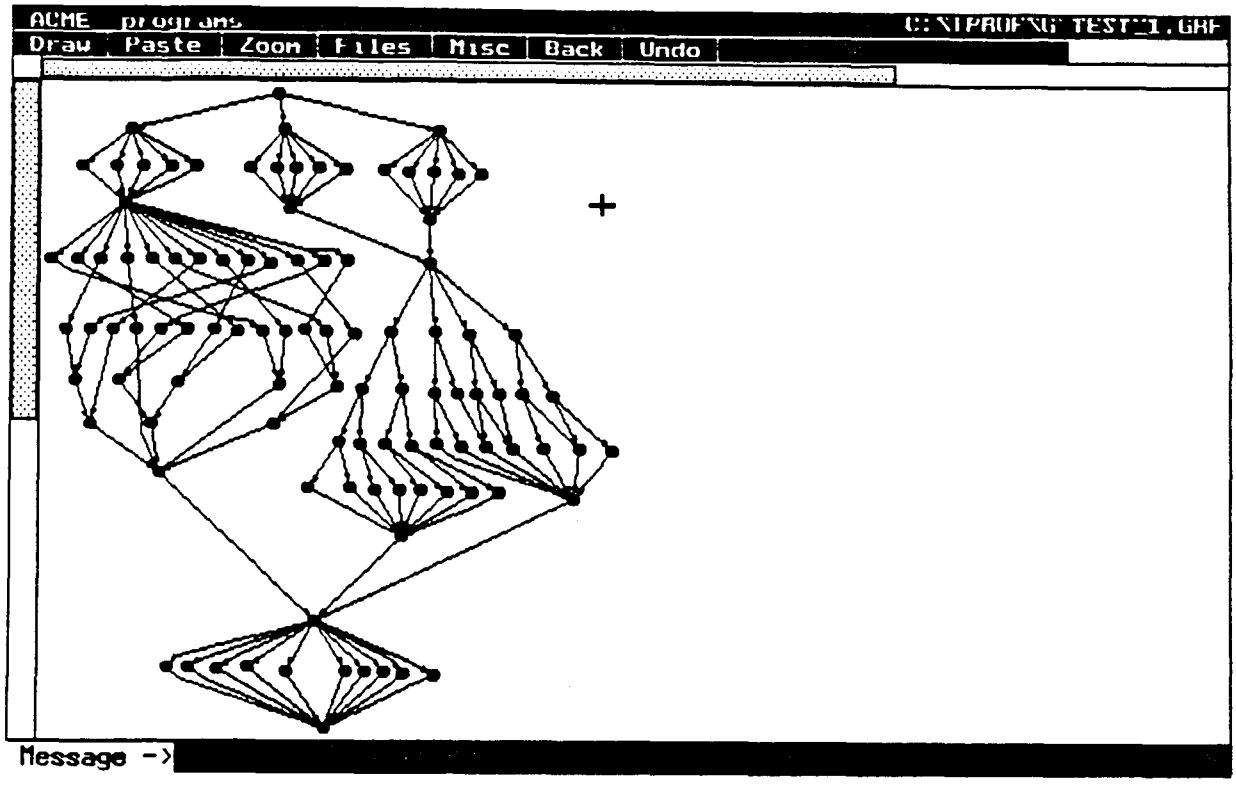


Figura 3.20

El tiempo de ejecución de los nodos viene reflejado en la tabla 3.2, como se observa toman valores entre 1 y 100.

Los nodos considerados como sensibles a los datos de entrada han sido 28 para el ejemplo (27% del total de nodos). Los valores de los nodos se muestran en la tabla 3.3. Para cada uno hemos elegido 10 valores E1-E10, E1 identifica al valor Ti correspondiente, mientras que los valores restantes E2-E10 identificarán 9 valores de Ti*.

El grafo de partida está compuesto por nodos cuyos tiempos pueden considerarse fijos (Ti), y por un conjunto de nodos fuertemente sensibles a los valores de los datos de entrada, cuyos tiempos pueden cambiar entre (Ti) y (Ti*). El punto de partida para obtener la asignación inicial proporcionada por el CP/MISF, con respecto a los nodos que pueden variar, se realiza con el primer valor extremo de cada uno de ellos.

GRAFO G_TEST_1									
Nodo	Ti	Nodo	Ti	Nodo	Ti	Nodo	Ti	Nodo	Ti
1	1	25	1	49	5	73	2	97	20
2	1	26	100	50	4	74	10	98	40
3	1	27	1	51	4	75	4	99	10
4	1	28	10	52	40	76	2	100	1
5	80	29	1	53	6	77	3	101	40
6	20	30	10	54	1	78	7	102	100
7	6	31	2	55	2	79	2	103	60
8	2	32	10	56	20	80	6	104	
9	4	33	6	57	1	81	9	105	
10	80	34	9	58	2	82	8	106	
11	100	35	1	59	2	83	7	107	
12	1	36	7	60	4	84	2	108	
13	100	37	6	61	2	85	1	109	
14	2	38	9	62	2	86	9	110	
15	1	39	8	63	10	87	12	111	
16	100	40	7	64	10	88	29	112	
17	1	41	6	65	9	89	2	113	
18	10	42	1	66	6	90	1	114	
19	24	43	2	67	9	91	6	115	
20	2	44	3	68	4	92	4	116	
21	1	45	6	69	9	93	90	117	
22	9	46	7	70	1	94	100	118	
23	10	47	2	71	2	95	70	119	
24	10	48	30	72	6	96	31	120	

Tabla 3.2

Nodo	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
5	80	20	70	60	50	40	30	80	70	20
6	20	100	30	40	50	60	90	80	70	20
10	80	1	40	10	20	30	40	50	60	70
11	100	20	90	80	70	60	50	40	30	100
12	1	100	10	20	30	40	50	60	70	80
13	100	40	90	80	70	60	50	40	90	100
14	2	70	10	20	30	40	50	60	2	70
16	100	1	90	80	70	60	50	40	30	20
26	100	10	80	70	60	50	40	30	20	10
32	10	40	20	30	40	15	25	35	40	40
33	6	10	6	7	8	9	10	10	9	8
36	7	30	10	15	20	25	30	25	20	15
48	30	50	40	45	50	45	40	35	30	40
52	40	80	50	60	70	80	45	55	65	75
61	2	20	5	10	15	20	15	10	5	2
87	12	10	10	12	10	12	10	12	10	12
88	29	100	29	30	40	50	60	70	80	90
92	4	14	4	5	7	9	11	13	14	14
93	90	1	80	75	70	60	50	40	30	10
94	100	10	90	80	70	60	50	40	30	10
95	70		70	69	68	67	66	65	64	63
96	31	100	40	50	60	70	80	90	100	31
97	20	19	18	17	16	15	14	13	12	11
98	40	20	35	30	25	20	30	40	40	30
99	10	40	15	20	25	30	35	40	35	30
101	40	2	35	30	25	20	15	10	5	2
102	100	40	40	80	70	60	50	40	95	75
103	60	10	50	40	30	10	20	30	40	50
On3		472	406	421	416	396	421	460	534	542
On2		481	404	427	416	397	439	454	534	507
Texec		596	427	444	454	483	527	579	602	564
Cp		446	396	418	407	388	409	404	491	497
Extremos		472	409	425	414	420	486	472	534	516

Tabla 3.3

Nuestro propósito es medir la bondad de los algoritmos anteriormente expuestos (CP/MISF/TD y CP/MISF/TD/RC), frente al CP/MISF en el caso de considerar este último aplicado individualmente para Ti y Ti*; valor que consideramos aceptable, y compararlos igualmente con el tiempo que proporciona el CP/MISF cuando consideramos la asignación que proporciona el CP/MISF para los tiempos de ejecución de los nodos Ti y ejecutamos el grafo para los Ti*. Siendo el índice de rendimiento del mismo, el tiempo de ejecución del grafo, la comparación entre el CP/MISF calculado individualmente tanto para los valores de los nodos Ti, Ti* y entre los algoritmos expuestos en este trabajo, nos da idea de la desviación frente a la cota de óptimo considerada: por otra parte, la comparación de los algoritmos expuestos anteriormente frente al valor obtenido por la asignación proporcionada por el CP/MISF para el primer valor extremo Ti ejecutando para el segundo valor extremo Ti*, nos proporciona una medida tanto de la sensibilidad del CP/MISF frente a la variación de los datos de entrada, como de la ganancia que proporcionan nuestros algoritmos al intentar adaptarse a estas situaciones de posibles cambios en el tiempo de ejecución de algunos nodos.

Con objeto de mostrar y facilitar la interpretación de resultados obtenidos en nuestra experimentación, con el fin de reflejar en tablas y gráficas condensadas el trabajo realizado, a continuación, exponemos el algoritmo de actuación y los resultados obtenidos para uno de los grafos ejemplos:

Paso 1.- Encontrar la asignación para el primer valor extremo Ti. Manteniendo esta asignación, aplicar los algoritmos CP/MISF/TD y CP/MISF/TD/RC ejecutando para los tiempos Ti* (valores que denominaremos On3 y On2 respectivamente).

Paso 2 .- Igualmente, encontrar la asignación que proporciona el CP/MISF para los valores extremos Ti*, y ejecutar para los tiempos del segundo valor extremo Ti* (valor que denominaremos CP).

Paso 3 .- Encontrar la asignación que proporciona el CP/MISF para los tiempos de ejecución Ti, y ejecutar considerando los tiempos de ejecución Ti* (valor que denominaremos Texec.).

Los resultados obtenidos para el ejemplo de la figura 3.20, se muestran en la tabla 3.3, donde E2-E10 representan los 9 valores para Ti* que han sido tomados en las 10 ejecuciones de este grafo.

La tabla 3.4, muestra los valores promedios obtenidos para las 10 ejecuciones del grafo G_test_1 presentado en la figura 3.20, así como los valores promedio de ganancia obtenidos y los valores promedio de desviación. A fin de condensar en una única tabla y gráfica los valores de ganancia y desviación para cada grafo, consideraremos los valores medios obtenidos de cada una de las medidas anteriormente expuestas. Así, los valores netos y porcentuales de ganancia y desviación para el ejemplo considerado se muestran en la tabla 3.4.

Nombre del Grafo	CP/MISF/TD				CP/MISF/TD/RC			
	Ganancia Media	G% On3	Desviacion Media	D% On3	Ganancia Media	G% On2	Desviacion Media	D% On2
G_test_1	91	75	24	5.6	69	76	22	5.1

G% On3: Ganancia relativa del algoritmo CP/MISF/TD
D%On3: Desviacion relativa del algoritmo CP/MISF/TD
G% On2: Ganancia relativa del algoritmo CP/MISF/TD/RC
D%On2: Desviacion relativa del algoritmo CP/MISF/TD/RC
La desviacion maxima relativa al algoritmo CP, es 21% para el ejemplo mostrado

Tabla 3.4

La técnica expuesta anteriormente, se aplica para cada uno de los ejemplos presentados en este trabajo, el resultado de la experimentación realizada se resume en la tabla 3.5, donde se muestra para cada uno de los grafos ejemplo, los valores de ganancia y desviación media. En la figura 3.21, se observan las gráficas de rendimiento relativas, que proporcionan nuestros algoritmos. En la tabla 3.6, se presentan los resultados porcentuales globales de ganancia respecto a las cotas expuestas anteriormente.

Nombre del Grafo	CP/MISF/TD		CP/MISF/TD/RC		Ganancia maxima (reduccion maxima)
	Ganancia Media	Desviacion Media	Ganancia Media	Desviacion Media	
G_test_1	68	23	69	22	91
G_test_2	0	0	0	0	0
G_test_3	8	7	11	4	15
G_test_4	20	35	10	45	55
G_test_5	59	-20	59	-20	39
G_test_6	67	39	66	40	106
G_test_7	52	14	46	20	66
G_test_8	69	34	68	35	103
G_test_9	93	28	93	28	121
G_test_10	36	110	53	93	146
Medio	49	27	48	27	74

Tabla 3.5

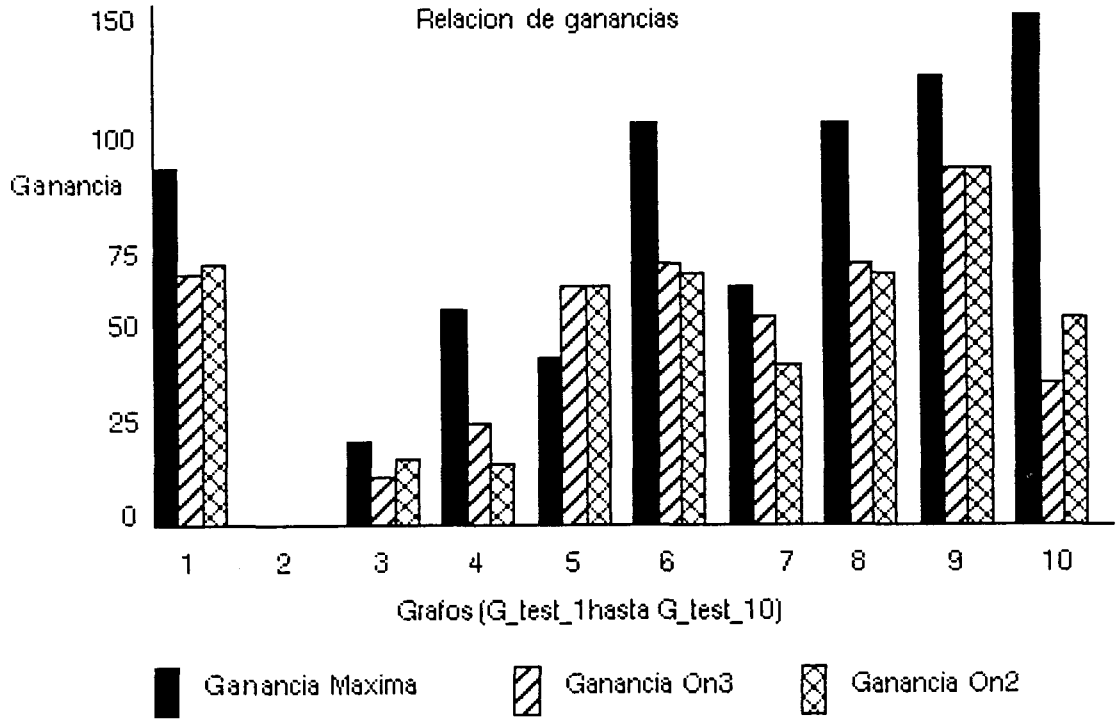


Figura 3.21

	% SOBRE GANACIA MAXIMA	
Grafo	CPMISF/TD	CP/MISF/TD/RC
G_test_1	75%	76%
G_test_2	0%	0%
G_test_3	53%	73%
G_test_4	36%	18%
G_test_5	151%	151%
G_test_6	66%	62%
G_test_7	79%	70%
G_test_8	67%	66%
G_test_9	77%	77%
G_test_10	38%	36%
Media	64%	63%

Tabla 3.6

Como puede deducirse de la tabla 3.6, resumen de los experimentos realizados, el algoritmo CP/MISF/TD es capaz de obtener una ganancia promedio sobre la ganancia máxima (reducción máxima del tiempo de ejecución) del 63.9%, igualmente, el algoritmo CP/MISF/TD/RC, obtiene una ganancia promedio del 62.9% sobre el valor promedio de ganancia máxima; no obstante, los valores promedios en este caso ocultan alguna característica importante que conviene comentar, y que puede ser deducida de los resultados individuales obtenidos que se muestran en el apéndice B, para los grafos en estudio. Para estos, se puede apreciar, que los valores de ganancia y desviación individuales, para cada conjunto de tiempos de ejecución, suele existir mayor desviación para el algoritmo CP/MISF/TD/RC que para el algoritmo CP/MISF/TD, lo que produce que las mejoras y pérdidas en promedio se compensan. En resumen, para todos los casos en general el algoritmo CP/MISF/TD proporciona unos resultados mejores que el algoritmo CP/MISF/TD, pero para algunos casos en concreto las desviaciones para algunos valores son mucho mayores para el CP/MISF/TD/RC; el resultado es bastante lógico ya que no se realiza ninguna comprobación por simulación cuando se duplica una tarea.

CAPITULO 4. APROXIMACION DEL MODELO DE DAG A LA SITUACION REAL. ADAPTABILIDAD DE LOS ALGORITMOS DE DUPLICACION

- Introducción

Uno de los aspectos más interesantes a considerar en el estudio del cómputo distribuido, consiste en la búsqueda de modelos a partir de los cuales sea posible predecir el rendimiento del sistema computacional.

Parte del trabajo expuesto en esta memoria, es un intento de desarrollar una cierta extensión al modelo de DAG, con objeto de aproximar este modelo de representación computacional. Como presentamos en el capítulo 3, nuestra propuesta iba dirigida, a la extensión del modelo de DAG con objeto de poder representar de manera más precisa, la incertidumbre que implica modelizar los tiempos de ejecución de código de tipo iterativo y condicional, a fin de aproximarnos al comportamiento real de los programas. Esta aproximación consistía en representar este tipo de secciones, mediante dos valores determinísticos, que denominábamos valores extremos, y utilizar la técnica de duplicación de tareas con objeto de minimizar el tiempo total de la aplicación. En el presente capítulo, pretendemos formular una nueva extensión al modelo propuesto, para aquellos nodos del grafo de tareas cuyo tiempo de cómputo pueda variar en ámplios márgenes de tiempo, considerando ahora, que los mismos, pueden tomar tiempos de cómputo dentro de un rango de valores. La adaptación a la nueva situación de los algoritmos de duplicación desarrollados anteriormente que modelan este tipo de secciones con dos valores extremos, la investigación de nuevos métodos que nos permitan resolver el problema

planteado, así como la experimentación realizada, constituyen el nucleo central de este capítulo.

A lo largo de la exposición realizada en este trabajo, el modelo computacional utilizado ha estado basado en los DAG, donde los vértices representan los tiempos de computación estimados, para las transformaciones de la información llevadas a cabo en cada uno de los nodos y los arcos las relaciones de precedencia entre los distintos nodos. En este tipo de modelos, una de las principales dificultades, surge al intentar representar de forma más aproximada los tiempos de ejecución de las estructuras de tipo cíclico y condicional que presentan los programas reales, con objeto de adecuarlas al modelo de representación de DAG.

Con objeto de poder aplicar y/o extender métodos de "scheduling" previamente desarrollados, cuyo modelo para representar los programas está constituido por un DAG, diversos autores [47,48,50,51] han desarrollado métodos específicos para reducir los modelos generales para representar programas (grafos cíclicos, grafos con tiempos variables asociados a los nodos) a modelos representados mediante DAG's.

También el problema de la "no consideración de las comunicaciones", mediante una adecuada selección del nivel de granularidad (granularity selection) y la "adición" de tiempos de ejecución en los nodos, ha sido estudiado, y constituye una cierta extensión (al menos en su uso y alcance para modelado) del modelo original del DAG.

- Martin, Estrin [50] y Elmarghraby [51], partiendo de aplicaciones reales, representadas por grafos dirigidos cíclicos, de granularidad fina para los nodos (donde las secciones de código están constituidas como máximo por una función del tipo raiz cuadrada, exponenciación, logaritmo, etc.), desarrollan una metodología con objeto de disminuir los tiempos de cómputo que implicaba la búsqueda de las posibles longitudes de todos los caminos, a fin de que la extracción de las medidas de rendimiento del sistema computacional, no supusiese un tiempo de cálculo excesivo. Con este objetivo, introducen reglas de sustitución, que permiten reducir el tipo de estructuras cíclicas y condicionales del grafo de la aplicación a nodos de un grafo

DAG; manteniendo la estructura o forma del grafo en el caso de Martín y Estrin o generando desde el punto de vista topológico, otro grafo de apariencia distinta en el caso de Elmarghraby.

- Otra extensión al modelo de DAG, con objeto de su adaptación a la situación real es estudiada por Kasahara y Narita [47,48] donde su objetivo radica principalmente, en encontrar una buena relación entre el volumen de cómputo de las tareas que componen el grafo y las comunicaciones generadas. Así, adaptan el modelo de DAG buscando, en base a la arquitectura donde se va a ejecutar la aplicación, la manera de considerar en el modelo las comunicaciones de forma implícita por la adecuación de los volúmenes de cómputo frente a las comunicaciones.

Otra de las características que se puede observar en sus trabajos, es la forma de elegir el tiempo de ejecución que va a representar el conjunto de nodos del grafo.

En los grafos que representan las aplicaciones por ellos implementadas, los distintos nodos que componen el grafo, representan operaciones de sumas, multiplicaciones en punto flotante, etc., operaciones que dependiendo de los datos de entrada, van a proporcionar un valor distinto dependiendo de estos valores en ejecución, pero que en promedio, no van a representar desviaciones importantes entre si. Por tanto, a la hora de elegir un valor representativo, se deciden por tomar el valor promedio como el valor de tiempo de ejecución que después utilizarán como valores de ejecución estimados a la hora de aplicar sus algortimos de "scheduling".

En los programas de aplicación pueden observarse dos tipos de dependencias frente a los datos; una primera que haría referencia a aquellas secciones cuya dependencia es pequeña, por lo que en principio en su modelado podría considerarse como representativo el tiempo promedio; y otro conjunto de secciones de código que incluyen estructuras que pueden hacer que las variaciones en el tiempo de ejecución de las mismas sean muy importantes. Mientras que las primeras podrían ser modeladas, en cuanto a su volumen de cómputo, por el valor medio de su tiempo de ejecución, dada la existencia de un pequeño valor de desviación, esto no ocurriría en

las segundas, las cuales habrían sido caracterizadas numéricamente mediante un valor medio y una importante magnitud en la desviación.

Esta caracterización no permitiría modelar su comportamiento (el de estos nodos con tiempos fuertemente variables) mediante su valor medio, por cuanto éste no identificaría comportamientos reales del mismo. Nosotros proponemos para el modelado de este tipo de secciones, la adopción de una pequeña extensión al modelo que consistiría en la representación del volumen de ciertos nodos, mediante dos valores extremos: máximo y mínimo. Esta extensión, se aplicaría a aquellos nodos cuyo tiempo de cómputo pueda variar en ámplios márgenes, y por tanto no fuesen susceptibles de ser correctamente modelados mediante la simple asignación, de un tiempo medio de ejecución.

Nosotros consideramos DAG, donde ya implícitamente han sido aplicadas las transformaciones necesarias en el grafo problema, con objeto de considerar la relación costo de cómputo/costo de comunicación adecuada a la arquitectura donde se ejecutará la aplicación; esto supondrá, no tener expresadas de forma explícita en el DAG los costos de comunicación.

Una vez que hemos considerado la extensión propuesta al modelo de DAG, la modelización de las secciones de código fuertemente sensibles, mediante dos valores extremos, (lo que en realidad correspondería a un cierto rango), la pregunta que cabe realizarse, es ¿ siguen siendo válidos los algoritmos de "scheduling" con duplicación de tareas propuestos, cuando se extiende el modelo original para el que fueron concebidos?. Intentaremos contestar a esta pregunta en los apartados siguientes.

4.1.- La representación de secciones de código cuyo tiempo de ejecución puede variar dentro de un rango de valores

El nuevo problema planteado, reside, en encontrar una única asignación para la aplicación desarrollada, que sea representativa de todo el conjunto de posibles grafos que podrían obtenerse, al hacer variar dentro de un rango, el tiempo de

ejecución para aquellos nodos del grafo cuyos tiempos de ejecución dependen fuertemente de los datos de entrada; de tal manera, que los tiempos de ejecución que se obtengan para la aplicación, estén lo más cercanos posible, a los tiempos de ejecución que se obtendrían, aplicando el algoritmo CP/MISF de forma individual, a cada uno de ellos.

4.1.a.- Generalización de los métodos de duplicación de tareas: Replicación

Si se extiende el modelo de representación parece natural intentar extender, en una linea similar, los algoritmos de "scheduling" desarrollados, por tanto extensión en el rango, significará la extensión de los mecanismos de duplicación mediante la acumulación de duplicaciones: Replicación de tareas.

A tal efecto, el método que proponemos en este apartado, se basa en modificar, los algoritmos de duplicación de tareas, con objeto de adaptarlos para el nuevo tipo de entorno donde han de ser aplicados. La solución propuesta en este sentido, consiste, en generar todo el conjunto de grafos posible que puede ser obtenido al hacer variar el conjunto de tiempos de ejecución correspondientes a los nodos fuertemente sensibles a los datos de entrada, con objeto de encontrar la asignación de tareas a procesadores, mediante la aplicación de nuestros algoritmos de duplicación de tareas, acumulando, para cada posible combinación de tiempos de ejecución de los nodos fuertemente sensibles, el conjunto de duplicaciones a que da lugar la aplicación individual de los algoritmos de duplicación de tareas.

4.1.a.1.- A continuación se muestra la metodología a seguir a la hora de obtener la asignación final con duplicaciones representativa de la "familia de grafos" obtenida, al hacer variar los distintos tiempos de ejecución de los nodos que en ejecución pueden tomar distintos valores.

Paso 1: Generar el primer grafo posible (combinación), este, representa uno de los grafos que podría aparecer en la ejecución real de la aplicación. Un estudio exhaustivo por tanto, nos obliga, a la generación de todos los posibles grafos de la familia. La tabla 4.1, muestra la familia de grafos generada (por la posible variación

de los tiempos de ejecución de los nodos (3, 9, 11, 12, y 15) para el grafo de la figura 4.1, en la cual se observa, el conjunto de posibles valores de tiempos de ejecución.

Paso 2: Aplicar el algoritmo CP/MISF/TD para la primera combinación (la que se obtiene a partir de los valores inferiores extremos de tiempos de ejecución de los nodos que pueden variar [1,3,6,2,2], en la tabla 4.1) correspondientes a los nodos (3,9,11,12,15) y obtener la asignación incluyendo las posibles duplicaciones, y el tiempo de ejecución de la aplicación.

Paso 3: Mientras se puedan generar combinaciones (288, para el caso de la figura 4.1), volver a aplicar el algortimo CP/MISF/TD, para otra combinación de tiempos de ejecución de los nodos variables, manteniendo la duplicación/es proporcionada/s por las combinación/es anterior/es. Este proceso se repite hasta alcanzar la combinación de los valores extremos de tiempo de ejecución correspondiente a los nodos variables [2,6,1,4,1].

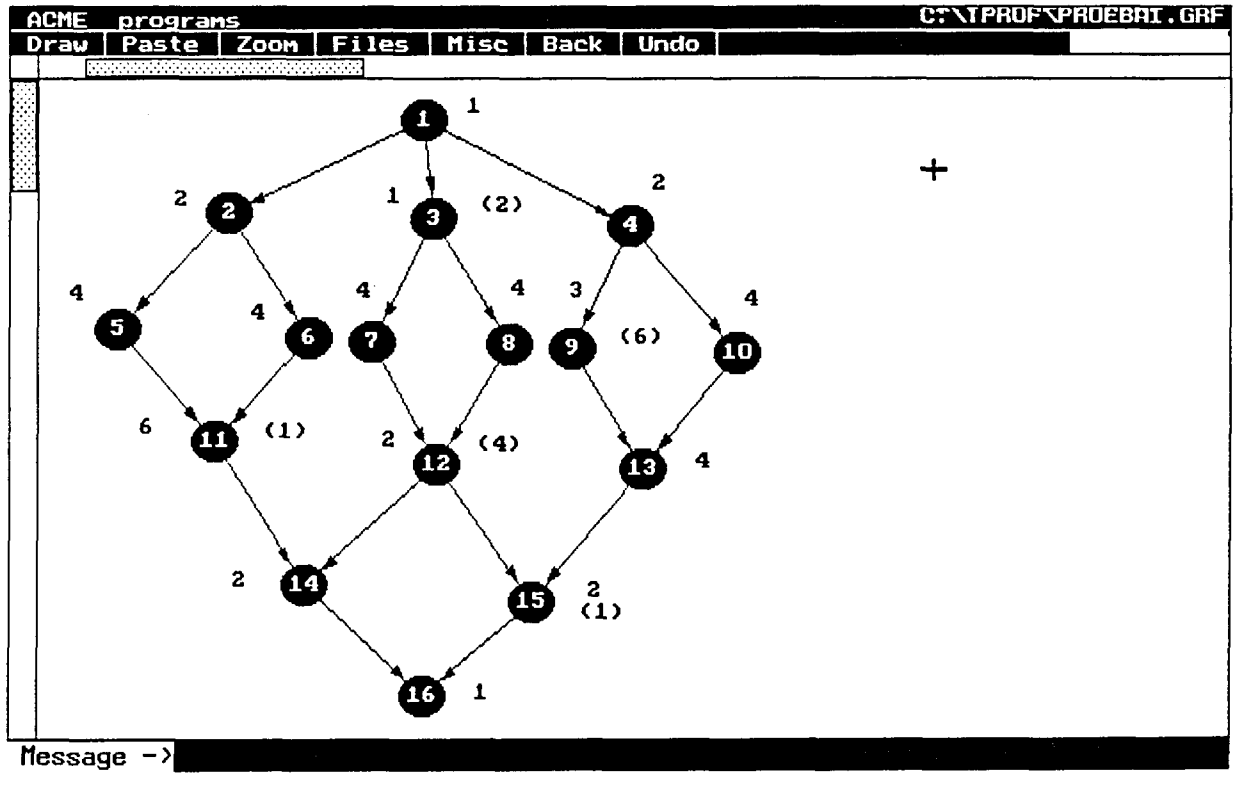


Figura 4.1

GRAFO PRUEBA1						
NODOS					DESVIACION "A"	DESVIACION "B"
3	9	11	12	15		
1	3	2	4	2	2.0000000000E+01	1.3333333333E+01
1	3	2	4	1	1.3333333333E+01	6.6666666667E+00
1	3	1	2	2	2.1428571429E+01	2.1428571429E+01
1	3	1	2	1	1.4285714286E+01	1.4285714286E+01
1	3	1	3	2	2.1428571429E+01	2.1428571429E+01
1	3	1	3	1	1.4285714286E+01	1.4285714286E+01
1	3	1	4	2	2.8571428571E+01	2.1428571429E+01
1	3	1	4	1	2.1428571429E+01	1.4285714286E+01
1	4	6	2	2	0.0000000000E+00	0.0000000000E+00
1	4	6	2	1	0.0000000000E+00	0.0000000000E+00
1	4	6	3	2	0.0000000000E+00	0.0000000000E+00
1	4	6	3	1	0.0000000000E+00	0.0000000000E+00

Tabla 4.1

Paso 4: Obtener la asignación final obtenida mediante la acumulación de duplicaciones y el tiempo de ejecución final.

Pronto se percibe la complejidad que implica la experimentación que se desearía llevar a cabo. Al intentar analizar la familia de grafos resultante de combinar todos los posibles valores que pueden tomar cada nodo en particular, se observa, la naturaleza intratable del problema en cuestión. Así por ejemplo, cuando el número de nodos que pueden variar es de 10, y el rango de posibles valores de tiempo de ejecución para cada uno es igualmente de 10, ([1, 10]), la familia de grafos a analizar sería de 10^{10}. Si consideramos Ti^u, el límite superior del intervalo para el nodo i-esimo, y Ti^l, el límite inferior del intervalo para el nodo i-esimo, el número posible de combinaciones a generar con un intervalo unidad para definir los distintos elementos del rango podríamos expresarlo con la siquiente expresión :

Combinaciones $= \prod_{i=1}^{z} (Ti^u - Ti^l + 1)$, donde z, es el conjunto de nodos que pueden variar.

Teniendo en cuenta, que el posible conjunto de nodos susceptibles de variación en los grafos que pretendemos analizar es del orden de 30 nodos, y los intervalos de variación para los Ti entre $Ti^1 - Ti^{100}$, donde $(Ti^l < Ti < Ti^u)$ para todos los intervalos, el número de combinaciones a analizar sería de 100^{30}, lo que hace el problema intratable incluso para sistemas con una elevada potencia de computación.

4.1.b.- Extensión de los métodos de duplicación a un rango de valores

Si consideramos que ya inicialmente al representar el volumen de cómputo de un nodo mediante 2 valores estábamos asumiendo un "cierto tipo de rango de valores"en su volumen de cómputo, entonces podemos probar a "extender" el rango de aplicación del algoritmo desarrollado mediante su aplicación al caso de definir un auténtico rango, para caracterizar el volumen de cómputo de un nodo.

La segunda aproximación por tanto, se orienta a la obtención de las medidas de rendimiento que proporcionan los algoritmos propuestos, manteniendo el método de duplicación de tareas para 2 valores, que ahora corresponderán a los valores extremos del rango de variación, cuando consideremos como tiempos reales de ejecución, diferentes valores del rango comprendidos entre los valores extremos considerados.

4.1.1.- Estudio realizado para analizar la familia de grafos

El análisis correspondiente a la validez de las dos aproximaciones propuestas se ha realizado mediante la correspondiente experimentación. El estudio experimental realizado permite además comparar las prestaciones ofrecidas por cada una de las dos propuestas de extensión al caso de considerar un "rango de valores".

Debido a la complejidad temporal que implica el analizar todo el conjunto de posibles combinaciones para grafos del tamaño presentado en este trabajo, elegimos grafos para su estudio,con un número pequeño de nodos sensibles así como un pequeño rango de variación para los nodos sensibles, a fin de poder realizar un análisis exhaustivo para un valor aceptable de combinaciones generadas.

Con objeto de mostrar la metodología de la experimentación realizada, hemos desarrollado un ejemplo al que haremos referencia a fin de mostrar los valores de rendimiento que presenta, y el conjunto de operaciones que se aplican a la hora de obtener estos valores.

A continuación presentamos los indices de rendimiento que utilizaremos posteriormente para medir las prestaciones de los distintos algoritmos utilizados, aplicados al ejemplo elegido.

a.- "Desviación A": La diferencia que se produce respecto al CP/MISF, como en el caso anterior, con respecto al algoritmo de acumulación de duplicaciones para las posibles combinaciones generadas, aplicando el método expuesto en 4.1.a), para cada uno de los grafos que se van obteniendo al ir modificando los tiempos de ejecución de los nodos.

b.- "Desviación B": La diferencia con respecto al tiempo de ejecución para cada grafo de la familia que se obtiene, a partir de la asignación proporcionada por el algoritmo CP/MISF/TD para los valores extremos correspondientes a los nodos sensibles, simulando la ejecución para los Ti donde (Til < Ti < Tiu), frente al tiempo de ejecución obtenido por el algoritmo CP/MISF para cada Ti donde (Til < Ti < Tiu).

Estas diferencias definidas anteriormente, son las que denominaremos "Desviacion A" y "Desviación B" respectivamente mostradas en la Tabla 4.2, es importante notar que estas, se miden respecto al "óptimo individual", es decir, se estaría midiendo la cota superior máxima de desviación, respecto a una situación óptima utópica para un caso estático.

Llegado este punto, a partir de los indices definidos en a y en b, pretendemos comparar las prestaciones de las metodologías expuestas en 4.1.a) y en 4.1.b), a fin de decidir la más adecuada. A continuación presentamos para el ejemplo mostrado en la figura 4.1, los resultados obtenidos.

Partiendo del grafo de la figura 4.1, para la familia de grafos generada, agrupamos los distintos grafos según el valor en porcentaje de sus desviaciones porcentuales, "desviación a" (según lo definido en a) y "desviación b" (según lo expuesto en b)) mostrados en la tabla 4.1, que corresponden a la aplicación del método 4.1a) y 4.1b) respectivamente para cada combinación generada. La tabla 4.2 muestra esta información de desviaciones para todos los posibles grafos de la familia expresando igualmente los valores obtenidos en diferencias porcentuales. La primera fila de la tabla 4.2 representa el número de combinaciones cuyo valor de desviaciónes es inferior al 10%, la segunda fila, representa el número de combinaciones cuyo valor de desviación está comprendido entre el 10% y el 20%, y así sucesivamente. Finalmente, se presentan los valores promedio de desviación para el total de combinaciones generadas, 288 en el caso que nos ocupa.

RESUMEN GRAFO PRUEBA1			
INTERVALO DE DESVIACION	Numero de Combinaciones	Valor promedio de desviacion (Todos los grafos)	
		A	B
0 % . . 10%	175		
10% . . 20%	80		
20% . . 30%	33	7.68E+0	8.22E+0

Tabla 4.2

4.1.2.- Comparación de ambos métodos. Análisis de resultados

A la vista de los resultados obtenidos para el grafo de la figura 4.1, mostrados en las tablas 4.1 y 4.2, podemos extraer las siguientes conclusiones:

a.- Aproximadamente, según la tabla 4.2, los valores que se obtienen de desviación para las columnas que hacen referencia a los valores definidos en "a" y "b", 2ª y 3ª columna respectivamente de la tabla 4.1, (en el apéndice A, Tabla a1, se muestra el desarrollo completo para toda la familia de grafos generada) son similares en un 95%, lo que nos lleva a la conclusión, de que para la técnica de acumulación de duplicaciones no se justifica la cantidad de tiempo necesario para generar todaslas posibles combinaciones en función de los resultados que se obtienen de mejora en el tiempo de ejecución. Este mismo resultado de forma global se muestra en la Tabla 4.2.

b.- Según la Tabla 4.2, resumen de la experimentación realizada para la figura 4.1, el 60% de las combinaciones generadas, muestran desviaciones menores que el 10% respecto al tiempo que proporcionaría el algoritmo CP/MISF para cada grafo individual de la familia, el 27% de combinaciones están entre el 10% y el 20% de desviación y el 13% aproximadamente entre el 20% y el 30%, tanto para el índice definido en "a" como en "b"; teniendo en cuenta la familia de grafos que representan

(288 grafos), tanto el método de acumular duplicaciones, como el método de considerar los valores extremos para calcular la asignación con duplicaciones para toda la familia de grafos, proporcionan unos resutados que consideramos satisfactorios. En la tabla 4.2, los respectivos métodos propuestos en 4.1.a) y 4.1.b) podemos observar que en promedio la desviación porcentual para todas las combinaciones es de 7.6% y 8.2 respecto a los valores de tiempo proporcionados por el CP/MISF.

En base a los resultados (practicamente similares aplicando 4.1.a) y 4.1.b)) presentados anteriormente, y considerando la mayor complejidad, que presenta la técnica de ir acumulando duplicaciones para cada posible combinación de tiempos de ejecución, correspondiente a los distintos valores que pueden ir tomando los nodos sensibles en ejecución, nuestra elección claramente se decanta, por considerar como asignación final de las tareas del grafo a los procesadores, la que proporciona el algoritmo CP/MISF/TD, cuando se aplica para los dos valores, identificados ahora como extremos de los rangos que caracterizan la incertidumbre del valor correspondiente al tiempo de ejecución de aquellos nodos fuertemente sensibles a los valores de entrada.

4.2.- Experimentación realizada y resultados obtenidos

En este apartado, una vez elegida la metodología de actuación frente a los nodos del grafo fuertemente sensibles, nos proponemos obtener, los valores de rendimiento para los algoritmos de duplicación de tareas cuando ampliamos el modelo de DAG, para que ciertos nodos modelen los valores extremos del rango correspondiente a su tiempo de ejecución.

A continuación definiremos los terminos de rendimiento utilizados en este apartado, con objeto de seguir nuestra discusión.

- Ganancia; este término se utiliza para expresar la mejora absoluta en el tiempo de ejecución que proporciona el algoritmo CP/MISF/TD para los valores

extremos, en comparación con el tiempo que proporcionaría el algoritmo CP/MISF cuando se calcula la asignación para unos valores de tiempos de ejecución específicos (Ti), y se realiza la ejecución para otros valores distintos (Ti*). Sea Ta, el tiempo de ejecución que proporciona el algoritmo CP/MISF/TD, y Tb el tiempo que proporciona la asignación obtenida a partir del algoritmo CP/MISF para los tiempos de los nodos (Ti), realizando la ejecución para los tiempos (Ti*); el término ganancia lo podríamos expresar como:

Ganancia = Ta - Tb

- Desviación; en este apartado se utiliza este término para expresar la diferencia obtenida en cuanto al tiempo de ejecución del grafo, cuando se obtiene la asignación proporcionada para el algoritmo CP/MISF para unos determinados valores Ti, y se realiza la ejecución para los mismos valores de Ti (Tcp), frente al tiempo que se obtiene para la asignación proporciona por el algoritmo CP/MISF/TD para los valores extremos de los nodos realizando la ejecución para los valores de los nodos Ti (Td); el término desviación lo podríamos expresar como:

Desviación = Td - Tcp

Este valor de desviación correspondería a la diferencia respecto a la hipotética cota superior.

- Ganancia máxima, representa la diferencia máxima entre el tiempo de ejecución proporcionado por el algoritmo CP/MISF, calculando la asignación para los valores iniciales (Ti), y realizando la ejecución para otros valores de tiempos de ejecución (Ti*) distintos; frente al tiempo de ejecución que proporcionaría la asignación proporcionada por el algoritmo CP/MISF para unos determinados valores de tiempo de ejecución (Ti*), realizando la ejecución para los mismos tiempos (Ti*).

Este valor de ganáncia máxima, representa el valor máximo que utilizaremos a la hora de medir las prestaciones de los algoritmos de duplicación propuestos.

La experimentación realizada en este apartado, se ha ralizado para el conjunto de grafos cuya topología se muestra en el capítulo 3, denominados G_test_1, hasta G_test_10, donde las características más importantes de los mismos se muestran a continuación:

1.- Número aproximado de nodos por grafo = 100

2.- Conjunto de nodos fuertemente sensibles: 30% del número total de nodos

3.- Tiempos de ejecución variable en nodos sensibles entre 1 y 100

4.- El rango de los nodos de tiempo variable se representa mediante 10 valores

La tabla 4.3 muestra para cada grafo (G_test_1, ..., G_test_10), y para un conjunto de 10 posibles valores de incertidumbre por nodo, los valores promedio de

Nombre del Grafo	INTERVALO	
	Ganancia	Desviacion
G_test_1	59	32
G_test_2	0	0
G_test_3	5	10
G_test_4	24	31
G_test_5	35	5
G_test_6	31	75
G_test_7	40	26
G_test_8	47	57
G_test_9	82	39
G_test_10	9	137
Media	33	41

Tabla 4.3

ganancia y desviación (campo denominado INTERVALO, en la tabla que nos ocupa) cuando utilizamos los valores extremos como representativos del conjunto de posibles tiempos de ejecución que pueden tomar los nodos en el intervalo. En esta tabla se observa para cada grafo motivo de estudio, los valores medios absolutos de ganancia y desviación (todos los valores presentados se generan partiendo de la extensión al modelo de DAG, donde los tiempos de ejecución de los nodos fuertemente sensibles que podían tomar valores de los tiempos de ejecución dentro de un rango era representada, por el tiempo de ejecución proporcionado por el algoritmo CP/MISF/TD para los valores extremos). Los valores máximos absolutos de ganancia medios para cada uno de los grafos, se presentan en la tabla 4.4, con objeto de poder compararlos con los valores de ganancia obtenidos por nuestro algoritmo de duplicación.

La ganancia porcentual (sobre la reducción máxima del tiempo de ejecución) se muestra en la tercera columna de la Tabla 4.4. Sus elevados valores en la mayoria de los casos muestra las buenas prestaciones obtenidas por el método propuesto. Comparando la columna denominada "Ganancia máxima" de la tabla 4.4, con la columna denominada "Ganancia" de la tabla 4.3, se observa, la mejora obtenida por nuestro algoritmo de duplicación tareas, siendo en algunos casos el 89% del valor máximo considerado como cota de máximo (grafo G_test_5, sobre una posible ganáncia máxima de 39 unidades de tiempo, el algoritmo proporciona una ganáncia de 35 unidades), y en media, para la experimentación realizada se obtiene según los valores mostrados en la tabla 4.4, una mejora del 44%, valor este que consideramos importante dado el grado de incertidumbre introducido en los valores de tiempo de ejecución referentes a los nodos fuertemente sensibles.

Un estudio individualizado de los tiempos de ejecución obtenidos para cada uno de los grafos de prueba puede encontrarse en el apéndice B de este trabajo.

En la tabla 4.4 se ha mostrado para cada uno de los grafos la ganancia promedio obtenida expresada en tanto por ciento sobre la posible ganancia máxima, individual

Nombre del Nombre Grafo del Grafo	INTERVALO	
	Ganancia Maxima	% sobre G. Maxima
G_test_1	91	65%
G_test_2	0	0%
G_test_3	15	33%
G_test_4	55	44%
G_test_5	39	89%
G_test_6	106	29%
G_test_7	66	61%
G_test_8	103	46%
G_test_9	121	68%
G_test_10	146	6%
Media	74	44%

Tabla 4.4

para cada uno de los grafos. En la figura 4.2 se muestra una gráfica donde se pueden observar los valores medios de ganáncia que proporciona nuestro algorimo de duplicación de tareas (representada por barras rayadas), en relación con el valor máximo de ganancia que podría obtenerse para cada grafo motivo de estudio (representada por barras de contenido liso). De las tablas y gráfica presentadas en este apartado se deduce que el hecho de considerar la asignación que proporciona el CP/MISF/TD cuando se calcula para los valores extremos de los nodos que varían de

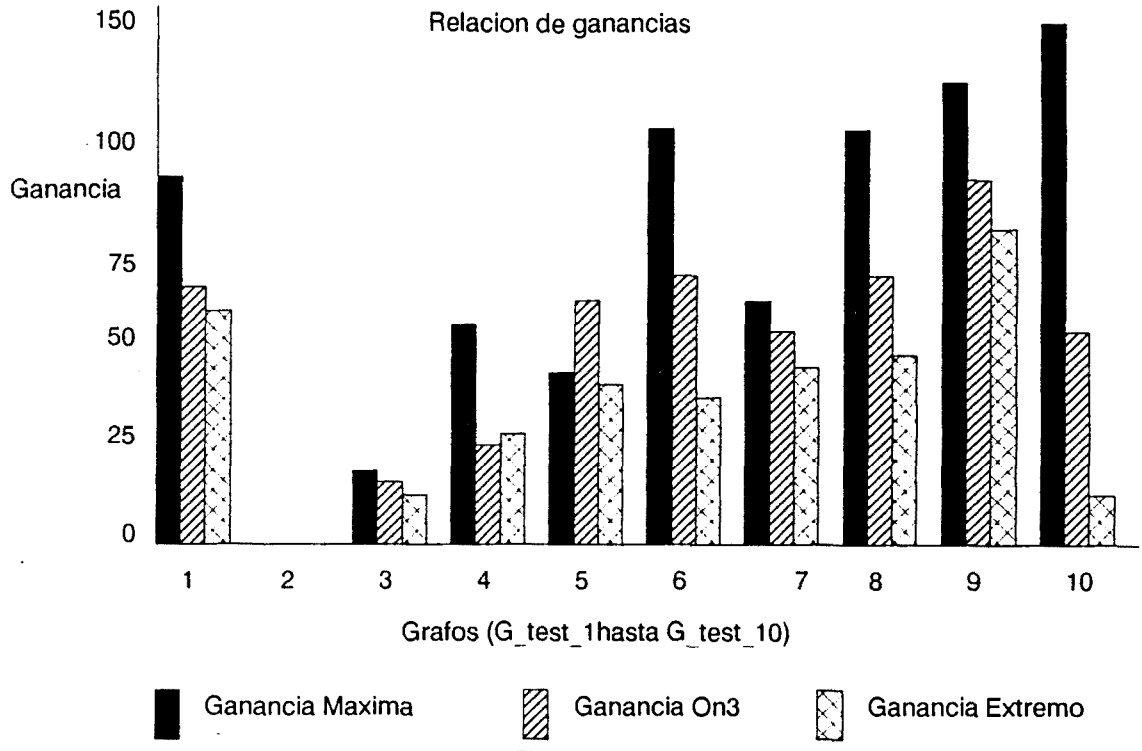


Figura 4.2

forma sensible, proporciona en promedio una ganáncia de aproximadamente el 47% sobre la ganáncia máxima, valor este que consideramos importante dado el grado de incertidumbre introducido en los grafos utilizados en la experimentación.

Conclusiones del capítulo

En este capítulo, se ha presentado una extensión de los métodos de "scheduling" mediante duplicación de tareas cuando se extiende el modelo de representación de DAG en el sentido de modelar, los tiempos de ejecución de las secciones de código fuertemente sensibles, representadas dentro de un rango de valores mediante sus valores extremos. Se ha mostrando, que mediante los resultados obtenidos en la fase experimental,los métodos de duplicación de tareas, pueden extender su rango de aplicación a dicho modelo.

Aportaciones

- Se propone la extensión del modelo de representación basado en DAG, con objeto de modelar programas reales. En una primera etapa, la ampliación ha consistido, en la inclusión de dos valores de tiempo de ejecución en los nodos, para modelar las estructuras de tipo iterativo y condicional.

- Se ha realizado un análisis de la sensibilidad de los métodos de "scheduling" estático basados en lista, frente a la variación de los tiempos de ejecución de los nodos en el modelo de DAG, lo cual implica: la detección de procesadores libres y la existencia de tareas listas para su ejecución asignadas a otros procesadores que no pueden ejecutar debido a que están ejecutando alguna de las que tenían asignadas.

- Se ha desarrollado de una política de "scheduling" que permite que algunos nodos puedan tomar tiempos de ejecución variables. Esta nueva política implica la duplicación de ciertas tareas, con una ejecución mutuamente exclusiva de las tareas duplicadas. Esta filosofía pretende identificar, la existencia de tareas listas en procesadores ocupados, conjuntamente con procesadores inactivos sin tareas listas, con objeto de eliminar los periodos de inactividad de los procesadores, mediante la duplicación de aquellas tareas listas en los procesadores inactivos.

- Creación de algoritmos para implementar la política de "scheduling" de duplicación de tareas, basados en distintos criterios de análisis sobre las trazas de Gantt, y de complejidades diferentes:

a.- Algoritmo CP/MISF/TD, basado en el camino crítico, considera una visión global del grafo del programa analizando el conjunto de huecos y posibles tareas susceptibles de llenar el hueco, para cada duplicación se calcula el tiempo de ejecución de la aplicación con la tarea duplicada con objeto de comprobar, si la duplicación mejora el tiempo final de la aplicación, el proceso continua intentando duplicar tareas una a una y realiza las comprobaciones pertinentes con objeto de

incorporar la tarea duplicada o deshechar la duplicación de la tarea simulada. El límite, superior, teórico de complejidad de este algoritmo es de $O(n^3m)$. Los resultados experimentales muestran que la complejidad real del algoritmo se aproxima a $O(K\ n^2T)$, donde n es el número de tareas, m el número de procesadores, T el número de tareas duplicadas y K es un factor constante que vale aproximadamente 3/5.

b.- Algoritmo CP/MISF/TD/RC, está basado igualmente en las ideas de camino crítico y de duplicación, la modificación sustancial con respecto al anterior consiste en el mecanismo de identificación de las duplicaciones. En este algoritmo se abandona la idea de hueco (procesador libre) para duplicar una tarea y aparece la idea de "primer instante en el cual el procesador está libre" y por tanto instante donde es susceptible de comenzar una duplicación. Si existen tareas listas en otros procesadores se asigna una, en base al criterio de elección de mayor nivel y se continua el proceso de análisis. La complejidad teórica de este algoritmo, como se ha mostrado es de $O(n^2)$ coincidiendo con la complejidad real obtenida por la experimentación realizada.

- Extensión de las políticas de "scheduling" cuando el modelo de DAG se amplia al considerar los dos valores asignados para representar el tiempo de ejecución, como límites del intervalo de variación de dicho tiempo de ejecución. De esta forma el conjunto de valores del intervalo se modela, en cuanto al problema del "scheduling", por sus valores extremos.

- Se ha procedido a la validación de los algoritmos propuestos mediante una ámplia experimentación que ha consistido en la ejecución de los mismos sobre un conjunto de grafos, sintéticos unos y correspondientes a aplicaciones reales otros.

- La política de "scheduling" desarrollada, materializada en distintos algoritmos específicos, se puede considerar, como una política híbrida entre el "scheduling" estático y dinámico. Por un lado se realiza una asignación previa, sin movimiento posterior de tareas (estático). En esta asignación, ciertas tareas, en función de su

comportamiento en tiempo de ejecución, son duplicadas en ciertos procesadores. La ejecución, mutuamente exclusiva de una de las copias, vendrá determinada por el comportamiento dinámico del programa. Externamente, por tanto, una tarea puede ser ejecutada en distintos procesadores para diferentes ejecuciones sin que la tarea haya sido migrada, por tanto se elimina el "overhead" de migración que implican los "scheduling" dinámicos.

Lineas abiertas:

Algunos de los caminos que han quedado abiertos y que consideramos de interés futuro serían los siguientes:

- Inclusión de las comunicaciones de manera explícita en el grafo de tareas, estudiando la aptabilidad de nuestros algoritmos de duplicación a la nueva situación.

- Estudiar el comportamiento de los algoritmos de planificación de tareas con duplicaciones, cuando se aplican de forma local a estructuras identificables en los grafos de tareas que representan aplicaciones.

- Estudiar la viabilidad del método propuesto de duplicación de tareas, utilizandolo como punto de partida en la realización de "schedulers" dinámicos, aprovechando las prestaciones que ofrecen los algoritmos de duplicación al permitir un elevado índice de incertidumbre en el conocimiento del tiempo de ejecución de algunos nodos.

- Estudiar el problema del ruteo de datos, cuando se aplican los algoritmos de duplicación a arquitecturas que no presentan una estructura de conexión completa y por tanto las relaciones entre los volúmenes de cómputo y comunicaciones ya no sean despreciables.