

**Centro de Investigación Científica y de  
Educación Superior de Ensenada**

División de Física Aplicada

Departamento de Ciencias de la Computación

**Simulación de una Estrategia Adaptativa  
para la Calendarización Dinámica.**

Tesis

Que para cubrir parcialmente los requisitos necesarios para  
obtener el grado de MAESTRO EN CIENCIAS presenta:

**Yolanda Marcela Avila Velasco**

Ensenada, B.C., México, octubre del 2003



**Centro de Investigación Científica y de  
Educación Superior de Ensenada**

División de Física Aplicada

Departamento de Ciencias de la Computación

**Simulación de una Estrategia Adaptativa  
para la Calendarización Dinámica.**

Tesis

Que para cubrir parcialmente los requisitos necesarios para  
obtener el grado de MAESTRO EN CIENCIAS presenta:

**Yolanda Marcela Avila Velasco**

Ensenada, B.C., México, octubre del 2003

**RESUMEN** de la tesis de **Yolanda Marcela Avila Velasco**, presentada como requisito parcial para la obtención del grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN. Ensenada, Baja California. Octubre del 2003.

## **SIMULACIÓN DE UNA ESTRATEGIA ADAPTATIVA PARA LA CALENDARIZACIÓN DINÁMICO.**

Resumen aprobado por:

---

Dr. Andrei Tchernykh  
Director de Tesis

La optimización de aplicaciones paralelas es difícil de lograr por las técnicas de optimización clásicas debido a su diversidad y de la gran variedad del equipo paralelo de soporte. Los esquemas adaptativos que son capaces de cambiar dinámicamente el algoritmo de calendarización (asignación de tareas) con el objetivo de optimizar el comportamiento del sistema global son la mejor alternativa para resolver éste problema. En esta tesis caracterizamos varios tipos de problemas que ocurren en la implementación de aplicaciones paralelas y se dá un marco para su calendarización dinámica de forma eficiente. Se presenta una nueva familia de estrategias llamada Esquema  $(a,b,c)$  Adaptativa ( $(a,b,c)$  Scheme por su nombre en inglés) que es utilizada como base de un calendarizador no determinístico que es capaz de cambiar de forma dinámica el algoritmo de calendarización y de modificar el número de procesadores asignados a las tareas durante su ejecución. Se muestra el análisis del cociente de desempeño de la estrategia  $(a,b,c)$  Scheme para algunos casos específicos con ciertas restricciones.

**Palabras clave:** Calendarización, Cociente de desempeño, Estrategia adaptativa, Factor de ineficiencia, Tarea maleable.

**ABSTRACT** of the thesis presented by **Yolanda Marcela Avila Velasco** as a partial requirement to obtain the MASTER OF SCIENCE degree in COMPUTER SCIENCE. Ensenada, Baja California, Mexico. October 2003.

## **SIMULATION OF AN ADAPTIVE STRATEGY FOR ON-LINE SCHEDULING.**

Abstract approved by:

---

Dr. Andrei Tchernykh  
Director de Tesis

The optimization of parallel applications is difficult to achieve by classical optimization techniques because of their diversity and the big variety of actual parallel and distributed supports. Adaptive schemes that are capable of dynamically changing the scheduling algorithm (allocation of jobs) during the execution in order to optimize the global system behavior are the best alternative for solving this problem. In this thesis, we characterize various types of problems occurring in the implementation of parallel applications and provide a framework for their efficient on-line scheduling. We present a new family of strategies, called adaptive (a,b,c)-Scheme that is used as the basis of the non-clairvoyant scheduler such that it is capable of on-line changing the scheduling algorithm, and of modifying the number of processors allocated to jobs during their execution. The analysis of the (a,b,c)-Scheme performance guarantee for some specific restricted cases and experimental results are provided.

**Keywords:** Scheduling, Performance ratio, Adaptive strategy, Inefficiency factor, Malleable task.

## **Dedicatoria**

A la persona que mas me ha dado en esta vida y que me enseñó lo mejor que tengo y que no tengo como agradecerle: Mi madre.

A mi padre que me enseñó lo que es superarse en la vida sin importar lo duro que sea.  
Descanse en Paz!.

A mis hermanos Ana, Angel, Leopoldo y Victor por darme siempre fuerzas para seguir y apoyarme en todo.

A Dios por lograrlo.

## Agradecimientos

Quisiera iniciar agradeciendo al Dr. José D. Frez Cardenas por su valioso apoyo al inicio de este trabajo de investigación.

Así como a la Dra. Ana Isabel Martínez por todo su apoyo durante mi estancia en CICESE.

A mi comité de tesis por sus valiosas aportaciones para la realización de este documento.

Al Sr. Gilberto Ibarra por su invaluable ayuda.

A mi gran amiga Verónica Ruíz por su amistad y ayuda incondicional...Achhhhh!!! .

A todo mis compañeros y amigos de generación por su gran apoyo en los momentos de locura y sobretodo por su amistad: Brenda Flores, Elitania Jiménez, Francisco García, Jorge Ibarra, Rafa Villa, Sonia Sosa y Mary Cuevas (descanse en paz!).

A mis "compas" de otras generaciones por tenerme tanta paciencia, por su apoyo y amistad!!!: Blanca Román, Edelmira Rodríguez, José Barraza, Manuel Alba, Octavio García, Rafa Llamas, Ricardo Acosta, Romualdo Zayas, Urania Ceseña.

A mis compañeros de laboratorio: Antonio Rodríguez, Cristóbal Salas, Reyes Juárez.

A las secretarias de Ciencias de la Computación por ser siempre tan amables conmigo: Carolina Amador y Lydia Salazar.

Le agradezco también a Steve y Betty Stribling por permitirme hacer tantas mezclas con mi horario de trabajo.

A Gilberto Ibarra por darme ánimos a seguir, por su amor, apoyo y estar siempre a mi lado en los momentos difíciles. Te amo!!!.

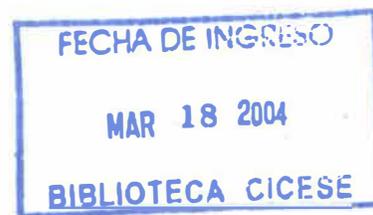
Finalmente a **CONACYT** por darme la oportunidad de continuar con mis estudios y cumplir con un sueño.

# Indice

	Pág.
I	Introducción..... 1
I.1	Notación..... 8
I.2	Objetivo de la tesis..... 9
I.3	Justificación..... 11
I.4	Metodología..... 13
I.5	Trabajo Previo..... 14
I.6	Organización del documento. .... 16
II	Calendarización..... 18
II.1	Calendarización de Lista..... 19
III	Modelo de Tareas Maleables y técnica de distribución Gang..... 23
III.1	Técnica de distribución Gang..... 23
III.2	Modelo de tareas maleables..... 24
III.3	Factor de ineficiencia $\mu$ ..... 26
IV	Scheme..... 30
V	$(a, b, c)$ -Scheme..... 35
V.1	Parámetro $(a)$ ..... 35
V.2	Parámetro $(b)$ ..... 36
V.3	Parámetro $(c)$ ..... 38
V.4	Algoritmo..... 40
VI	Análisis del problema..... 46
VI.1	Caso $(a, l, \theta)$ -Scheme..... 48
VI.2	Caso $(a, m, \theta)$ -Scheme..... 51
VII	Sistema experimental..... 60
VII.1	Generación del conjunto de tareas..... 61
VII.2	Descripción de la implementación del calendarizador..... 66
VII.2.1	Lectura de datos de entrada..... 69
VII.2.2	Procesamiento del conjunto de tareas..... 69
VII.2.2.1	Procedimiento <i>Modeling</i> ..... 72
VII.2.2.2	Procedimiento <i>MallCondition</i> ..... 73
VII.2.2.3	Procedimiento <i>MallAsing</i> ..... 74
VII.3	Almacenamiento de los resultados..... 78
VIII	Resultados Experimentales..... 80
VIII.1	Tiempo de procesamiento de una tarea $(p)$ ..... 80
VIII.2	Funciones utilizadas para calcular el factor de ineficiencia..... 83
VIII.3	Descripción de las estrategias de calendarización utilizadas..... 85
VIII.4	Análisis de los resultados experimentales..... 87
IX	Conclusiones y trabajo futuro..... 97
IX.1	Aportaciones..... 99
IX.2	Trabajo Futuro..... 100

## Índice (continuación)

	Pág.
X Referencias y Literatura citada.....	102
Apéndice A Resultados experimentales con conjuntos de 500 tareas y tiempos de procesamiento no unitarios.....	105
Apéndice B Herramienta Scheduling.....	120
Apéndice C Computadora Paralela cicese2000.....	128



## Lista de figuras

Pág.

Figura 1.	Gráfica de Gantt. En la figura 1a el criterio para asignar las tareas es de acuerdo a su llegada al sistema. En la figura 1b el criterio que se aplica es de acuerdo al menor tiempo de procesamiento de cada tarea. En la figura 1c se toma como criterio el mayor tiempo de procesamiento de la tarea.....	22
Figura 2.	Factor de ineficiencia.....	28
Figura 3.	Estrategia <i>Scheme</i> . La figura de la izquierda representa cómo se ejecutan las tareas si se aplica el algoritmo de Graham únicamente. La figura de la derecha muestra como se lleva a cabo la estrategia <i>Scheme</i> . Al inicio, cada tarea la ejecuta un solo procesador (etapa 1), a partir del tiempo $t$ , a cada una de las tareas que aún no terminan su ejecución se le asignan todos los procesadores (etapa 2). La etapa 2 inicia cuando se tiene un procesador libre.....	32
Figura 4.	Estrategia $(a,b,c)$ - <i>Scheme</i> . La figura de la izquierda ilustra la calendarización de un conjunto de tareas. Cada tarea requiere un solo procesador y se aplica el algoritmo de Graham. En la figura de la derecha se muestra el calendario generado por la estrategia $(a,b,c)$ - <i>Scheme</i> para el mismo conjunto de tareas. Las tareas se interrumpen en el momento que existe un procesador ocioso, como se observa en el tiempo $t$ . Las áreas no sombreadas representan una tarea que se interrumpió en $t$ y que a partir de $t'$ se paraleliza en los $m$ procesadores de acuerdo a la estrategia Gang. ....	42
Figura 5.	Estrategia $(a,b,c)$ - <i>Scheme</i> , donde $a = 4$ , $b = m$ y $c = 0$ . En la figura de la izquierda cada tarea se asigna a un procesador y cuando éste termina de ejecutarla toma una nueva tarea de acuerdo al algoritmo de Graham. En la figura de la derecha las tareas se ejecutan de acuerdo a la estrategia $(a,b,c)$ - <i>Scheme</i> . En la etapa 1a, cada tarea la ejecuta un procesador. En ambas figuras durante esta etapa todos los procesadores están ocupados. En la etapa 1b, existen menos de $a$ procesadores ociosos, por lo tanto no se interrumpe la ejecución de las tareas. La duración de esta etapa se representa por $\delta$ . En la etapa 2, se observa que a partir del tiempo $t'$ las tareas que aún no concluyen su ejecución se interrumpen y se asigna cada una de éstas a los $m$ procesadores, como lo muestra la figura de la derecha. Esta etapa inicia en el momento que se tienen los $a$ procesadores ociosos.....	45
Figura 6.	Ejemplo de la calendarización $(a,1,0)$ - <i>Scheme</i> . ....	49
Figura 7.	$\rho_{(a,m,0)\text{-Scheme}}$ al variar el número de procesadores, $a=30$ , $\mu_m=0.0038m+0.9975$ . ....	58
Figura 8.	$\rho_{(a,m,0)\text{-Scheme}}$ al variar el valor de $a$ , $m=160$ ; $\mu=0.0038m+0.9975$ . ....	59

## Lista de figuras (continuación)

Pág.

Figura 9.	Representación en diagrama de flujo del mecanismo para la generación del conjunto de tareas. ....	64
Figura 10.	Información almacenada en el archivo <i>aaa.pas</i> . ....	65
Figura 11.	Diagrama de flujo de la etapa de implementación del calendarizador. ....	68
Figura 12.	Diagrama de flujo general de la etapa de procesamiento del conjunto de tareas. ....	71
Figura 13.	Diagrama de flujo del procedimiento <i>Modeling</i> . ....	72
Figura 14.	Representación en diagrama de flujo de la función <i>MallCondition</i> . ....	74
Figura 15.	Representación gráfica de <i>MallAssign</i> . ....	75
Figura 16.	Asignación de tareas de acuerdo a la estrategia <i>Gang</i> . ....	75
Figura 17.	Asignación de tareas de acuerdo a la estrategia $(a,b,c)$ -Scheme. ....	77
Figura 18.	Gráfica de FDP de la distribución Gaussiana con $\mu_x = 70$ y $\sigma_x = 28$ . ...	82
Figura 19.	Comportamiento del factor de ineficiencia. ....	85
Figura 20.	Cociente de desempeño promedio de $(a,b,c)$ -Scheme con 100 experimentos, donde $n=500$ , $m=160$ . ....	89
Figura 21.	Cociente de desempeño promedio de $(a,b,c)$ -Scheme (100 experimentos, $n=500$ , $a=30$ ). ....	90
Figura 22.	Cociente de desempeño promedio de la estrategia de distribución <i>Gang</i> (100 experimentos, $n=500$ , $m$ varía de 31 a 160). $\mu$ es una función lineal, cóncava y convexa. ....	91
Figura 23.	Cociente de desempeño promedio de la estrategia $(a,maleable,1)$ -Scheme (100 experimentos, $n=500$ , $m=160$ ). ....	92
Figura 24.	Cociente de desempeño promedio de la estrategia $(a,maleable,1)$ -Scheme (100 experimentos, $n=500$ , $a=30$ ). ....	92
Figura 25.	Cociente de desempeño promedio de las estrategias $(a,maleable,1)$ -Scheme y $(a,m,0)$ -Scheme (100 experimentos, $n=500$ , $m=160$ ). $\mu$ es una función lineal. ....	93
Figura 26.	Cociente de desempeño promedio de $(30,maleable,1)$ -Scheme y $(30,m,0)$ -Scheme (100 experimentos, $n=500$ , $m$ varía de 31 a 160). $\mu$ es una función lineal. ....	94
Figura 27.	Desempeño promedio de 100 experimentos de $(a,b,c)$ -Scheme para $a=1$ y diferentes valores de $b$ . $m$ varía de 2 a 160 procesadores, aplicando un factor de ineficiencia lineal. ....	105
Figura 28.	Desempeño promedio de 100 experimentos de $(a,b,c)$ -Scheme para $a=1$ y diferentes valores de $b$ . $m$ varía de 2 a 160 procesadores, aplicando un factor de ineficiencia cóncavo. ....	105
Figura 29.	Desempeño promedio de 100 experimentos de $(a,b,c)$ -Scheme para $a=1$ y diferentes valores de $b$ . $m$ varía de 2 a 160 procesadores, aplicando un factor de ineficiencia convexo. ....	106

## Lista de figuras (continuación)

	Pág.
Figura 30. Desempeño promedio de 100 experimentos de LPT y $(a,b,c)$ -Scheme para $a = 1$ y diferentes valores de $b$ . $m$ varía de 2 a 160 procesadores, aplicando un factor de ineficiencia convexo.....	106
Figura 31. Desempeño promedio de 100 experimentos de $(a,maleable,l)$ -Scheme para $a=1$ . $m$ varía de 2 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal. ....	107
Figura 32. Desempeño promedio de 100 experimentos de $(a,m,0)$ -Scheme para $a=1$ . $m$ varía de 2 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal. ....	107
Figura 33. Desempeño promedio de 100 experimentos de $(a,b,c)$ -Scheme para $a=30$ y diferentes valores de $b$ . $m$ varía de 31 a 160 procesadores, aplicando un factor de ineficiencia lineal.....	108
Figura 34. Desempeño promedio de 100 experimentos de $(a,b,c)$ -Scheme para $a=30$ y diferentes valores de $b$ . $m$ varía de 31 a 160 procesadores, aplicando un factor de ineficiencia cóncavo. ....	108
Figura 35. Desempeño promedio de 100 experimentos de $(a,b,c)$ -Scheme para $a=30$ y diferentes valores de $b$ , $m$ varía de 31 a 160 procesadores, aplicando un factor de ineficiencia convexo. ....	109
Figura 36. Desempeño promedio de 100 experimentos de LPT y $(a,b,c)$ -Scheme para $a=30$ y diferentes valores de $b$ . $m$ varía de 31 a 160 procesadores, aplicando un factor de ineficiencia convexo.....	109
Figura 37. Desempeño promedio de 100 experimentos de $(a,maleable,l)$ -Scheme para $a=30$ . $m$ varía de 31 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal.....	110
Figura 38. Desempeño promedio de 100 experimentos de $(a,m,0)$ -Scheme para $a=30$ . $m$ varía de 31 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal. ....	110
Figura 39. Desempeño promedio de 100 experimentos de $(a,b,c)$ -Scheme para $a=80$ y diferentes valores de $b$ . $m$ varía de 81 a 160 procesadores, aplicando un factor de ineficiencia lineal. ....	111
Figura 40. Desempeño promedio de 100 experimentos de $(a,b,c)$ -Scheme para $a=80$ y diferentes valores de $b$ . $m$ varía de 81 a 160 procesadores, aplicando un factor de ineficiencia cóncavo. ....	111
Figura 41. Desempeño promedio de 100 experimentos de $(a,b,c)$ -Scheme para $a=80$ y diferentes valores de $b$ . $m$ varía de 81 a 160 procesadores, aplicando un factor de ineficiencia convexo. ....	112
Figura 42. Desempeño promedio de 100 experimentos de LPT y $(a,b,c)$ -Scheme para $a=80$ y diferentes valores de $b$ . $m$ varía de 81 a 160 procesadores, aplicando un factor de ineficiencia convexo.....	112

## Lista de figuras (continuación)

	Pág.
Figura 43. Desempeño promedio de 100 experimentos de $(a, maleable, l)$ -Scheme para $a=80$ . $m$ varía de 81 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal.....	113
Figura 44. Desempeño promedio de 100 experimentos de $(a, m, 0)$ -Scheme para $a=80$ . $m$ varía de 81 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal.....	113
Figura 45. Desempeño promedio de 100 experimentos de $(a, b, c)$ -Scheme para $a=120$ y diferentes valores de $b$ . $m$ varía de 121 a 160 procesadores, aplicando un factor de ineficiencia lineal.....	114
Figura 46. Desempeño promedio de 100 experimentos de $(a, b, c)$ -Scheme para $a=120$ y diferentes valores de $b$ . $m$ varía de 121 a 160 procesadores, aplicando un factor de ineficiencia cóncavo.....	114
Figura 47. Desempeño promedio de 100 experimentos de $(a, b, c)$ -Scheme para $a=120$ y diferentes valores de $b$ . $m$ varía de 121 a 160 procesadores, aplicando un factor de ineficiencia convexo.....	115
Figura 48. Desempeño promedio de 100 experimentos de LPT y $(a, b, c)$ -Scheme para $a=120$ y diferentes valores de $b$ . $m$ varía de 121 a 160 procesadores, aplicando un factor de ineficiencia convexo.....	115
Figura 49. Desempeño promedio de 100 experimentos de $(a, maleable, l)$ -Scheme para $a=120$ . $m$ varía de 121 a 160 procesadores, aplicando los factores de ineficiencia lineal, cóncavo y convexo.....	116
Figura 50. Desempeño promedio de 100 experimentos de $(a, m, 0)$ -Scheme para $a=120$ . $m$ varía de 121 a 160 procesadores, aplicando los factores de ineficiencia lineal, cóncavo y convexo.....	116
Figura 51. Desempeño promedio de 100 experimentos de $(a, b, c)$ -Scheme donde $a$ varía de 1 a 159 con diferentes valores de $b$ . $m=160$ , aplicando un factor de ineficiencia lineal.....	117
Figura 52. Desempeño promedio de 100 experimentos de $(a, b, c)$ -Scheme donde $a$ varía de 1 a 159 con diferentes valores de $b$ . $m=160$ , aplicando un factor de ineficiencia cóncavo.....	117
Figura 53. Desempeño promedio de 100 experimentos de $(a, b, c)$ -Scheme donde $a$ varía de 1 a 159 con diferentes valores de $b$ . $m=160$ , aplicando un factor de ineficiencia convexo.....	118
Figura 54. Desempeño promedio de 100 experimentos de LPT y $(a, b, c)$ -Scheme donde $a$ varía de 1 a 159 con diferentes valores de $b$ . $m=160$ , aplicando un factor de ineficiencia convexo.....	118
Figura 55. Desempeño promedio de 100 experimentos de $(a, maleable, l)$ -Scheme, donde $a$ varía de 1 a 159 con diferentes valores de $b$ . $m=160$ , aplicando los factor de ineficiencia cóncavo, convexo y lineal.....	119

## Lista de figuras (continuación)

	Pág.
Figura 56. Desempeño promedio de 100 experimentos de $(a,m,0)$ -Scheme donde $a$ varía de 1 a 159. $m=160$ , aplicando los factor de ineficiencia cóncavo, convexo y lineal. ....	119
Figura 57. Pantalla principal de la herramienta <i>Scheduling</i> . ....	124
Figura 58. Pantalla con resultados de la calendarización. ....	125
Figura 59. Activación de ventanas para almacenar datos o salir del sistema. ....	126
Figura 60. Ventana para almacenar los datos del conjunto de tareas. ....	126
Figura 61. Ventana para terminar la ejecución de la herramienta. ....	127
Figura 62. Uso diario de CPU de <i>cicese2000</i> de abril a septiembre del 2002. ....	132
Figura 63. Uso mensual de CPU de <i>cicese2000</i> de abril a septiembre del 2002. ....	133

## Lista de tablas

	Pág.
Tabla I. Valores del parámetro $a$ en la estrategia $(a, b, c)$ -Scheme. ....	36
Tabla II. Valores del parámetro $b$ . ....	38
Tabla III. Cociente de desempeño de $(a, b, 0)$ -Scheme. ....	57
Tabla IV. Clasificación de $(a, b, c)$ -Scheme de acuerdo al valor de $(a, b, c)$ . ....	122

# I Introducción

A pesar del gran avance tecnológico observado en los últimos años en el desarrollo de las computadoras, aún nos enfrentamos a ciertas limitaciones. Nos referimos al hecho de que a pesar de contar con computadoras muy poderosas y rápidas, éstas no son suficientes para satisfacer las necesidades de cómputo de las grandes aplicaciones, ya que éstas requieren más velocidad de cómputo de la que actualmente se puede ofrecer.

El procesamiento en paralelo de dichas aplicaciones representa una forma de resolver este problema, ya que se puede aumentar la velocidad de procesamiento utilizando una supercomputadora o bien utilizando un grupo interconectado de computadoras independientes. A este agrupamiento se le conoce como *cluster*, por su nombre en inglés. En un cluster cada computadora cuenta con su propia memoria principal y unidades de procesamiento, las cuales aparentan ser un solo sistema y comparten recursos mediante una capa de software.

Seleccionar el orden a seguir para la ejecución de las tareas que conforman una aplicación, así como su asignación a los procesadores, nos lleva al problema de calendarización.

Calendarizar consiste en asignar procesadores a las tareas, asignándoles además un tiempo, a fin de minimizar el tiempo total de ejecución de todas las tareas, minimizar los retrasos por comunicación o maximizar la utilización de los recursos. La importancia de la calendarización radica en lograr el balance entre minimizar el tiempo de ejecución de las tareas y el adecuado uso de los recursos del sistema. Al calendarizar las tareas de una aplicación se busca encontrar el calendario más cercano al óptimo de acuerdo a un criterio dado.

Los esquemas de calendarización propuestos hasta este momento no están completamente adecuados a las características de las computadoras paralelas reales.

El problema de calendarización de tareas es uno de los problemas más difíciles tanto para sistemas distribuidos como para sistemas paralelos. Este es un problema NP-completo en su forma general (El Rewini *et al.*, 1996). Se han obtenido soluciones óptimas en casos restringidos donde sólo se toma en cuenta ciertas consideraciones de las aplicaciones reales. Para los propósitos de la calendarización, las computadoras paralelas son difíciles de definir y describir formalmente. Muchos investigadores trabajan en la búsqueda de un modelo que permita abstraer las características necesarias para llevar a cabo un estudio teórico de aplicaciones reales.

Se han desarrollado distintas estrategias de calendarización como un intento por resolver el problema en su forma general, pero estas estrategias sólo llevan a soluciones cercanas a la óptima.

En un sistema paralelo, para llevar a cabo la calendarización, se deben considerar factores como: el tipo de algoritmo, paralelismo intrínseco del algoritmo, tamaño del problema, el número de procesadores, tipo de topología, costos relacionados con la distribución de las tareas y administración del paralelismo, costos por coordinación y sincronización de los procesadores, etcétera.

La calendarización es un problema complejo en una computadora paralela debido a que se deben considerar muchos factores. Básicamente nos referimos a tres diferentes factores:

Los factores de aplicación. Éstos los determina el algoritmo aplicado, la calidad de su implementación, el tamaño del problema, el paralelismo intrínseco, etc.

Los factores del procesador. Que incluyen parámetros como el número de procesadores, la jerarquía, y las características de topología, como diámetro y ancho de bisección de la red de interconexión, ya que éstos tienen impacto en los tiempos de comunicación.

Los factores del sistema operativo. Los cuales implican un sobre costo debido a la creación y administración de tareas paralelas, a los recursos de sistema básicos, a la distribución de datos, a la coordinación de procesos paralelos, a los costos por interrupciones de las tareas y otros.

Por otra parte, podemos observar que como consecuencia de los constantes avances en el área de computación, hoy en día se tiene un mayor acceso y disponibilidad de sistemas paralelos, lo que lleva a la necesidad de tener un mejor manejo de estos sistemas. Como sabemos, los sistemas paralelos están enfocados a mejorar el desempeño de las aplicaciones que realizan un gran número de operaciones o de tareas durante su procesamiento, por lo que es necesario distribuir estas tareas entre los procesadores disponibles aplicando mecanismos para mejorar su tiempo de procesamiento y lograr una mejor distribución de la carga entre los procesadores. Además, debemos de tomar en cuenta que al aumentar el número de procesadores en el sistema, se incrementa el costo por sincronización y el sobre costo por comunicación entre los procesadores.

Los recursos de un sistema paralelo se pueden administrar a través de la calendarización, ya que ésta permite asignar los recursos en tiempo, para llevar a cabo las tareas que forman parte de la aplicación. Es decir, la calendarización permite diseñar un calendario que minimice algunos objetivos como: el tiempo total de procesamiento de todas las tareas (conocido como *makespan*) o el tiempo promedio de procesamiento de las tareas, donde las

*tareas* son las acciones ejecutadas por los procesadores. De acuerdo a la información que se conoce antes de iniciar la ejecución de las tareas, la calendarización puede ser:

- Estática o determinística. Se conoce de antemano toda la información sobre las tareas para llevar a cabo la calendarización. La asignación de las tareas a los procesadores es fija.
- No determinística. Se desconoce cierta información sobre las tareas. El tiempo de ejecución de la tarea se conoce hasta que ésta finaliza su ejecución. Las tareas se asignan a los procesadores conforme arriban al sistema.

La información sobre las aplicaciones puede ayudar a mejorar la calendarización. Las aplicaciones pueden crear diferentes restricciones en las políticas de calendarización dependiendo de sus características dinámicas o estáticas. Las aplicaciones pueden ser:

- Aplicaciones regulares. Aquí los patrones de computación y comunicación son fijos, se conocen los componentes de las librerías y los patrones se generan durante la compilación. Si el tamaño del problema se conoce, el grado de paralelismo se puede estimar. Para este tipo de aplicaciones es posible adaptar los resultados de la teoría clásica de calendarización estática o determinística.
- Aplicaciones Irregulares. Los patrones de computación, comunicación, tiempo de ejecución, y su paralelismo son irregulares, ya que existe dependencia de datos y cambian dinámicamente.
- Aplicaciones imprevisibles. Los patrones de comunicación y el tiempo de ejecución son impredecibles, las tareas pueden estar disponibles en cualquier momento. Por ejemplo, las tareas que están siendo ejecutadas por un número de procesadores

pueden crear nuevas tareas en forma imprevisible durante su ejecución. Imprevisible implica que la calendarización no se conoce antes de la ejecución de las tareas. No permite el arribo de tareas en un futuro. Lo dinámico de un sistema paralelo también afecta la imprevisibilidad, ya que muchos o pocos procesadores pueden estar disponibles en un tiempo no previsible, y por lo tanto, el calendarizador de tareas debe ajustar dicha variación de procesadores o de recursos. Como calendarizador nos referimos al algoritmo encargado de construir un calendario para el problema de calendarización.

- Tareas administradas no definidas, en muchos sistemas prácticos de calendarización múltiples usuarios envían tareas dinámicamente a una computadora. Un número desconocido de tareas puede arribar, lo que hace difícil describir el comportamiento o predecir a priori su ejecución.

Si las restricciones de la aplicación no son previsibles o son indefinidas, el calendarizador dinámico puede asignar los recursos a las tareas basándose en la disponibilidad o información previsible, como son los factores de procesador o factores de sistema operativo.

A través de un *calendario* se sabe que procesador ejecuta cada tarea, así como el orden en que habrán de ejecutarse cada una de éstas. Se pueden tomar diferentes criterios para ordenar un conjunto de tareas tanto en tiempo como en espacio, lo que genera calendarios con tiempos de ejecución diferentes. En este trabajo nuestro interés se centra en el problema de calendarización que busca encontrar el calendario que muestre el menor tiempo de ejecución para un determinado conjunto de tareas.

El modelo de tareas con factor de penalidad es uno de los modelos que intentan proponer un acuerdo entre las restricciones de una aplicación. El factor de penalidad toma implícitamente en consideración algunas restricciones, cuando éstas son indefinidas, u oculta características de aplicaciones reales o parámetros de cómputo; sin embargo, ellos son muy complejos para utilizarse como restricciones formales de calendarización.

Para un estudio teórico, el factor de penalidad debe tomar implícitamente en cuenta todos los factores ya mencionados. La definición de este factor de penalidad o su valor se puede basar en el análisis empírico de la carga de trabajo, evaluación comparativa del desempeño (*benchmarking*, por su nombre en inglés), evaluación del desempeño a través del modelado o medición, o bien, basada en la aplicación de otras herramientas de instrumentación disponibles en el mercado.

Dependiendo del problema, las soluciones de su calendarización pueden ser simples o complejas; además de ser diferentes, requieren de una variedad de modelos y técnicas. Una forma de manejar problemas reales de calendarización es el utilizar un marco teórico que soporte diferentes soluciones y que se puede adaptar en forma dinámica.

La necesidad por el diseño de aplicaciones paralelas eficientes, es la que lleva al uso del modelo de tareas maleables, el cual difiere del modelo tradicional donde una tarea la ejecuta un procesador a la vez. Se considera que una tarea es *maleable*, si ésta la puede ejecutar más de un procesador en una misma unidad de tiempo, el número de procesadores asignados no es fijo y su tiempo de ejecución depende del número de procesadores asignados.

Dentro del área de cómputo paralelo el problema de calendarización, en particular el de calendarización dinámica y el de calendarización de tareas maleables, está enfocado a la reducción del tiempo de ejecución de las tareas. Este problema consiste en asignar los recursos del sistema de tal forma que las tareas, ya sean dependientes o independientes, se asignan a un conjunto de procesadores idénticos o con características distintas. Las tareas se pueden ejecutar en un número arbitrario de procesadores y su tiempo de procesamiento depende del número de procesadores asignados a las tareas. Esto se conoce como el *modelo de tareas maleables*. Dentro del modelo de tareas maleables, se dice que una tarea es *no maleable* una vez que se conoce el número de procesadores asignados a esta tarea.

Por otra parte, el número máximo de tareas que se puede ejecutar en un momento dado define el grado de paralelismo del modelo computacional.

Se han realizado un gran número de trabajos enfocados al problema de reducir el tiempo de ejecución de un conjunto de tareas. Durante mucho tiempo, el proceso de calendarización se llevó a cabo utilizando únicamente una sola estrategia de calendarización durante todo el proceso de calendarización. Recientemente Rapine *et al.*, (1998) propusieron el uso de una nueva estrategia llamada *Scheme*, cuya característica principal es el de aplicar estrategias diferentes durante el proceso de calendarización, con el objetivo de reducir el número de procesadores ociosos y mejorar la eficiencia. Llamamos a este tipo de estrategia, *estrategia adaptativa*.

Por otra parte, Tchernykh *et al.*, (2002) propusieron recientemente la estrategia adaptativa llamada  $(a,b,c)$ -*Scheme*, la cual se basa en el método propuesto por Rapine *et al.*, (1998). Básicamente esta estrategia cambia a una segunda estrategia cuando existen  $a$  procesadores ociosos. A partir del cambio de estrategia, se determina el número  $b$  de procesadores que se

asignarán a cada tarea. Por otra parte,  $c$  indica el tipo de estrategia utilizada para la selección de una tarea. Las características de la estrategia adaptativa  $(a,b,c)$ -Scheme se analizan detalladamente a lo largo de este documento, además se lleva a cabo la simulación de dicha estrategia.

## 1.1 Notación

En este trabajo se analiza un conjunto de tareas maleables independientes, con el objetivo de minimizar el calendario que muestra el tiempo total de procesamiento de las tareas, llamado *makespan*. Se estudia el siguiente problema de calendarización. Dado un conjunto  $P = \{P_1, \dots, P_m\}$  de  $m$  procesadores idénticos y un conjunto  $T = \{T_1, \dots, T_n\}$  de  $n$  tareas maleables independientes e interrumpibles cuyo tiempo de procesamiento se desconoce antes de terminar su ejecución. Llamaremos  $p_i$  al tiempo de procesamiento de la tarea  $T_i$ . Debemos mencionar que el número de procesadores necesarios para la ejecución de una tarea no se impone necesariamente al inicio de la ejecución. Una tarea se puede interrumpir y continuar su ejecución en un número diferente de procesadores. Las interrupciones no afectan el tiempo de procesamiento total ya que no se tiene considerado un pago explícito por éstas. Con  $w_m^i$  denotamos el área computacional o trabajo de la tarea  $T_i$  ejecutada en  $m$  procesadores, es decir,  $w_m^i = p_m^i \cdot m$ , donde  $p_m^i$  es el tiempo de procesamiento de la tarea  $T_i$  ejecutada en  $m$  procesadores. Por otra parte, cuando una tarea  $T_i$  se ejecuta en  $m$  procesadores existe cierta ineficiencia, debido al pago que es necesario hacer por la administración del paralelismo. A esto lo llamaremos  $\mu_m^i$  o factor de ineficiencia. Para más información consúltese la sección 3.2.

Describiremos con  $C_{opt}$  a la longitud del calendario óptimo para el problema dado. Sea  $I$  un caso del problema y  $S_{opt}(I)$  un calendario óptimo para  $I$ , donde  $C_{opt}(I)$  corresponde al tiempo total de procesamiento de las tareas (*makespan*). Dada una estrategia  $A$ , denotaremos como  $S_A(I)$  al calendario correspondiente con tiempo total de procesamiento de las tareas (*makespan*)  $C_A(I)$ . Llamaremos  $C_A$  a la longitud del calendario generado por la estrategia  $A$ . Denotaremos como  $\rho_A^*(I) = \frac{C_A(I)}{C_{opt}(I)}$  al cociente de desempeño de la estrategia  $A$  para el caso  $I$ . Podemos observar que este resultado se obtiene al dividir el calendario generado por la estrategia con el calendario óptimo. El desempeño de la estrategia  $A$  se define como  $\rho_A^* = \sup \left\{ \frac{C_A(I)}{C_{opt}(I)} \right\}$ .

Denotaremos con  $a$  al número de procesadores ociosos al momento de hacer el cambio de estrategia. Llamaremos  $b$  al número de procesadores asignados a una tarea después del cambio de estrategia.

## ***1.2 Objetivo de la tesis***

La necesidad de procesar grandes cantidades de información, así como la de distribuir los recursos eficientemente, es una motivación constante para tratar de encontrar estrategias de calendarización eficientes que mejoren los resultados conocidos a la fecha. Siguiendo esta necesidad, el presente trabajo de investigación tiene como objetivo el analizar el comportamiento de una nueva estrategia de calendarización no determinística, llamada  $(a,b,c)$ -Scheme. Esta estrategia permite aplicar diversas estrategias de calendarización

durante el procesamiento de las tareas, además de permitir que una tarea se asigne a más de un procesador a la vez durante su procesamiento.

Se analiza tanto el desempeño teórico como experimental de la estrategia al calendarizar un conjunto de tareas  $T_i$  independientes, con tiempos de procesamientos no unitarios, dentro de un sistema compuesto por  $m$  procesadores idénticos, a fin de determinar el caso promedio y el peor caso para cada estrategia estudiada.

Además, entre los objetivos podemos mencionar que se busca conocer el momento idóneo para que se lleve a cabo el cambio de estrategia, para evitar que existan muchos procesadores libres, así como el determinar cuál de las diferentes estrategias aplicadas después del cambio de estrategia muestra una mejor eficiencia. Los resultados generados por las estrategias estudiadas se comparan con resultados conocidos.

Para lograr el objetivo principal es necesario cumplir con las siguientes metas:

1. Investigar el modelo de tareas maleables e identificar las características de la calendarización no determinística.
2. Investigar el comportamiento del factor de ineficiencia.
3. Estudiar la estrategia de calendarización en la cual se basa la estrategia adaptativa  $(a, b, c)$ -*Scheme*.
4. Investigar la estrategia adaptativa que permita que una tarea se asigne a más de un procesador. Se restringió dicha investigación únicamente a dos casos propuestos, para esta estrategia.

5. Diseñar una herramienta útil que comprenda, quizá no todas las especificaciones de la estrategia adaptativa analizada, pero capaz de realizar parte de los casos restringidos de esta estrategia.
6. Implementar la herramienta y realizar un análisis experimental de la paralelización de conjuntos pequeños de tareas utilizando la estrategia adaptativa.

Por otra parte, se tiene contemplado diseñar una herramienta que sirva como software de experimentación para validar los resultados de las estrategias.

En resumen, podemos decir que nuestro objetivo se centra en el procesamiento de tareas independientes que se pueden asignar a más de un procesador en un mismo instante de tiempo, con tiempos de procesamiento no unitarios. Estas tareas se ejecutan en procesadores con características idénticas y se puede interrumpir después de iniciar su ejecución, tomando como criterio de optimización la minimización del tiempo total de procesamiento de las tareas.

### ***1.3 Justificación***

En el presente trabajo se busca analizar el comportamiento de la estrategia adaptativa de calendarización conocida como  $(a,b,c)$ -Scheme, que permite que tareas independientes, con tiempos de ejecución no determinísticos, se calendaricen utilizando más de una estrategia. Se busca obtener el resultado que se aproxime más al óptimo. Específicamente se busca conocer el momento idóneo para realizar el cambio de estrategia, de manera que se obtenga un equilibrio entre la utilización de los recursos del sistema, el número de procesadores ociosos y el tiempo de ejecución de las tareas; lo que representa una gran ventaja para todos aquellos usuarios que requieren paralelizar grandes aplicaciones en equipos de cómputo

paralelo con características homogéneas, reduciendo los costos por paralelización, utilizando eficientemente los recursos y logrando que la ejecución de la aplicación sea en el menor tiempo posible, en aplicaciones para las cuales se desconocen ciertos aspectos antes de su ejecución.

Actualmente esta estrategia propone diversas formas de llevar a cabo la calendarización, pero nuestro trabajo está enfocado en dos casos en particular. La importancia del primer caso radica en que los procesadores están siempre ocupados, evitándose así los tiempos ociosos, pero se tiene que pagar un costo elevado por la paralelización de las tareas. Mientras que en el segundo caso es menor el pago que se hace por paralelizar una tarea, ya que ésta se paraleliza conforme se tienen procesadores ociosos por lo que no es necesario asignar todos los procesadores a una sola tarea. Además de lo anterior, se busca realizar la paralelización en el menor tiempo posible.

Es importante mencionar que hasta el momento no se cuenta con trabajos previos que demuestren en forma experimental el comportamiento de esta estrategia, o si efectivamente logra mejorar los resultados conocidos hasta el momento; así como tampoco se tienen resultados por implementaciones hechas en aplicaciones reales.

Este trabajo lleva a cabo la simulación de la estrategia a fin de comparar los resultados teóricos con los generados por la simulación. Estos resultados permiten observar bajo qué circunstancias se logra obtener un buen desempeño con la estrategia adaptativa. Nos permite observar si alguno de los casos analizados ofrece alguna ventaja sobre el otro, lo que representa un avance para la calendarización de las tareas utilizando esta estrategia.

Por lo tanto, la justificación de nuestro trabajo se basa en aportar los primeros resultados de la estrategia adaptativa, mediante los cuales se puede observar como varía el

comportamiento de la estrategia de acuerdo al momento y al tipo de estrategia que se aplica durante la calendarización. Este es un primer avance de utilidad para aquellos que en un futuro lleven a cabo la implementación de la estrategia en aplicaciones reales.

#### ***I.4 Metodología***

Para llevar a cabo la investigación que se propuso en este documento, fue necesario cumplir con los siguientes aspectos:

1. Estudiar los antecedentes teóricos de las estrategias de calendarización, identificándose sus principales características, así como de los temas relevantes relacionados con el tema de estudio.
2. Estudiar el modelo de calendarización de lista.
3. Estudiar a profundidad el modelo de tareas maleables y el factor de ineficiencia.
4. Estudiar la estrategia de calendarización llamada *Scheme*.
5. Realizar un análisis de la estrategia de calendarización  $(a,b,c)$ -*Scheme* y de los resultados teóricos logrados hasta el momento.
6. Diseñar e implementar una herramienta capaz de simular la estrategia de calendarización estudiada.
7. Analizar los resultados logrados durante la etapa de experimentación, con el objetivo de identificar el mejor cociente de desempeño promedio mostrado por esta nueva estrategia de calendarización.

## 1.5 Trabajo Previo

La mayoría de los trabajos realizados para tareas maleables se basan en el método de dos pasos propuesto por Ludwing (1995). La idea básica de este método consiste en seleccionar el número de procesadores que se debe asignar a cada tarea, y una vez establecido este número, se procede a la solución del problema de calendarización no maleable (Mounié *et al.*, 1999). En otras palabras, se selecciona el calendario que muestre el mejor resultado como consecuencia de aplicar un algoritmo para tareas no maleables en combinación con cada una de las asignaciones generadas (Ludwing, 1995). Se dice que una tarea *maleable* se llama tarea *no maleable* después de asignarle a esa tarea cierto número de procesadores.

De acuerdo a este método, Turek *et al.*, (1992) mostraron que un algoritmo con factor de aproximación  $p$ , que se ejecuta en un tiempo de  $O(n, m)$  para el problema de calendarización no maleable, se puede adaptar a un algoritmo con factor de aproximación  $p$ , ejecutando en un tiempo de  $O(n, m \cdot O(n, m))$  el problema de calendarización maleable.

Otro logro importante relacionado al problema de calendarización de tareas maleables es el propuesto por Ludwing (1995), quien obtuvo un algoritmo con factor de aproximación  $p$  que se ejecuta en un tiempo de  $O(mn)$ , tomando como criterio de optimización la minimización del tiempo total de procesamiento de las tareas maleables, conocido como *makespan*. Para llevar a cabo la calendarización de las tareas maleables, Ludwing presentó un método general de dos pasos que separa el problema en la selección de la asignación y la calendarización de las tareas no maleables (Ludwing, 1995). Además, Ludwing mostró un algoritmo que selecciona la asignación óptima de procesadores para un conjunto de tareas  $T$  de peso desconocido; este algoritmo se ejecuta en un tiempo de  $O(n^3 + mn)$ . También

presentó un algoritmo para la selección de la asignación de procesadores con factor de aproximación igual a dos, que se ejecuta en un tiempo de  $O(n^2 + mn)$  con tareas con peso.

Jansen y Porklab (1999) en su trabajo propusieron un esquema de aproximación completo basado en la formulación de programación lineal entera para la calendarización de un conjunto de  $n$  tareas maleables independientes (Mounié *et al.*, 1999).

El problema de tareas no maleables se formula como un programa lineal después de proponer para el problema de tarea maleable un esquema de aproximación de tiempo lineal. Con base en lo anterior, se probó que para cualquier valor fijo de  $m$ , hay un esquema de aproximación con tiempo polinomial para el problema de tareas no maleables que ejecute en un tiempo de  $O(n)$  un calendario no interrumpible (Jansen y Porklab, 1999). La longitud de este calendario es a lo más  $(1 + \varepsilon)$  veces la longitud del calendario óptimo.  $\varepsilon$  es un número entero.

Mounié *et al.*, (1999) propusieron un algoritmo para calendarizar un conjunto de  $n$  tareas maleables independientes que muestra un cociente de desempeño de  $\sqrt{3}$  al buscar la minimización del tiempo total de procesamiento de las tareas, o *makespan*, en un sistema multiprocesador compuesto por  $m$  procesadores idénticos. Este resultado mejora los resultados ya conocidos para la solución de este problema.

El método propuesto por Mounié *et al.*, (1999) muestra cómo el problema de selección de la asignación se formula como un problema de optimización de tipo *knapsack*, utilizando ya sea un algoritmo de aproximación exacta o bien un algoritmo de aproximación basado en el problema *knapsack*. En cuanto al problema de calendarización, éste lo resuelve un algoritmo de aproximación dual que lleva a un calendario sencillo de dos niveles

consecutivos. Este tipo de calendario se conoce como calendario *k - Shelf* (Mounié *et al.*, 1999).

Ahora bien, respecto al proceso de calendarización en el cual se desconoce cierta información sobre las tareas, existe una cantidad de trabajos que muestran el tipo de estrategia aplicada y el resultado obtenido durante todo el proceso. La gran mayoría de estos trabajos únicamente permiten aplicar un solo tipo de estrategia durante todo el proceso de calendarización, pero pensando en la ventaja que puede representar el aplicar dos estrategias diferentes durante el proceso de calendarización, Rapine, Scherson y Trystram propusieron una nueva estrategia llamada *Scheme* (Rapine *et al.*, 1998). Esta estrategia combina la ventaja de aplicar la estrategia *Graham* mientras la eficiencia del proceso sea igual a uno, pero una vez que existe un procesador ocioso todos los recursos del sistema se asignan a cada tarea que está pendiente de ejecutarse de acuerdo a la calendarización *Gang*. Como observamos, las dos estrategias utilizadas por *Scheme* son *Graham* y *Gang*. Dado que la estrategia que se analiza en este trabajo de tesis se basa en *Scheme*, se ha dedicado toda una sección de este documento a dar una explicación detallada de la estrategia y de los resultados obtenidos con ésta.

## ***1.6 Organización del documento***

El documento se ha distribuido de la siguiente manera. El capítulo 2 trata sobre la calendarización y su clasificación. En el capítulo 3 se explica el modelo de tareas maleables, la técnica de distribución *Gang* y el comportamiento del factor de ineficiencia. El capítulo 4 describe cada uno de los pasos de la estrategia *Scheme*. El capítulo 5 explica detalladamente la estrategia adaptativa  $(a,b,c)$ -*Scheme* y su algoritmo, así como los

resultados teóricos de la estrategia. En el capítulo 6 se muestra el análisis del problema de calendarización. El capítulo 7 trata sobre los pasos de la implementación del calendarizador. En el capítulo 8 se muestran los resultados de la simulación de las diferentes estrategias. Por último el capítulo 9 contiene las conclusiones y sugerencias para trabajo futuro. Además, en el documento se incluyen tres apéndices. El apéndice A contiene las graficas de los resultados experimentales con conjuntos de 500 tareas y tiempos de procesamiento no unitarios. En el apéndice B se describe el uso de la herramienta diseñada llamada *Scheduling*. Finalmente, en el apéndice C se describen las características de la computadora paralela *cicese2000*, así como el porcentaje de uso de sus procesadores al paralelizar las aplicaciones solicitadas por sus usuarios durante un periodo de 6 meses.

## II Calendarización

El problema de calendarización surge por la necesidad de minimizar el tiempo de procesamiento de un conjunto de tareas que se debe distribuir entre los procesadores tanto en tiempo como en espacio.

La calendarización determina qué procesador y cuándo debe ejecutar cierta tarea. De acuerdo a la información que se tiene disponible antes de iniciar la calendarización, ésta se puede clasificar como *determinística* o *no determinística*.

Nos referimos a calendarización *determinística* cuando se conoce toda la información del sistema antes de llevar a cabo la ejecución de las tareas. Es decir, se conoce el tiempo de ejecución de cada tarea, así como las relaciones de precedencia que existen entre las tareas y los costos de comunicación (El Rewini *et al.*, 1994).

Por otra parte, se habla de calendarización *no determinística* cuando se desconoce cierta información del sistema antes de iniciar la ejecución del programa. Únicamente en este esquema se tiene conocimiento del tiempo de ejecución de las tareas, de la topología del grafo y de los costos de comunicación conforme se ejecuta el programa. Por lo tanto, el problema de calendarización no determinística es más complicado que el problema de calendarización determinística (El Rewini *et al.*, 1994).

El problema no determinístico surge, por ejemplo, en el caso de ciclos o condiciones de decisión al desconocer el número de iteraciones antes de ejecutar las tareas, lo que puede provocar un sobre costo en la comunicación o bien retrasar el proceso de calendarización.

La calendarización no determinística se puede clasificar en:

- Estática. La información sobre el grafo de tareas se debe estimar antes de la ejecución del programa. Se conoce como *grafo*, al conjunto de nodos y arcos, donde los nodos representan las tareas y los arcos las relaciones que conectan a los nodos.
- Dinámica. Las decisiones de calendarización se toman mientras el programa se ejecuta. Las tareas se asignan a los procesadores y se determina el mejor momento para su ejecución conforme se ejecuta el programa.
- Híbrida. Es una combinación de los modelos estático y dinámico. Se lleva a cabo estáticamente un pre-procesamiento como guía para el calendarizador dinámico.

Obsérvese que los tipos de calendarización no determinística mencionados difieren por el momento en que se toma la decisión de calendarización

### ***II.1 Calendarización de Lista***

La calendarización de lista es una estrategia que asigna una prioridad a cada tarea, con la cual se forma una lista descendente de acuerdo a la prioridad de las tareas. Se asigna al primer procesador disponible la tarea con mayor prioridad en la lista de tareas pendientes; además, en el caso de tareas dependientes los predecesores de dicha tarea deben haber concluido su ejecución. Cuando más de una tarea tiene asignada la misma prioridad, se selecciona arbitrariamente una tarea a ejecutarse. En caso de que dos procesadores traten de tomar la misma tarea al mismo tiempo, el procesador con menor índice toma la tarea.

El calendarizador puede aplicar diferentes criterios en la asignación de prioridad de las tareas, así como en la selección del procesador que habrá de ejecutar la tarea. Por lo tanto, se pueden generar calendarios con longitudes diferentes de acuerdo al criterio tomado.

El algoritmo de lista, dado una lista  $L$  de tareas  $T = \{T_1, T_2, \dots, T_n\}$  ordenadas por su prioridad, permite asignar tareas a los procesadores disponibles.

Como se puede observar, el algoritmo de *Graham* consiste simplemente en una asignación voraz de las tareas a los procesadores, de manera que no hay procesadores ociosos si existe una tarea pendiente por calendarizar.

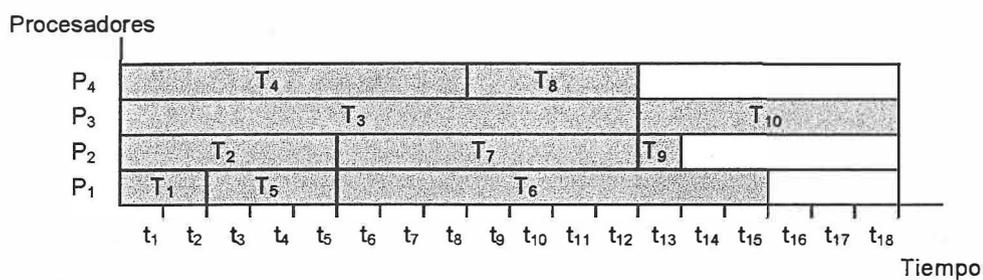
El resultado conocido para una estrategia de calendarización determinística es el cociente competitivo

$$\rho = 2 - \frac{1}{m}. \quad (1)$$

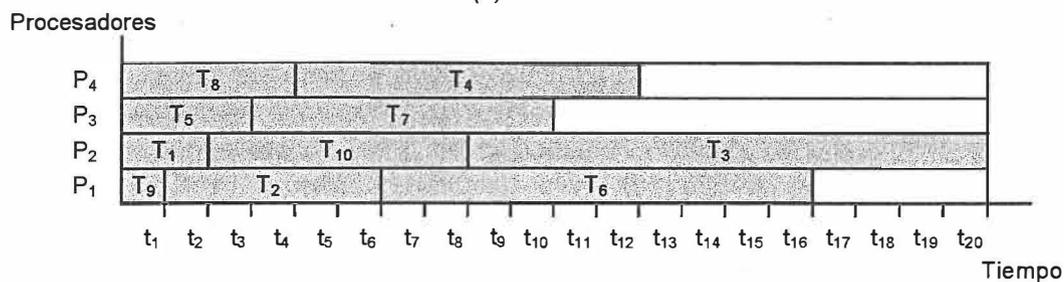
Un ejemplo de una estrategia de lista determinística es la estrategia llamada Tiempo de Procesamiento Mayor (Longest Processing Time, *LPT*, por sus siglas en inglés). Se tiene conocimiento de antemano de toda la información referente a las tareas que se van a ejecutar. En lo que respecta a la creación de la lista de tareas, como su nombre lo indica, se toma como prioridad el tiempo de procesamiento de las tareas. A todas aquellas tareas que requieren mayor tiempo de procesamiento se les asigna mayor prioridad.

Por otra parte, para ilustrar la calendarización de lista, en la figura 1 se muestra un calendario, en forma de gráfica de Gantt, con 10 tareas  $T = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}\}$  y 4 procesadores disponibles. Los cuadros de color blanco significan tiempos ociosos de los procesadores; en cambio, el color gris indica que un procesador está ejecutando una tarea. El tiempo de procesamiento requerido por cada tarea es  $P = \{2, 5, 12, 8, 3, 10, 7, 4, 1, 6\}$ , respectivamente, donde la lista de tarea se ordena de acuerdo a los siguientes criterios:

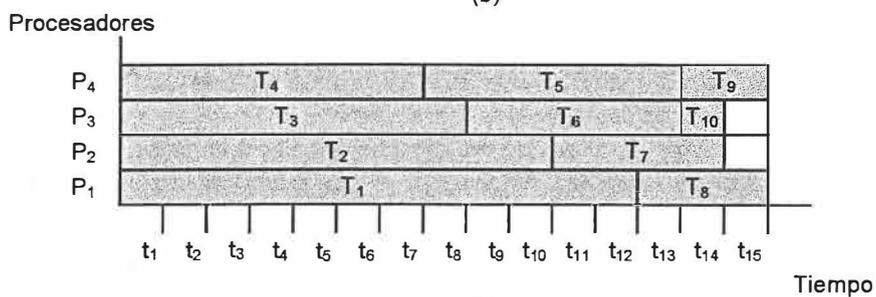
- De acuerdo a la llegada de las tareas (ver figura 1a). La lista de tareas queda de la siguiente forma  $L = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}\}$ .
- De acuerdo al menor tiempo de procesamiento de cada tarea (ver figura 1b). Para este caso la lista de tareas es  $L = \{T_9, T_1, T_5, T_8, T_2, T_{10}, T_7, T_4, T_6, T_3\}$ .
- De acuerdo al mayor tiempo de procesamiento de cada tarea (ver figura 1c). Este caso corresponde a la estrategia LPT mencionada anteriormente, y su lista de tareas es  $L = \{T_3, T_6, T_4, T_7, T_{10}, T_2, T_8, T_5, T_9, T_1\}$ .



(a)



(b)



(c)

Figura 1. Gráfica de Gantt. En la figura la el criterio para asignar las tareas es de acuerdo a su llegada al sistema. En la figura 1b el criterio que se aplica es de acuerdo al menor tiempo de procesamiento de cada tarea. En la figura 1c se toma como criterio el mayor tiempo de procesamiento de la tarea.

### III Modelo de Tareas Maleables y técnica de distribución

#### Gang

Existen diversas formas para asignar los procesadores a las tareas. En este capítulo hablamos sobre la técnica *Gang* y del modelo de tareas maleables.

Es importante mencionar que se puede evaluar qué tan eficiente es una estrategia de calendarización por medio de los factores llamados *cociente de desempeño* y *factor de ineficiencia*. Más adelante se describe en qué consiste cada uno de éstos.

En las últimas décadas se han desarrollado muchos trabajos con el objetivo de brindar una mejor solución al problema de calendarización de aplicaciones paralelas. Con el fin de no afectar el desempeño de estas aplicaciones se busca distribuir adecuadamente las tareas entre los procesadores, de manera que éstos se encuentren siempre ocupados. Al mismo tiempo se busca minimizar los volúmenes de comunicación y los problemas de sincronización generados por la distribución de carga. Lo anterior con el objeto de lograr la calendarización eficaz de la aplicación paralela.

#### III.1 Técnica de distribución *Gang*

*Gang* es una forma de asignar los procesadores a las tareas, llamada también co-calendarización, la cual consiste en otorgar simultáneamente durante la misma unidad de tiempo todos los procesadores a cada una de las tareas que forman una aplicación (Drozdowski *et al.*, 1996). Una de las ventajas de esta calendarización es el hecho de que se comparte eficientemente los procesadores (Silva *et al.*, 1998a).

En este tipo de calendarización, una tarea se divide en hilos y cada hilo de ejecución de la tarea se calendariza en un procesador independiente, ya que cada tarea se asigna a la máquina paralela completa por una unidad de tiempo antes de que ésta se interrumpa (Silva *et al.*, 1998a).

Se ha propuesto este tipo de calendarización como una solución práctica al problema de calendarización que permite la posibilidad de llegadas arbitrarias de nuevas tareas (Silva *et al.*, 1998a).

Por otra parte, este tipo de calendarización, analizada recientemente en Silva *et al.*, (1998b), supone tareas multiprocesadores, debido a que más de un procesador se utiliza simultáneamente, las cuales se asignan al número total de procesadores. La función “*felicidad*” (*Happiness*, por su nombre en inglés) definida en Silva *et al.*, (1998a), probó que ésta es la mejor estrategia de calendarización para tareas multiprocesadores interrumpibles.

Una forma eficiente de paralelizar las aplicaciones es aplicando el modelo de tareas maleables. En este modelo las tareas se pueden asignar a más de un procesador, con la ventaja de que el número de procesadores puede cambiar.

A continuación detallamos en que consiste este modelo.

### ***III.2 Modelo de tareas maleables***

Antes de iniciar la descripción del modelo de tareas maleables, es necesario identificar el tipo de tareas que maneja este modelo. Una *tarea maleable* se considera como el reagrupamiento de un conjunto de tareas secuenciales ejecutadas en paralelo por un número de procesadores seleccionados por la estrategia de calendarización (Mounié, 2000).

Uno de los aspectos que han motivado el uso del modelo de tareas maleables es la característica de que permite unificar las tareas uni-procesador y las tareas que requieren más de un procesador para su ejecución (Lepere *et al.*, 2000). Además de considerar que este modelo difiere del modelo tradicional, en el cual una tarea la ejecutaba un procesador a la vez, otra característica importante de este modelo es el hecho de que permite ocultar el comportamiento complejo de la ejecución de un conjunto de tareas, debido a que los costos incurridos por la paralelización de las tareas maleables se consideran implícitamente por la función  $\mu$ , llamada *factor de ineficiencia* (Mounié *et al.*, 1999). El comportamiento de esta función se analiza más adelante. En otras palabras, podemos decir que el conjunto de  $n$  tareas maleables se puede ejecutar con un número  $m$  de procesadores. El tiempo de procesamiento de la tarea  $T_i$  depende del número de procesadores asignados a la tarea (Blazewicz *et al.*, 2000). Llamaremos  $p_m^i$  al tiempo requerido para procesar una tarea  $T_i$  con  $m$  procesadores. El trabajo de una tarea ejecutada en  $m$  procesadores es el producto de

$$w_m^i = mp_m^i. \quad (2)$$

Ahora bien, además del modelo de tareas maleables existen otros modelos donde las tareas se pueden ejecutar por más de un procesador, tal es el caso del modelo de tareas multiprocesador y del modelo de tareas divisibles. A continuación mostramos una breve descripción de dichos modelos con el objeto de identificar claramente en qué consiste cada uno de éstos y cuál es su diferencia con respecto al modelo de tareas maleables:

- *Modelo de tareas multiprocesador.* Las tareas se pueden asignar en más de un procesador en una misma unidad de tiempo (Drozdowski, 1996). Cada tarea requiere un

número fijo de procesadores, dicho número puede variar de 1 a  $m$ , donde  $m$  es el número total de procesadores.

- *Modelo de tareas divisibles*. El número de procesadores asignados a la tarea puede tomar valores entre 0 y  $m$ ; su valor depende de la política de calendarización. Las tareas comparten los procesadores como un recurso divisible continuo (Blazewicz *et al.*, 1999). Este tipo de tareas se puede dividir con granularidad arbitraria en partes independientes, las cuales se resuelven en paralelo por medio de sistemas distribuidos (Blazewicz *et al.*, 1999).

Como podemos observar, el modelo de tareas maleables es una combinación de los modelos mencionados anteriormente. El número de procesadores asignados puede tomar valores entre 1 y  $m$ . A diferencia del modelo de tareas multiprocesador, este número no es fijo.

Una tarea maleable asignada a  $m$  procesadores muestra un tiempo de procesamiento mayor que el tiempo de procesamiento que se tiene al asignar la tarea a un solo procesador y dividirla entre  $m$  procesadores.

### ***III.3 Factor de ineficiencia $\mu$***

El sobre costo por la administración del paralelismo se abstrae implícitamente en el parámetro llamado *factor de ineficiencia*, el cual además permite ocultar el comportamiento complejo de un subconjunto de tareas (Lepere *et al.*, 2000). Con este factor se busca un equilibrio entre los sistemas complejos reales (los cuales no se pueden formalizar debido a la gran cantidad de parámetros que contemplan) y un modelo sencillo.

Dicho equilibrio se puede lograr apoyándose en el hecho de que el factor de ineficiencia abstrae todos los gastos incurridos en la paralelización.

El factor de ineficiencia representa el factor de expansión del área total de un conjunto de tareas mientras éstas utilizan recursos paralelos, en otras palabras, podemos decir que representa el pago que se debe hacer por paralelizar dichas tareas. Otra forma de verlo es como la pérdida de tiempo durante la ejecución de una tarea en más de un procesador.

**Definición 1** Para una tarea  $T_i$  asignada a  $m$  procesadores, su factor de ineficiencia  $\mu_m^i$  se define como

$$\mu_m^i = \frac{m * p_m^i}{p_i}, \quad (3)$$

donde  $p_i = p_1^i$ .

El factor de ineficiencia se considera monótonico, debido a que incrementa con el número de procesadores al menos hasta un cierto límite (Blazewicz *et al.*, 2000), y decrementa con el tamaño de la tarea paralela. Es decir, que al incrementar el número de procesadores asignados a una tarea, el tiempo de procesamiento paralelo de la tarea decrementa. Por otra parte, al asignar un mayor número de procesadores a una tarea, incrementa su área computacional o trabajo (Lepere *et al.*, 2000). (Véase figura 2). Se define como área computacional o trabajo al producto de multiplicar el tiempo de procesamiento por el número de procesadores, como se representa en la ecuación (10).

El factor de ineficiencia afecta al tiempo de procesamiento de una tarea maleable, lo cual se muestra en la figura 2. Para nuestro ejemplo, considérese que la tarea la van a ejecutar dos

procesadores. Llamaremos  $p_1^1 = t_2$ , al tiempo de procesamiento si la tarea se ejecuta por un solo procesador.

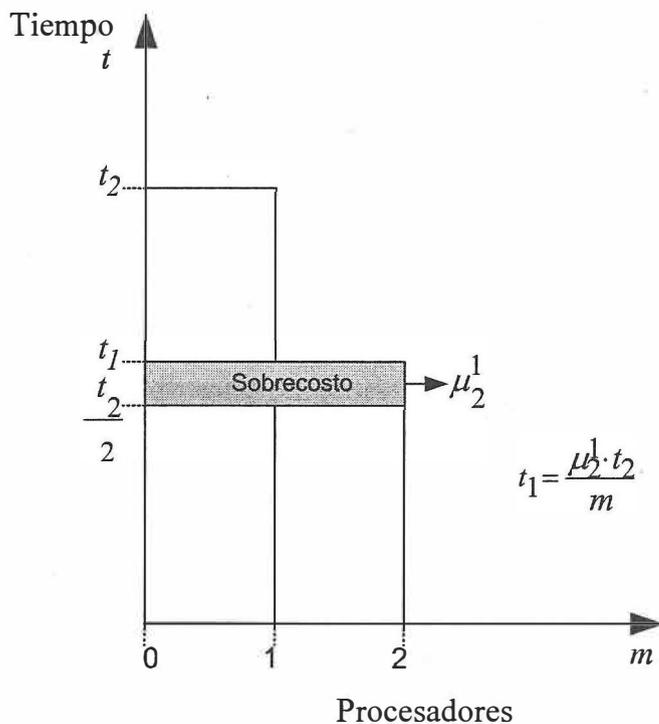


Figura 2. Factor de ineficiencia.

$\mu_2^1$  es el factor de ineficiencia por paralelizar la tarea con dos procesadores y  $t_1$  es el tiempo de procesamiento de la tarea si se asigna a dos procesadores. Este tiempo incluye el pago por ejecutar la tarea en dos procesadores.

Por otra parte, en nuestro documento asumimos que el factor de ineficiencia depende de que el número de procesadores no decremente:  $\mu_m^i \leq \mu_{m+1}^i$ , para cualquier  $m$ . Esta suposición excluye tareas con aceleración super lineal (*speedup*, por su nombre en inglés).

Llamaremos  $\mu_m$  al  $\max_{1,n} \{\mu_m^i\}$  y  $p$  al  $\max_{1,n} \{p_i\}$ . Considerando que  $p_m^i$  depende de que  $m$  no se incremente, al menos hasta un número razonable de procesadores. Por lo tanto,

$$(m+1)p_{m+1}^i \leq \left(1 + \frac{1}{m}\right)mp_m^i \Rightarrow \mu_{m+1}^i \leq \left(1 + \frac{1}{m}\right)\mu_m^i. \quad (4)$$

De acuerdo a la función de trabajo (1), se puede decir que

$$\mu_m^i = \frac{w_m^i}{w_1^i}. \quad (5)$$

## IV Scheme

*Scheme*, calendarización multiprocesador eficiente (*Scheduling Multiprocessors Efficiently*, por su nombre en inglés), es una estrategia propuesta por Rapine *et al.*, (1998). A continuación se muestra parte del material presentado en Rapine *et al.*, (1998).

Esta estrategia es una combinación de la calendarización *Graham* y de la calendarización *Gang*, que tiene como objetivo optimizar el tiempo total de ejecución de un conjunto de tareas independientes aplicando para ello un algoritmo de dos etapas.

Cuando la calendarización *Graham* se aplica, cada tarea se ejecuta únicamente en un solo procesador, de manera que no hay procesadores ociosos mientras existan tareas listas para ejecutarse.

Por otra parte, cuando se aplica la calendarización *Gang*, se asumen tareas de tipo paralelizable, las cuales las ejecutan el número total de procesadores en el sistema. A continuación mostramos el cociente competitivo de la estrategia de calendarización *Gang*, al comparar el calendario generado por la estrategia con su calendario óptimo.

Considérese que el tiempo total de procesamiento se representa por

$$W_{tot} = \sum_{i=1}^n p_i. \quad (6)$$

Al asignar las tareas en más de un procesador, es necesario hacer un pago debido a la administración del paralelismo, el cual se representa por el factor de ineficiencia; por lo tanto, el calendario o tiempo de ejecución utilizando la calendarización *Gang* se puede expresar de la siguiente forma:

$$C_{Gang} = \sum_{i=1}^n \frac{\mu_m^i}{m} p_i \leq \frac{\mu_m}{m} W_{tot}. \quad (7)$$

Nótese que la longitud del calendario óptimo  $C_{opt}$  está acotado inferiormente por

$$C_{opt} \geq \frac{W_{tot}}{m}. \quad (8)$$

De acuerdo a la longitud del calendario generado por la estrategia y a la longitud del calendario óptimo, podemos observar que el cociente competitivo de la calendarización *Gang* se expresa de la siguiente forma:

$$\rho_{Gang}^* \leq \mu_m \quad (9)$$

A continuación se describe el algoritmo de dos etapas utilizado por la estrategia *Scheme* para calendarizar un conjunto de tareas independientes:

- *Etapla 1.* Cuando  $n \geq m$  cada tarea la ejecuta un solo procesador. En este momento todos los procesadores están ocupados, por lo tanto se tiene una eficiencia de uno. El algoritmo aplicado es el conocido algoritmo de *Graham* (ver figura 3a).
- *Etapla 2.* Esta etapa inicia en la unidad de tiempo  $t$ , cuando se tienen menos de  $m$  tareas en el sistema. En este momento se puede tener una ineficiencia como consecuencia de los procesadores ociosos en el sistema. Para evitar los tiempos ociosos *Scheme* interrumpe a cada una de las tareas, las cuales posteriormente se asignan a los  $m$  procesadores de acuerdo a la estrategia *Gang* (ver figura 3b).

La figura 3 muestra cómo las tareas independientes se ejecutan utilizando *Scheme*. Esta figura se obtuvo de Rapine *et al.*, (1998). Las áreas sombreadas representan tareas que concluyeron su ejecución, mientras que las áreas sin sombra representan las tareas que se interrumpen al momento de existir un procesador ocioso en el sistema. En la figura 3a, cada

tarea la ejecuta un solo procesador hasta terminar su ejecución, sin tomar en cuenta que existen procesadores ociosos a partir del tiempo  $t$ . En la figura 3b, cada tarea la ejecuta un procesador durante la etapa 1. La etapa 2 inicia cuando un procesador está libre y no se tienen más tareas por asignar. Aquí se interrumpen las tareas y cada tarea la ejecutan todos los procesadores.

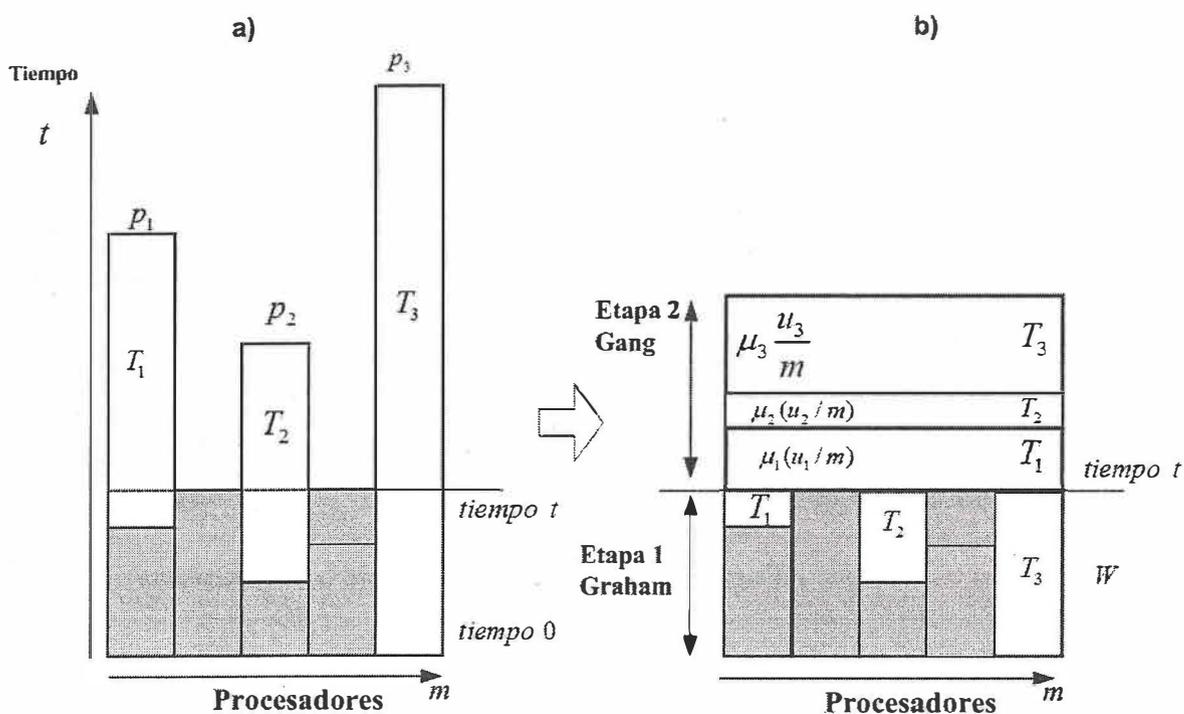


Figura 3. Estrategia *Scheme*. La figura de la izquierda representa cómo se ejecutan las tareas si se aplica el algoritmo de Graham únicamente. La figura de la derecha muestra cómo se lleva a cabo la estrategia *Scheme*. Al inicio, cada tarea la ejecuta un solo procesador (etapa 1), a partir del tiempo  $t$ , a cada una de las tareas que aún no terminan su ejecución se le asignan todos los procesadores (etapa 2). La etapa 2 inicia cuando se tiene un procesador libre.

$u_i$  es el tiempo de procesamiento que falta por ejecutarse de la tarea  $T_i$  al iniciar la etapa 2,  $p_i$  es el tiempo de procesamiento de la tarea  $T_i$ , si ésta la ejecuta únicamente un procesador y  $\mu_i \frac{u_i}{m}$  representa el tiempo en que se ejecuta la tarea al paralelizarla en  $m$  procesadores.

Suponiendo que la *etapa* 1 inicia en la unidad de tiempo 0 y termina en la unidad de tiempo  $t$ , el trabajo realizado en esta etapa se denota como

$$W = m \cdot p. \quad (10)$$

Considerando que después del tiempo  $t$  pueden existir menos de  $m$  tareas que no han terminado su ejecución, denotaremos a esas tareas como  $T_1, \dots, T_k, k \leq m-1$  y llamaremos  $u_1, \dots, u_k$  al tiempo que falta por ejecutarse de dichas tareas. Por lo tanto, la longitud del calendario aplicando la estrategia está acotado de la siguiente manera:

$$C_{Scheme} \leq \frac{W}{m} + \frac{\mu_m}{m} \sum_{i=1}^k u_i, \quad (11)$$

donde  $W$  representa el trabajo total cuando todos los procesadores están ocupados.

Considérese que la cantidad total de trabajo para ejecutar todas las tareas se denota por

$$W_{tot} = W + \sum_{i=1}^k u_i, \quad (12)$$

donde  $k \leq m-1$ .

Por otra parte, la cantidad máxima de trabajo o tiempo de las tareas que faltaban por ejecutarse después de  $t$  se denota como

$$u = \max \{u_1, \dots, u_k\}. \quad (13)$$

Despejando el valor de  $\bar{W}$  en la ecuación (12) y sustituyendo su valor en la ecuación (11), podemos observar que la longitud del calendario generado por la estrategia se acota de la siguiente forma:

$$C_{Scheme}(m) \leq \frac{W_{tot}}{m} + \frac{\mu_m - 1}{m} \sum_{i=1}^{i=k} u_i \leq \frac{W_{tot}}{m} + \frac{m-1}{m} (\mu_m - 1)u. \quad (14)$$

La cota inferior de un calendario óptimo sin paralelización de tareas está dada por

$$C_{opt} \geq \frac{W_{tot}}{m}. \quad (15)$$

Otra cota que se considera para el calendario óptimo es

$$C_{opt} \geq u. \quad (16)$$

Ahora bien, considérese que se tiene un caso compuesta por  $m-1$  tareas de tamaño  $m$  y  $m$  tareas de tamaño 1, donde las tareas pequeñas se ejecutan primero y se aplica la estrategia *Gang* para las  $m-1$  tareas de tamaño  $m$ .

De acuerdo a lo anterior, sustituimos los valores en la ecuación (11) y obtenemos que la longitud del calendario generado por la estrategia *Scheme* está dado por

$$C_{Scheme} = 1 + (m-1)\mu_m. \quad (17)$$

Sustituyendo las ecuaciones (16) y (17) en la ecuación para obtener el cociente competitivo para la estrategia *Scheme*, tenemos que

$$\rho_{Scheme}^* \leq \left(1 - \frac{1}{m}\right)\mu_m + \frac{1}{m}. \quad (18)$$

## V $(a,b,c)$ - Scheme

La estrategia adaptativa llamada  $(a,b,c)$ - Scheme fue propuesta por Tchernykh *et al.*, (2002). Ésta consiste en una modificación de la estrategia de dos etapas conocida como *Scheme* propuesta en Rapine *et al.*, (1998). Es decir, durante la etapa 1 de  $(a,b,c)$ - Scheme se aplica una estrategia de calendarización llamada estrategia 1 (ST1) y durante la etapa 2 el sistema cambia a una segunda estrategia llamada ST2. Esta nueva estrategia introduce tres nuevos parámetros llamados  $(a,b,c)$  que afectan la calendarización de las tareas.

Durante el estudio de esta estrategia, no se va a considerar el caso de tareas con restricciones de precedencia, o bien el problema dinámico con arribo de nueva tareas donde el número de nuevas tareas en el sistema se incrementa, debido a que esto originaría un cambio recursivo de la estrategia ST2 a la estrategia ST1. Debemos mencionar que para ST1 se utilizará el algoritmo de *Graham* y para el caso de ST2 podrá utilizarse tres diferentes estrategias: *Gang*, *Maleable* o *Paralelizable*.

Antes de describir en qué consiste cada una de las etapas del algoritmo utilizado por la estrategia adaptativa  $(a,b,c)$ -Scheme, es necesario primeramente describir el significado de los parámetros  $a$ ,  $b$  y  $c$ , manejados en dicha estrategia.

### V.1 Parámetro $(a)$

Este parámetro representa el número de procesadores ociosos que deben de existir en el sistema para llevar a cabo el cambio de ST1 a la estrategia ST2, donde dicho parámetro puede tomar el valor de  $0 \leq a \leq m$  (ver tabla I).

Este cambio de estrategia se lleva a cabo con el objetivo de obtener una mejor eficiencia al calendarizar las tareas, evitando que exista un gran número de procesadores ociosos en el sistema y con ello la ineficiencia. Se dice que se tiene eficiencia igual a 1 cuando todos los procesadores están ocupados.

Tabla I. Valores del parámetro  $a$  en la estrategia  $(a,b,c)$ -Scheme.

$a$	Estrategia
0	Inicia algoritmo de la estrategia ST2
$1 \leq a < m$	$(a,b,c)$ - Scheme
$m$	Algoritmo de la estrategia ST1=Graham

Este parámetro nos permite buscar un balance entre la posibilidad de paralelizar un gran número de tareas, lo que trae consigo un sobrecosto mayor (esta situación puede ocurrir al asignar al parámetro  $a$  un valor pequeño) y la posibilidad de iniciar la paralelización de las tareas una vez que se tiene un número grande de procesadores ociosos en el sistema (en este caso se reduce el número de tareas a paralelizar).

## V.2 Parámetro (b)

Este parámetro representa el número de procesadores que una tarea requiere para ejecutarse al momento de su paralelización de acuerdo a la clasificación de Feitelson *et al.*, (1997). Además se utiliza para distinguir entre diferentes tipos de tareas. El valor de este parámetro depende de su tipo, por ejemplo:

- *Fijo*. Un número fijo de procesadores se asigna a la tarea y este número no cambia hasta que la tarea termina. El área computacional se representa por un rectángulo en

el diagrama de Gantt. Existen diversas políticas que permiten llevar a cabo la asignación de los procesadores:

- 1 (conocida como estrategia *Graham*). Únicamente un procesador se asigna a una tarea.
- $m$  (conocida como estrategia *Gang*). El número total de procesadores en el sistema se asigna a cada tarea.
- $2, 3, \dots, k$ . A una tarea se le asignan  $k$  procesadores, donde  $a < k < m$ . Dicho valor se puede asignar en forma externa al calendarizador ya sea por:
  - El usuario, por ejemplo, para una tarea rígida (Feitelson *et al.*, 1996).
  - El administrador del sistema de cómputo, aplicando, por ejemplo, una política de *Equi-partición*, donde a cada tarea se le asigna un número igual de procesadores, o ya sea *por de facto*, cuando la partición la realiza el sistema operativo.
- *Moldeable*. La tarea la puede ejecutar cualquier número de procesadores, pero una vez asignado el número a la tarea por el calendarizador, éste se mantiene constante durante la ejecución de la tarea (Feitelson *et al.*, 1996).
- *Cube-S*. El número de procesadores que requiere una tarea es de acuerdo a una potencia de 2. Por ejemplo, 1, 2, 4, etc., procesadores (Blazewicz *et al.*, 2000; Chen *et al.*, 1998).
- *Maleable* (también llamado *modelo cualquiera* (Blazewicz *et al.*, 2000)). El número de procesadores requeridos por la tarea no se asigna desde un inicio. Éste depende del número de procesadores libres al momento de la asignación, del cambio en los requerimientos o de la carga. En cualquier instante de tiempo en que

se disponga de otro procesador libre, la misma tarea se puede interrumpir, redistribuir y continuar con su ejecución en un número diferente de procesadores. Si observamos su área computacional en un diagrama de Gantt ésta se representa por una unión de rectángulos. Además, podemos observar que las tareas maleables dan al sistema una completa flexibilidad en cuanto a la asignación de los recursos.

- Evolutiva. La tarea requiere diferente número de procesadores durante su ejecución (Feitelson *et al.*, 1996). Esta opción se refiere a tareas que cambian su paralelismo durante diferentes etapas de su ejecución. Llamadas especiales de sistema, pueden ser aquellos puntos en el programa donde la tarea cambia su paralelismo.

De acuerdo a la clasificación anterior, la tabla II muestra el número de procesadores que se puede asignar a una tarea de acuerdo al tipo de  $b$ , en el cual basaremos nuestro caso de estudio.

Tabla II. Valores del parámetro  $b$ .

Tipo de $b$	Valor de $b$
1 (Graham)	1
$m$ (Gang)	$m$
Maleable	el número de procesadores libres, donde $1 \leq b \leq m$

### V.3 Parámetro (c)

Este parámetro muestra la estrategia en la selección de la tarea para la paralelización. Además, se puede utilizar para indicar la selección de la cola de una clase de tarea de acuerdo a ciertas prioridades, al número de procesadores necesarios, etc. A continuación mostraremos algunas de las estrategias permitidas por este parámetro:

- **Lista (0).** No existe regla alguna para la selección de la tarea a paralelizar (calendarización de lista arbitraria).
- **Sencilla (1).** En cada instante de tiempo, únicamente una tarea se paraleliza en el sistema. Cuando existe más de un procesador ocioso, el sistema selecciona una tarea; por ejemplo, una tarea maleable y la asigna a los procesadores disponibles. Tiempo después cuando existen más procesadores disponibles, la misma tarea se interrumpe para ser redistribuida y continuar su ejecución, pero ahora en un número mayor de procesadores. Esta estrategia permite maximizar el número de procesadores asignados a la tarea seleccionada.
- ***mt* (múltiple).** Cada vez que un procesador está disponible, se selecciona una nueva tarea ya sea de la misma cola o de alguna diferente para que esta tarea se paralelice. Esta estrategia minimiza el número de procesadores asignados a la tarea, ya que la eficiencia de las tareas paralelas decrementa conforme incrementa el número de procesadores asignados.
- **(2,3,...,k).** Un número  $k$  de tareas se selecciona para ejecutarse en  $m$  procesadores, donde  $1 < k < m$ .
- **Todas.** Todas las tareas restantes se seleccionan para ejecutarse en  $m$  procesadores. Se supone que el número de tareas es menor o igual que el número de procesadores. Por ejemplo, se puede asignar  $n/m$  procesadores a cada tarea, donde  $n \leq m$ .

Una vez definidos los parámetros a utilizar en esta estrategia, a continuación describimos las dos etapas que conforman el algoritmo utilizado por la estrategia adaptativa *(a,b,c)-Scheme*.

#### V.4 Algoritmo

- **Etapa 1a.** Cuando  $n \geq m$ , cada tarea se asigna a un procesador de acuerdo a ST1. Por lo tanto, las tareas maleables o paralelizables se ejecutan secuencialmente, no existen tiempos ociosos ya que todos los procesadores están ocupados.
- **Etapa 1b.** Cuando  $n < m$ , pero existe un número de procesadores ociosos menor que  $a$ , donde  $0 \leq a \leq m$ . Cada tarea se asigna a un procesador de acuerdo a ST1.
- **Etapa 2.** Cuando el número de procesadores ociosos en el sistema es mayor o igual que  $a$ , se cambia la estrategia ST1 para evitar que aumente el número de procesadores ociosos. Tanto el número de procesadores asignados a cada tarea, así como la estrategia ST2 dependen del parámetro  $b$ . Por otro lado, el parámetro  $c$  se encarga de definir la estrategia para la selección de la tarea.

A manera de ilustrar como se lleva a cabo la calendarización de un conjunto de tareas aplicando la estrategia  $(a,b,c)$ - *Scheme*, a continuación se muestran dos casos con diferentes valores de  $a$  e igual valor de  $b$ , cuya  $c = 0$  y donde  $m = 6$ :

- **Caso 1.** Donde,  $a = 1$ ,  $b = m$  y  $c = 0$ .

En la figura 4, del lado izquierdo se muestra la calendarización de un conjunto de tareas aplicando el algoritmo de Graham. Es decir, cada una de las tareas la ejecuta un solo procesador hasta concluir su ejecución. Observemos que hasta el tiempo  $t$ , todos los procesadores están ocupados (área sombreada). En cambio, a partir del tiempo  $t'$  sólo existen 5 tareas pendientes de concluir su ejecución (área no sombreada), por lo tanto existe un procesador ocioso.  $p_i$  es el tiempo que falta por ejecutar de cada tarea.

En la figura de la izquierda, las tareas continúan su ejecución sin interrupción alguna a pesar de que se tienen procesadores ociosos.

Por otra parte, en la figura de la derecha se observa que en el momento que existe un procesador ocioso la ejecución de las cinco tareas que aún no concluyen se interrumpe (tiempo  $t$ ). A partir de este momento cada una de esas tareas se ejecuta en  $m$  procesadores de acuerdo a la estrategia Gang, como se muestra en el área no sombreada en la figura de la derecha. Observe que durante la etapa 2 en la figura de la izquierda todos los procesadores están ocupados, cosa que no ocurre en la figura de la izquierda.  $\delta$  representa el tiempo durante el cual se ejecuta una tarea por un procesador a partir de que existen procesadores ociosos, pero no los requeridos para

hacer cambio de estrategia.  $\sum_{i=1}^k \frac{\mu_m}{m} (p_i - \delta)$  es el tiempo en el que se paralelizan las

tareas pendientes de ejecutar aplicando la estrategia Gang.

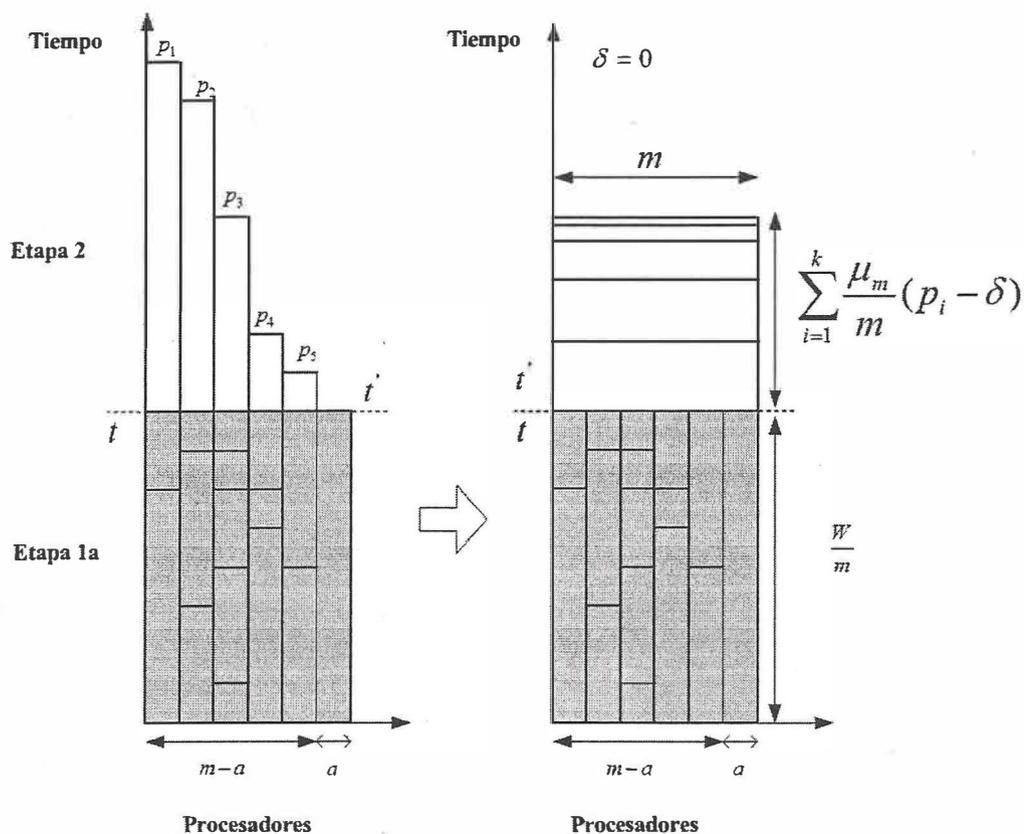


Figura 4. Estrategia  $(a,b,c)$ - Scheme. La figura de la izquierda ilustra la calendarización de un conjunto de tareas. Cada tarea requiere un solo procesador y se aplica el algoritmo de Graham. En la figura de la derecha se muestra el calendario generado por la estrategia  $(a,b,c)$ - Scheme para el mismo conjunto de tareas. Las tareas se interrumpen en el momento que existe un procesador ocioso, como se observa en el tiempo  $t$ . Las áreas no sombreadas representan una tarea que se interrumpió en  $t$  y que a partir de  $t'$  se paraleliza en los  $m$  procesadores de acuerdo a la estrategia Gang.

$p_i$  es el tiempo que falta por ejecutar de la tarea  $i$  al momento de existir un procesador ocioso en el sistema. Dicha tarea la ejecuta un solo procesador.

$t$  es el momento hasta el cual todos los procesadores están ocupados. No existen procesadores ociosos. La eficiencia es igual a 1.

$t'$  es el tiempo a partir de que se hace cambio de estrategia, momento en el que se tienen los  $a$  procesadores ociosos necesarios para hacer cambio de estrategia.

$\delta$  representa el tiempo comprendido a partir del momento en que existen procesadores ociosos hasta el momento antes de hacer cambio de estrategia. Para este caso  $\delta = 0$ , ya que se cambia de estrategia cuando existe un procesador ocioso, por lo tanto no existe etapa 1b.

$\sum_{i=1}^k \frac{\mu_m}{m} (p_i - \delta)$  representa la sumatoria del tiempo paralelo en que se ejecuta cada una de las tareas cuya ejecución se interrumpe en el momento  $t$ . Representa el intervalo de tiempo durante el cual las tareas se ejecutan de acuerdo a la estrategia Gang.

- **Caso 2.** Donde  $a = 4$ ,  $b = m$  y  $c = 0$ .

En la figura 5 se muestra como se lleva a cabo la calendarización de un conjunto de tareas al aplicar la estrategia  $(a,b,c)$ - *Scheme* (figura de la derecha), así como la calendarización al aplicar el algoritmo de Graham (figura de la izquierda). Obsérvese que en la figura de la izquierda cada tarea requiere un solo procesador para su ejecución y ésta no se interrumpe en ningún momento de la calendarización. Una vez que el procesador concluye la ejecución de la tarea se le asigna una nueva tarea que se encuentre pendiente de ejecutar.

Por otra parte, la figura de la derecha muestra claramente las etapas por las que pasa la estrategia  $(a,b,c)$ - *Scheme* para llevar a cabo la calendarización del conjunto de tareas.

- En la etapa 1a cada procesador toma una tarea, y una vez que termina de ejecutarla, toma una nueva tarea de la lista de espera; durante esta etapa todos los procesadores están ocupados.

- En la etapa 1b no se tienen tareas pendientes de ejecutar que se puedan asignar a los procesadores libres. Además el número de procesadores ociosos es menor al establecido para hacer cambio de estrategia. En nuestro ejemplo este valor corresponde a cuatro procesadores ociosos. Las tareas continúan su ejecución como en la etapa 1a.
- $t'$  representa el momento en que se tienen los cuatro procesadores libres. Es decir, es el momento en el que se hace el cambio de estrategia. Por lo tanto, se interrumpen las tareas que aún no terminan su ejecución y se asignan a los  $m$  procesadores, como se muestra en la etapa 2 de la figura de la derecha.  $(p_i - \delta)$  representa el tiempo que falta por ejecutar de cada tarea.  $p_i$  es el tiempo total de ejecución de una tarea cuando ésta se ejecuta en un solo procesador.  $\delta$  corresponde al tiempo durante el cual se ejecuta una tarea. En nuestro ejemplo únicamente dos tareas son las que se interrumpen.

En la etapa 2 se aplica un pago (factor de ineficiencia) a cada tarea que se interrumpe para que ésta se ejecute en los  $m$  procesadores. En otras palabras, si al momento de interrumpir una tarea el tiempo que le falta para concluir su ejecución es  $(p_i - \delta)$ , entonces, el tiempo para paralelizarla en  $m$  procesadores durante la etapa 2 es igual a  $\frac{\mu_m}{m}(p_i - \delta)$ .

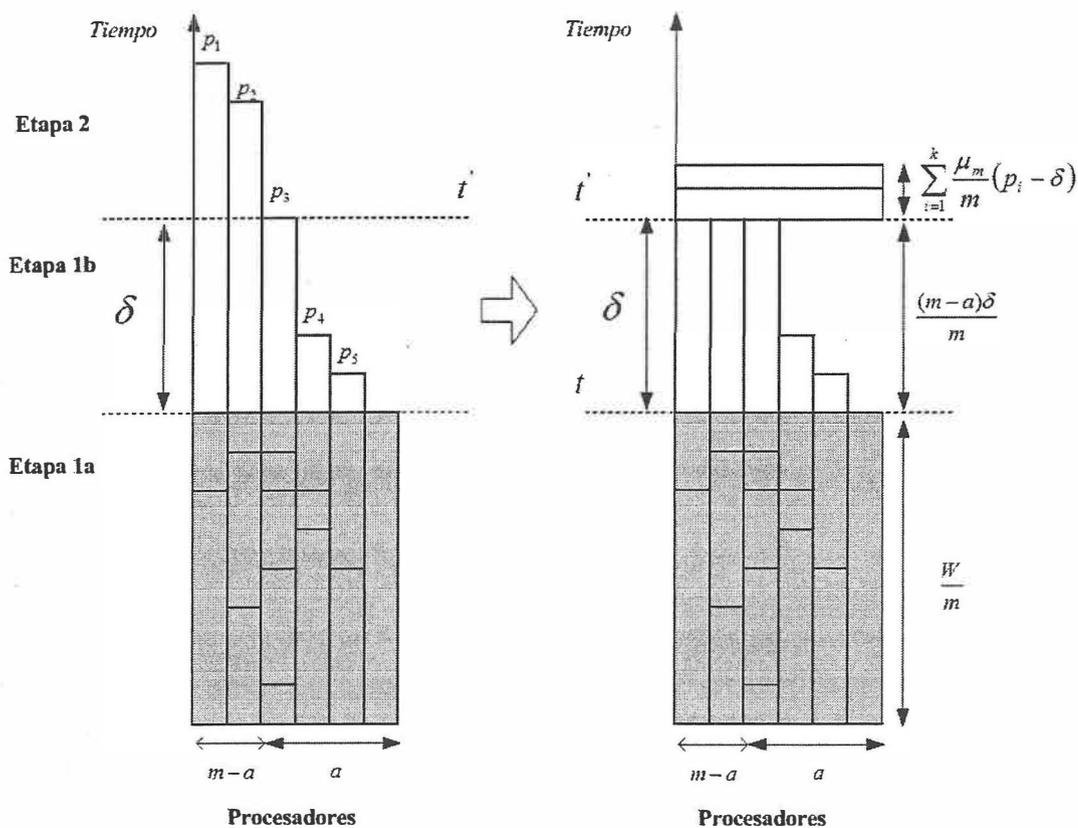


Figura 5. Estrategia  $(a,b,c)$ - Scheme, donde  $a = 4$ ,  $b = m$  y  $c = 0$ . En la figura de la izquierda cada tarea se asigna a un procesador y cuando éste termina de ejecutarla toma una nueva tarea de acuerdo al algoritmo de Graham. En la figura de la derecha las tareas se ejecutan de acuerdo a la estrategia  $(a,b,c)$ - Scheme. En la etapa 1a, cada tarea la ejecuta un procesador. En ambas figuras durante esta etapa todos los procesadores están ocupados. En la etapa 1b, existen menos de  $a$  procesadores ociosos, por lo tanto no se interrumpe la ejecución de las tareas. La duración de esta etapa se representa por  $\delta$ . En la etapa 2, se observa que a partir del tiempo  $t'$  las tareas que aún no concluyen su ejecución se interrumpen y se asigna cada una de éstas a los  $m$  procesadores, como lo muestra la figura de la derecha. Esta etapa inicia en el momento que se tienen los  $a$  procesadores ociosos.

## VI Análisis del problema

El análisis del problema de calendarización presentado en esta sección se basa en el trabajo propuesto por Tchernykh *et al.*, (2002) sobre la estrategia  $(a,b,c)$ -Scheme. A continuación se muestra parte de este material, enfocándose en particular al análisis del desempeño para los casos cuando  $b = 1$  o bien  $b = m$ , donde el parámetro  $c$  no influye en la calendarización de las tareas. Por lo tanto, estos dos casos analizados forman parte de un caso particular de  $(a,b,c)$ -Scheme llamado  $(a,b,0)$ -Scheme.

Por otra parte, debemos tener en cuenta que al trabajar con estrategias de calendarización no determinísticas, como es el caso, únicamente se conoce cierta información del problema. A partir de esta información se pueden elaborar suposiciones que sirven como apoyo para el análisis de las estrategias.

- *Suposición 1.* En cada unidad de tiempo únicamente se conocen las tareas listas para su ejecución. Al finalizar la tarea se tiene información sobre los procesadores ociosos, así como sobre la cantidad de tareas que se pueden procesar.
- *Suposición 2.* Se conoce de antemano el número de procesadores que se van a asignar a cada tarea, siempre y cuando se trabaje bajo la estrategia *Gang*.

Una propiedad obvia de  $(a,b,0)$ -Scheme es el hecho de que ésta puede cambiar de la estrategia *Graham* a la estrategia *Gang* cuando existen  $a$  procesadores ociosos.

Antes de continuar con el análisis debemos de tener en cuenta que para comparar el resultado de una estrategia dada con el valor óptimo debemos de calcular el cociente competitivo  $\rho^*$  entre las estrategias a comparar, el cual está dado por

$$C/C_{opt}. \quad (19)$$

Al buscar una cota superior para el cociente competitivo entre el calendario por aplicar la calendarización *Gang* y el óptimo de un conjunto de tareas independientes, llegamos al siguiente lema.

**Lema 1.** El cociente competitivo de la calendarización *Gang* cuando  $m > 2$  y  $n < m$  es

$$\rho_{Gang}^* \leq \mu_m \left(1 - \frac{1}{m}\right). \quad (20)$$

*Demostración.* Se puede mejora el resultado obtenido en Rapine *et al.*, (1998) para el caso específico cuando el número de tareas es menor que el número de procesadores.

Sea la longitud del calendario aplicando *Gang* dada por la siguiente expresión:

$$C_{Gang} = \sum_{i=1}^n \frac{\mu_m^i}{m} p_i \leq \frac{\mu_m n p}{m}, \quad (21)$$

y la longitud del calendario óptimo acotado inferiormente por el tiempo de procesamiento de la tarea, es decir,

$$C_{opt} \geq p. \quad (22)$$

Sustituyendo las ecuaciones (21) y (22) en la ecuación (19) obtenemos el resultado

$$\rho_{Gang}^* \leq \frac{n\mu_m}{m}. \quad (23)$$

Por otra parte, el valor máximo del cociente competitivo se alcanza cuando  $n = m - 1$ . Si sustituimos el valor de  $n$  en la ecuación (23), obtenemos el resultado buscado:

$$\rho_{Gang}^* \leq \mu_m \left(1 - \frac{1}{m}\right). \quad (24)$$

Como se menciona al inicio de la sección, sabemos que una tarea la puede ejecutar un solo procesador o bien todos los procesadores en el sistema. En la siguiente sección se muestra el caso cuando las tareas las ejecuta un solo procesador, es decir, nos referimos al caso cuando  $b = 1$ .

### VI.1 Caso $(a,1,0)$ -Scheme

Suponiendo que el número de procesadores requeridos por una tarea es igual a 1, podemos expresar el cociente competitivo para dicho caso, por medio del siguiente lema:

**Lema 2.** El cociente competitivo para el caso  $(a,1,0)$ -Scheme es

$$\rho_{(a,1,0)\text{-Scheme}}^* = 1 + \frac{a}{m}. \quad (25)$$

*Demostración.* Considérese que en un calendario obtenido aplicando *Graham* a lo más  $a$  procesadores están ociosos hasta el final, donde  $a$  puede tomar valores de  $0 \leq a \leq m$ . Además considérese que las tareas que componen el caso se pueden expresar de la siguiente manera:

$$W * m + (m - a). \quad (26)$$

En la figura 6 se puede ver como se interrumpe la ejecución de las tareas que aún no concluyen al momento que se tiene  $a$  procesadores ociosos en el sistema. En este caso tres tareas son las que se interrumpen. Después de interrumpir las tareas, cada una de éstas se asigna a un solo procesador. En este caso, observamos que antes y después de la interrupción cada tarea se asigna a un solo procesador. Este caso se conoce como  $(a,1,0)$ -Scheme.

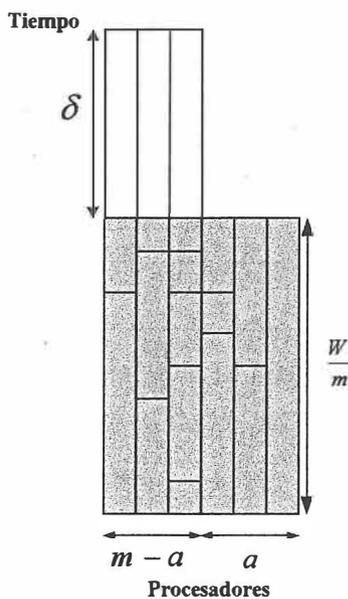


Figura 6. Ejemplo de la calendarización  $(a,1,0)$ -Scheme.

Como puede observarse, el tiempo de terminación de la estrategia se determina por

$$C_{(a,1,0)\text{-Scheme}} = \frac{W}{m} + \delta, \quad (27)$$

donde  $\delta$  corresponde al tiempo durante el cual no todos los procesadores están ocupados ejecutando tareas, mientras que la primera parte de la ecuación corresponde al tiempo durante el cual todos los procesadores están ocupados.

Por otra parte, considérese que la longitud del calendario óptimo está acotado inferiormente por los siguientes dos límites

$$C_{opt} \geq \frac{W_{tot}}{m}, \quad (28)$$

y

$$C_{opt} \geq \delta, \quad (29)$$

donde

$$W_{tot} = W + (m - a) \cdot \delta. \quad (30)$$

Por lo tanto, podemos decir que

$$\frac{W}{m} \leq C_{opt} - \frac{(m - a)}{m} \delta. \quad (31)$$

De acuerdo a la ecuación (27) observamos que la longitud del calendario de la estrategia está acotada por

$$C_{(a,1,0)\text{-Scheme}} \leq C_{opt} + \left(1 - \frac{(m - a)}{m}\right) \delta \leq C_{opt} \left(1 + \frac{a}{m}\right). \quad (32)$$

Ahora bien, para probar el valor de la cota, considérese un caso compuesto por  $(m - a)$  tareas con tiempo de ejecución  $m$  y  $(m \cdot a)$  tareas con tiempo de ejecución 1. Por lo tanto se tiene que la longitud del calendario es de

$$C_{(a,1,0)\text{-Scheme}} = m + a, \quad (33)$$

y la longitud del calendario óptimo es

$$C_{opt} = m. \quad (34)$$

Sustituyendo las ecuaciones (33) y (34) en la ecuación (25) obtenemos el resultado buscado.

Por último podemos expresar por medio del siguiente lema que:

**Lemma 3.** En cualquier estrategia ST2 el número de interrupciones está limitado por  $m - a$ .

A continuación se evalúa el caso cuando las tareas las ejecutan todos los procesadores, es decir, nos referimos al caso cuando  $b = m$ .

## VI.2 Caso $(a, m, 0)$ -Scheme

Para conocer el cociente competitivo como resultado de comparar el calendario generado por la estrategia y el calendario óptimo, se establece el siguiente teorema:

**Teorema 1.** El cociente competitivo se puede expresar de acuerdo al valor de  $m$ , tal como se muestra a continuación:

$$\rho_{(a,m,0)\text{-Scheme}}^* = 1 + \frac{a-1}{m}, \quad (35)$$

cuando

$$m \leq \frac{a\mu_m - 1}{\mu_m - 1}; \quad (36)$$

o bien puede ser

$$\mu_m \left(1 - \frac{a}{m}\right) + \frac{a}{m} \quad (37)$$

cuando,

$$m \geq \frac{a\mu_m - 1}{\mu_m - 1}. \quad (38)$$

*Demostración.* Siendo las etapas seguidas por el algoritmo las siguientes (ver figura 5):

- 1a.  $[0, t]$  Todos los procesadores están ocupados (estrategia *Graham*).
- 1b.  $[t, t']$  Al menos  $(m-a+1)$  procesadores están trabajando (estrategia *Graham*).
2.  $[t', t'']$  por lo menos  $a$  procesadores están ociosos; por lo tanto se aplica la estrategia *Gang*.

Considérese que una tarea que no finaliza antes de  $t'$  se ejecuta en la etapa 2. Por lo tanto, asumiendo que parte de esas tareas no se han procesado en la etapa 1a o en la etapa 1b, el trabajo pendiente de ejecutar es menor o igual a

$$\sum_{i=1}^k (p_i - \delta), \quad (39)$$

donde

$$k \leq m - a \quad (40)$$

y

$$\delta = t' - t. \quad (41)$$

Por lo tanto, el tiempo de terminación del calendario generado por  $(a, m, 0)$ -Scheme se determina de la siguiente manera (ver figura 5):

$$C_{(a,m,0)\text{-Scheme}} = \frac{W}{m} + \delta + \sum_{i=1}^k \frac{\mu_m}{m} (p_i - \delta), \quad (42)$$

suponiendo que la longitud del calendario óptimo está limitado inferiormente por las cotas

$$C_{opt} \geq p \quad (43)$$

y

$$C_{opt} \geq \frac{W + (m - a + 1)\delta + \sum_{i=1}^k (p_i - \delta)}{m}. \quad (44)$$

Por lo tanto, podemos decir que

$$\frac{W}{m} \leq C_{opt} - \frac{(m - a + 1)}{m} \delta - \frac{1}{m} \sum_{i=1}^k (p_i - \delta) \quad (45)$$

y que la longitud del calendario generado por la estrategia está limitado por

$$C_{(a,m,0)\text{-Scheme}} \leq C_{opt} - \frac{m-a+1}{m} \delta - \frac{1}{m} \sum_{i=1}^k (p_i - \delta) + \delta + \sum_{i=1}^k \frac{\mu_m}{m} (p_i - \delta). \quad (46)$$

Ahora bien, si consideramos que  $k$  tareas se ejecutan después de  $t'$ , podemos decir que la longitud del calendario generado por la estrategia está limitado por

$$C_{(a,m,0)\text{-Scheme}} \leq C_{opt} - \frac{m-a+1}{m} \delta + \frac{\mu_m - 1}{m} (p - \delta)k + \delta, \quad (47)$$

es decir,

$$C_{(a,m,0)\text{-Scheme}} \leq C_{opt} + \left( \frac{\alpha - 1}{m} - \frac{\mu_m - 1}{m} k \right) \delta + \frac{\mu_m - 1}{m} kp. \quad (48)$$

Partiendo de la consideración de que

$$B(\alpha) = \frac{\alpha - 1}{m} \quad (49)$$

y

$$A(k) = \frac{\mu_m - 1}{m} k, \quad (50)$$

podemos sustituir las ecuaciones (49) y (50) en la ecuación (48), y obtenemos la siguiente expresión:

$$J(\alpha, k) = (B(\alpha) - A(k))\delta + A(k)p. \quad (51)$$

En cuanto al tamaño que  $B(\alpha)$  y  $A(k)$  pueden tomar, se han definido tres casos:

**Caso 1.** Si  $A(k) = B(\alpha)$  entonces  $J(\alpha, k) \leq A(k)p$ , por lo tanto, la longitud del calendario generado por la estrategia puede estar limitada por

$$C_{(a,m,0)\text{-Scheme}} \leq C_{opt} + \frac{(\mu_m - 1)k}{m} p, \quad (52)$$

o bien, por

$$C_{(a,m,0)\text{-Scheme}} \leq C_{opt} + \frac{a-1}{m} p. \quad (53)$$

El cociente competitivo para este caso es

$$\rho_{(a,m,0)\text{-Scheme}}^* \leq 1 + \frac{(\mu_m - 1)k}{m} \leq 1 + \frac{(\mu_m - 1)(m - a)}{m} = \mu_m \left(1 - \frac{a}{m}\right) + \frac{a}{m}, \quad (54)$$

es decir,  $A(k) = B(a)$  cuando

$$\frac{a-1}{m} - \frac{(\mu_m - 1)(m - a)}{m} = 0. \quad (55)$$

Despejando el valor de  $m$  obtenemos que

$$m = \frac{a\mu_m - 1}{\mu_m - 1}. \quad (56)$$

**Caso 2.** Si  $A(k) \leq B(a)$  entonces  $J(a, k) \leq (B(a) - A(k))u + A(k)u \leq B(a)u$ .

Considérese que la longitud del calendario generado por la estrategia dada está limitada por

$$C_{(a,m,0)\text{-Scheme}} \leq C_{opt} + \frac{a-1}{m} p, \quad (57)$$

dando un cociente competitivo limitado superiormente, como se muestra en la siguiente expresión:

$$\rho_{(a,m,0)\text{-Scheme}}^* \leq 1 + \frac{a-1}{m}. \quad (58)$$

Por otro lado, se cumple  $A(k) \leq B(a)$  cuando

$$\frac{a-1}{m} - \frac{\mu_m - 1}{m} k \geq 0, \quad (59)$$

por lo tanto,

$$k \leq \frac{a-1}{\mu_m + 1}. \quad (60)$$

Para  $k = m - \alpha$  tenemos que  $m - \alpha \leq \frac{\alpha - 1}{\mu_m + 1}$  y  $m \leq \frac{\alpha \mu_m - 1}{\mu_m - 1}$ .

De acuerdo a lo anterior, podemos expresar que para un valor donde

$$m \leq \frac{\alpha \mu_m - 1}{\mu_m - 1}, \quad (61a)$$

o bien,

$$\alpha \geq m - \frac{m - 1}{\mu_m}, \quad (61b)$$

su cociente competitivo es

$$\rho_{(a,m,0)\text{-Scheme}}^* \leq 1 + \frac{\alpha - 1}{m}. \quad (62)$$

**Caso 3.** Si  $A(k) \geq B(\alpha)$  entonces  $J(\alpha, k) \leq A(k)p$ ; por lo tanto, la longitud del calendario generado por la estrategia está limitada por

$$C_{(a,m,0)\text{-Scheme}} \leq C_{opt} + \frac{(\mu_m - 1)k}{m} p. \quad (63)$$

Asúmase que el cociente competitivo para este caso es

$$\rho_{(a,m,0)\text{-Scheme}}^* \leq 1 + \frac{(\mu_m - 1)k}{m} \leq 1 + \frac{(\mu_m - 1)(m - \alpha)}{m} = \mu_m \left( 1 - \frac{\alpha}{m} \right) + \frac{\alpha}{m}. \quad (64)$$

Además debemos de considerar que  $A(k) \geq B(\alpha)$  cuando

$$\frac{\alpha - 1}{m} - \frac{\mu_m - 1}{m} k \leq 0, \quad (65)$$

por lo tanto,

$$k \geq \frac{\alpha - 1}{\mu_m + 1}. \quad (66)$$

Si consideramos que  $k \leq m - \alpha$  sabemos que

$$m - a \geq \frac{a - 1}{\mu_m + 1} \quad (67)$$

y

$$m \geq \frac{a\mu_m - 1}{\mu_m - 1}, \text{ si } (m > a). \quad (68)$$

De acuerdo a lo anterior, podemos expresar que para

$$m \geq \frac{a\mu_m - 1}{\mu_m - 1} \quad (69a)$$

o

$$a \leq m - \frac{m - 1}{\mu_m}. \quad (69b)$$

su cociente competitivo es

$$\rho_{(a,m,0)\text{-Scheme}}^* \leq \mu_m \left( 1 - \frac{a}{m} \right) + \frac{a}{m}. \quad (70)$$

A continuación se muestran los cocientes de desempeño para distintos valores que puede tomar  $a$  (ver tabla III).

Nótese que cuando  $a = 1$ , el cociente de desempeño de la estrategia  $(1, m, 0)$ -Scheme es

$$\rho_{(1,m,0)\text{-Scheme}}^* \leq \mu_m \left( 1 - \frac{1}{m} \right) + \frac{1}{m}. \quad (71)$$

Obsérvese que esta cota es igual a la cota que presenta la estrategia *Scheme* (ver tabla III).

Ahora bien, se dice que los trabajos son paralelizados con la estrategia *Gang* sin sobre costo cuando  $\mu_m = 1$ . Es decir, se tiene un calendario óptimo sin tiempos ociosos, por lo tanto, podemos decir que su cociente de desempeño es

$$\rho_{(\alpha,m,0)\text{-Scheme}}^* = 1. \quad (72)$$

Por último, se hace referencia al caso cuando  $\alpha = m$ , en el cual únicamente se aplica la estrategia *Graham*. Por lo tanto podemos expresar que

$$\rho_{(a,m,0)\text{-Scheme}}^* \leq 1 + \frac{a-1}{m} = 2 - \frac{1}{m}. \quad (73)$$

La siguiente tabla muestra el cociente de desempeño para los diferentes valores de  $(a,b,0)$ -Scheme:

Tabla III. Cociente de desempeño de  $(a,b,0)$ -Scheme.

$a$	$b$	Desempeño	Estrategia
0	$m$	$\mu_m$	$(0,m,0)$ -Scheme-Gang
1	$m$	$\mu_m \left(1 - \frac{1}{m}\right) + \frac{1}{m}$	$(0,m,0)$ -Scheme-Scheme
$1 \leq a \leq m$	$m$	$\mu_m \left(1 - \frac{a}{m}\right) + \frac{a}{m}$ , para $m \geq \frac{a\mu_m - 1}{\mu_m - 1}$ $1 + \frac{a-1}{m}$ , para $m \leq \frac{a\mu_m - 1}{\mu_m - 1}$	$(0,m,0)$ -Scheme
$m$	1	$2 - \frac{1}{m}$	$(m,1,0)$ -Scheme-Graham

De acuerdo a los diferentes valores que  $(a,b,c)$  pueden tomar, en la tabla IV se muestran las estrategias que se deberán aplicar dentro de la estrategia  $(a,b,c)$ -Scheme.

Tabla IV. Clasificación de  $(a,b,c)$ -Scheme de acuerdo al valor de  $(a,b,c)$ .

$(a,b,c)$ -Scheme	Estrategia
$(0,m,0)$ Scheme	Gang
$(1,m,0)$ Scheme	Scheme
$(m,1,0)$ Scheme	Graham
$(0,(1\dots k))$ Scheme	El conjunto de tareas se divide en $k$ subconjuntos $T^1, T^2, \dots, T^k$ . Cada tarea en $T^s$ requiere exactamente $s$ procesadores. Width- $s$ tareas o $T^s$ tareas (Blazewicz <i>et al.</i> , 2000)

Las figuras 7 y 8 muestran el comportamiento de desempeño de  $(a,b,c)$ -Scheme cuando se varía el número de procesadores y el valor de  $a$ .

En la figura 7 podemos observar que teóricamente el balance mínimo del cociente de desempeño al variar el número de procesadores se presenta cuando  $m$  tiene un valor en el rango de 95 a 110 procesadores.

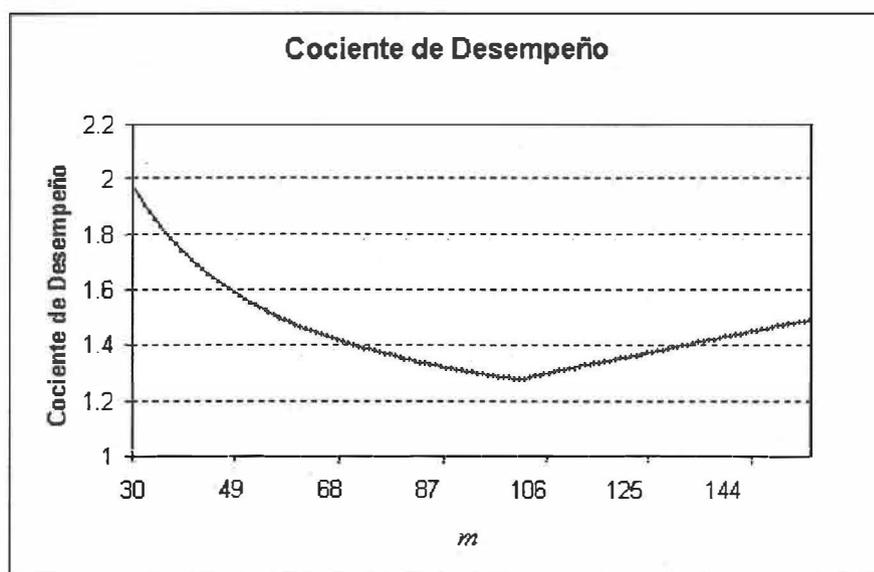


Figura 7.  $\rho_{(a,m,0)\text{-Scheme}}$  al variar el número de procesadores,  $a=30$ ,  $\mu_m=0.0038m+0.9975$ .

Ahora bien, podemos observar que teóricamente el balance mínimo del cociente de desempeño al variar el valor del parámetro  $\alpha$ , se encuentre cuando se tiene un valor entre 80 a 90 (ver figura 8).

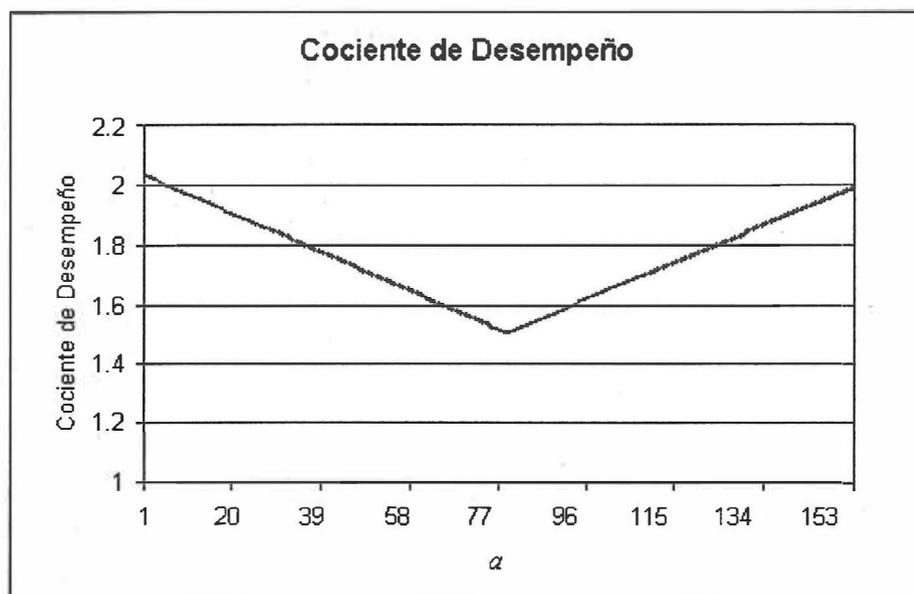


Figura 8.  $\rho_{(\alpha,m,0)\text{-Scheme}}$  al variar el valor de  $\alpha$ ,  $m=160$ ;  $\mu=0.0038m+0.9975$ .

## VII Sistema experimental

Para llevar a cabo la experimentación de las estrategias analizadas en el presente trabajo se desarrolló un sistema experimental, el cual se divide en dos grandes etapas. La primera etapa corresponde a la generación del conjunto de tareas y la segunda al procesamiento del conjunto de tareas. Ambas etapas se ejecutan por medio del programa llamado *Scheduling*. Este programa está diseñado en lenguaje Delphi.

*Scheduling* forma parte de un proyecto desarrollado anteriormente, el cual se diseñó en un inicio para simular estrategias de calendarización de lista, en donde a las tareas se asigna una prioridad de acuerdo a criterios tales como: el número de sucesores o predecesores de la tarea, el menor o mayor tiempo de procesamiento de la tarea, por mencionar algunos. Además, una tarea sólo se puede asignar a un procesador a la vez. Otra característica de *Scheduling* es el hecho de que estaba diseñado únicamente para trabajar con conjuntos de tareas con precedencias. Llamaremos a todo esto *primer módulo*. Actualmente *Scheduling* está formado por dos tipos de módulos. En el *segundo módulo* están incluidas todas las estrategias que se han venido analizando a lo largo de este documento, las cuales permiten ejecutar una tarea en más de un procesador a la vez. Estas estrategias utilizan conjuntos de tareas independientes y requieren conocer el valor de ciertos parámetros al inicio del programa. Para indicar al sistema que se desea ejecutar esta parte de la implementación, llamaremos a este segundo módulo *kmalbl*.

Por otra parte, se tiene una serie de argumentos que se proporcionan al inicio del programa, los cuales representan los argumentos requeridos tanto por la etapa de generación, como por la etapa de procesamiento del conjunto de tareas.

Para dar inicio a cualquiera de las dos etapas, ya sea la encargada de generar, o bien la encargada de procesar el conjunto de tareas, es necesario ejecutar el programa *Scheduling*.

A continuación se describen detalladamente las etapas que conforman el proceso de implementación.

### ***VII.1 Generación del conjunto de tareas***

Para llevar a cabo esta etapa se requiere que los siguientes argumentos se proporcionen al dar inicio la ejecución del programa:

*total\_task*, *total\_experim*, *total\_tree*, *ktime*, *kgcreate*, *mymodel*, *kgtype*, *kbeta*.

- *total\_task*, es el número de tareas que contiene un conjunto de tareas.
- *total\_experim*, es el número de experimentos a realizarse.
- *total\_tree*, se refiere a la cantidad de conjuntos de tareas a generarse.
- *ktime*, se refiere a los tiempos de procesamiento de las tareas; estos pueden ser unitarios o no unitarios. En caso de ser no unitarios, los tiempos asignados pueden tomar valores aleatorios con distribución Gaussiana con  $\mu_x = 70$  y  $\sigma_x = 28$ .
- *kgcreate*, indica si se va a crear un nuevo conjunto de tareas o no. Cuando su valor es positivo se debe generar un nuevo conjunto. En el caso de que su valor sea negativo, el conjunto creado previamente se puede utilizar cuantas veces se desee por las diferentes estrategias manejadas en el sistema. Al iniciar la ejecución del sistema se le asigna el valor de verdadero a la opción para generar un conjunto de tareas.
- *mymodel*, indica el módulo de estrategias a utilizarse en este programa. Como se mencionó anteriormente, la implementación realizada forma parte de un sistema mayor,

por lo que únicamente nos concentraremos en la parte que contiene las estrategias mencionadas en este trabajo, llamada *kmalbl*.

- *kgrtype*, indica el tipo de conjunto de tareas a utilizarse. En nuestro análisis, únicamente estudiaremos conjuntos de tareas independientes.
- *kbeta*, se refiere al tipo de *b* o número de procesadores que requiere cada tarea al momento de paralelizar la tarea de acuerdo a la estrategia a aplicarse. Los tipos *b* que se pueden elegir son:
  - o *malbl*, para el caso de la estrategia *maleable*, es decir, depende del número de procesadores libres.
  - o *single*, para la estrategia *Graham*, es decir, una tarea requiere únicamente un procesador.
  - o *total*, para la estrategia *Gang*, es decir, todos los procesadores se asignan a cada tarea.

Durante esta etapa se generan *total\_tree* conjunto de tareas de tipo *kgrtype*, con determinado número de tareas con tiempos de procesamiento no unitario. El conjunto de tareas generado se almacena temporalmente en el archivo de nombre *aaa.pas* dentro del subdirectorio *Temp*. Una vez que se indica que se va a almacenar definitivamente los datos de ese conjunto de tareas, dicho archivo se almacena en el subdirectorio *Result*.

El mecanismo general para la generación del conjunto de tareas es el siguiente:

1. Crear archivo temporal *aaa.pas* en directorio *Temp*.
2. Generar el conjunto de tareas.
3. Asignar el tiempo de procesamiento a las tareas del conjunto, siguiendo una distribución Gaussiana.

4. Asignar nivel a cada tarea del conjunto de tareas, basándose en el tiempo de procesamiento requerido por la tarea.
5. Asignar conivel a cada tarea del conjunto. Ya que se trabaja con un conjunto de tareas independientes, el conivel de las tareas es igual a su tiempo de procesamiento.
6. Asignar a cada tarea del conjunto de tareas su tipo de  $b$ , el cual puede ser *malbl*, *single* o *total*. Aquí únicamente se le asigna el tipo de  $b$  de la tarea; es durante el procesamiento cuando se le asigna un valor de acuerdo al tipo de  $b$  asignado a la tarea.
7. Si *kgcreate* es verdadera:
  - a. Guardar los datos del conjunto de tarea creado en el archivo *aaa.pas* dentro del directorio *Temp*.
  - b. De lo contrario ir al paso 1.

El diagrama de flujo de los pasos mencionados anteriormente se muestra en la figura 9.

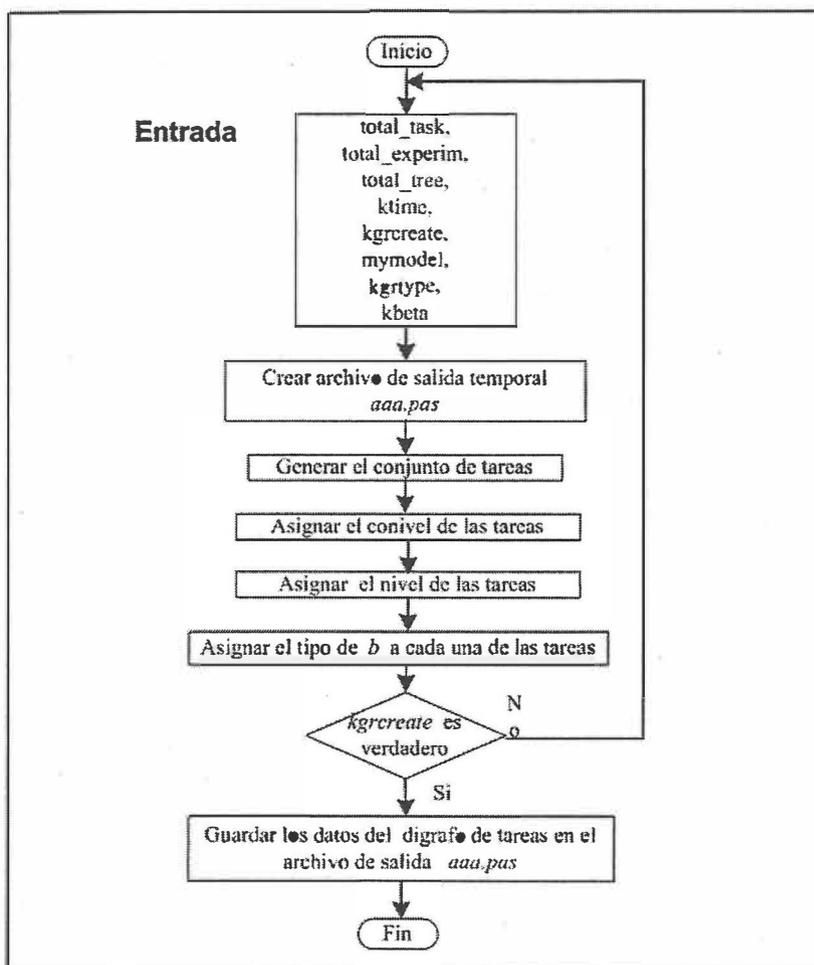


Figura 9. Representación en diagrama de flujo del mecanismo para la generación del conjunto de tareas.

A continuación, se describe el formato que se utiliza para almacenar las características de las tareas en el archivo *aaa.pas*:

- Número de tarea.
- Capa a la que pertenece la tarea. Como se trata de tareas independientes todas las tareas del conjunto se encuentran en la capa 1.
- Conivel de la tarea.

- Nivel de la tarea.
- Tiempo de procesamiento de la tarea.
- Tipo de  $b$  que se asignó a la tarea.
- Número de sucesores de la tarea. Como las tareas manejadas son independientes su valor es igual a 0.
- Número de predecesores de la tarea. Los conjunto de tareas generados utilizan tareas independientes, por lo tanto las tareas no tienen predecesores.

Es importante aclarar que la información contenida en este archivo está distribuida de manera que los datos de cada tarea se guarden en un solo renglón. Por lo tanto, para una mejor claridad durante nuestra explicación se optó por mostrar por separado cada uno de los datos contenidos en el archivo. La figura 10 muestra la información contenida en el archivo *aaa.pas* al generar un conjunto con 10 tareas, con tiempos no unitarios.

1	1	113	0	113	single	[0]=	[0]=
2	1	99	0	99	single	[0]=	[0]=
3	1	125	0	125	single	[0]=	[0]=
4	1	96	0	96	single	[0]=	[0]=
5	1	48	0	48	single	[0]=	[0]=
6	1	47	0	47	single	[0]=	[0]=
7	1	30	0	74	single	[0]=	[0]=
8	1	29	0	84	single	[0]=	[0]=
9	1	26	0	72	single	[0]=	[0]=
10	1	12	0	156	single	[0]=	[0]=

Figura 10. Información almacenada en el archivo *aaa.pas*.

## VII.2 Descripción de la implementación del calendarizador

Esta etapa requiere que los argumentos que se enlistan a continuación se proporcionen al inicio del programa:

*kproc, kmmu, kmalfa, malbnum, kgrsave*

- *kproc*, es el número de procesadores en el sistema. Éste puede ser un valor fijo o bien ser un número variable de procesadores. Al introducirse el valor de 0 (cero), el sistema varía los valores de 1 hasta  $m$ . El valor máximo de procesadores que se pueden utilizar es de 160; esta condición se explica a detalle en la sección 8.2.
- *kmmu*, corresponde al factor de ineficiencia o tipo de  $\mu$  que debe aplicarse a cada una de las tareas, si éstas se paralelizan. Este factor puede ser cóncavo, convexo o lineal.
- *kmalfa*, es el número de procesadores ociosos que debe haber para hacer cambio de estrategia. Su valor puede ser un número en específico o bien se puede escribir el valor 0 (cero), para indicar que se realice todas las combinaciones para  $a$  de 1 hasta  $m - 1$ .
- *malbnum*, indica el tipo de estrategia a utilizar en el procesamiento del conjunto de tareas. Éstas se pueden calendarizar aplicando las estrategias *Graham*, *LPT*, o bien distribuyendo los procesadores de acuerdo a *Gang* o *(a,b,c)-Scheme*.
- *kgrsave*, especifica si los datos generados por las estrategias se van a almacenar. Su valor es verdadero en el caso de que se desee grabar los datos, o bien es falso si no se desea grabar los datos generados por las estrategias.

En forma general, podemos describir la implementación del calendarizador de la siguiente manera:

1. Lectura de datos de entrada:

- a) Los datos generados durante la creación del conjunto de tareas (paso descrito en la sección anterior).
  - b) Del número de procesadores en el sistema, del valor de  $a$  y del tipo de  $\mu$  (estos datos se introducen al momento de ejecutar el programa *Scheduling*).
2. Procesamiento del conjunto de tareas de acuerdo a las estrategias seleccionadas.
  3. Creación de archivos temporales con extensión “.yyy” dentro del directorio *Temp*, para almacenar en formato binario los resultados generados por las estrategias utilizadas.
  4. Si *kgrsave* es verdadera, entonces, almacenar la información contenida en los archivos con extensión “.yyy” dentro de cuatro archivos diferentes con extensión “.txt”, en los cuales la información está distribuida de la siguiente manera:
    - a) Creación del archivo con nombre *C\_opt.txt* para almacenar los resultados óptimos para cada una de las estrategias utilizadas.
    - b) Creación del archivo *c\_Malbl.txt*, que contiene el tiempo de procesamiento generado por cada una de las estrategias.
    - c) Creación del archivo *total.txt*, para almacenar la cantidad de tiempos ociosos que se tuvieron durante todo el procesamiento de cada una de las estrategias.
    - d) Creación del archivo con nombre *r\_Malbl.txt*, donde se almacena el cociente competitivo originado por cada una de las estrategias de acuerdo a la información almacenada en *C\_opt.txt* y en *c\_Malbl.txt*.

En caso contrario, si se desea crear un conjunto de tareas, se debe ir al paso 1, de no ser así, al paso 2.

En la figura 11 se muestra el diagrama de flujo de la etapa de procesamiento del calendarizador.

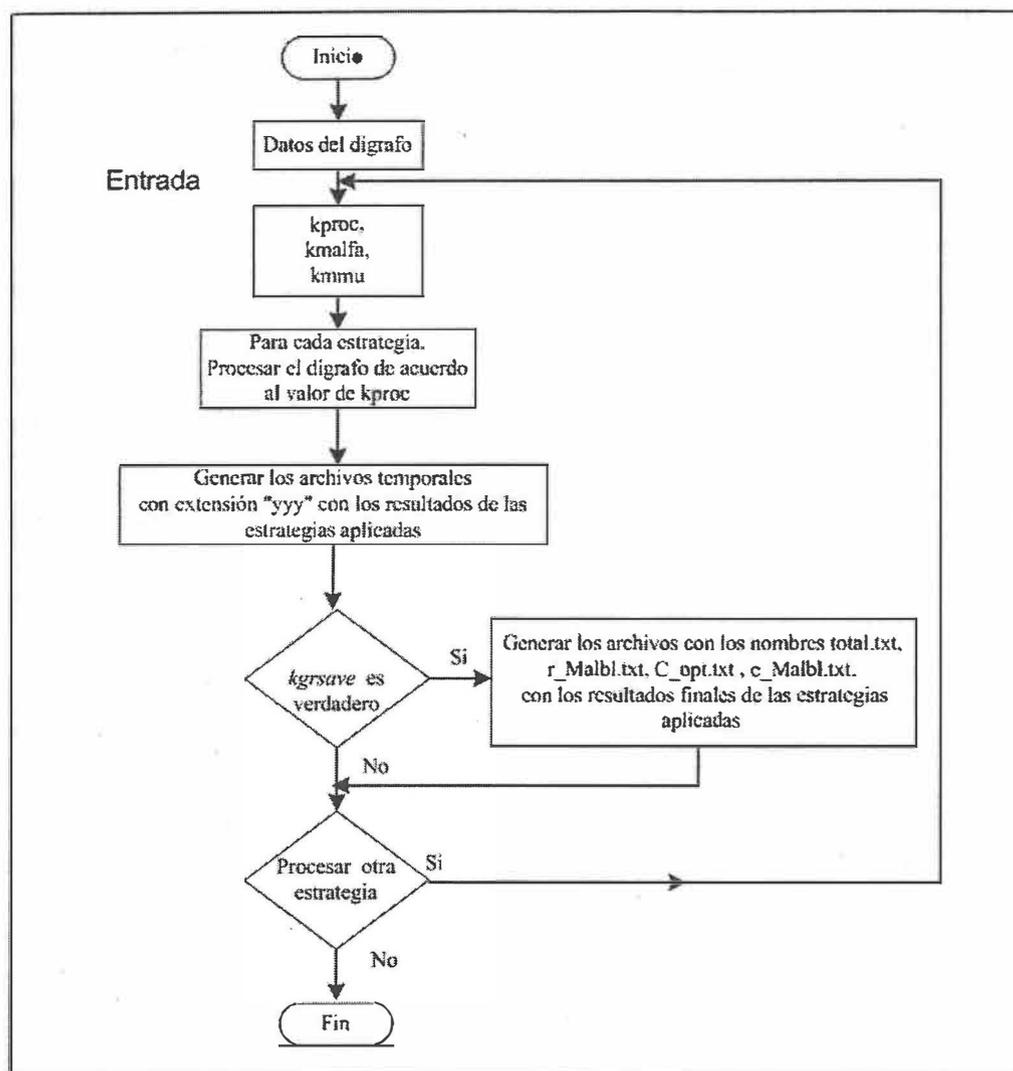


Figura 11. Diagrama de flujo de la etapa de implementación del calendarizador.

A continuación se describe a detalle cada una de las partes que conforman la etapa de implementación del calendarizador.

### VII.2.1 Lectura de datos de entrada

Se lleva a cabo la lectura de la información contenida en el archivo *aaa.txt* generado durante la primera etapa de la implementación, así como la de aquella información proporcionada al inicio del programa. Apoyándose en lo anterior, se da inicio a la etapa de implementación del calendarizador. A partir de este momento, el sistema conoce todos los parámetros necesarios para llevar a cabo la calendarización de las tareas.

### VII.2.2 Procesamiento del conjunto de tareas

Mediante un ciclo controlado por la estrategia a aplicarse, se hace un llamado al procedimiento *ForMalblProc*, que se encarga de ejecutar al procedimiento *ForMalblHeu*.

De acuerdo al valor asignado a *kproc* al inicio del programa, el llamado a *ForMalblHeu* puede repetirse varias veces. Si el valor asignado a *kproc* es un número en específico, este procedimiento se ejecuta una sola vez. Por otra parte, si se asignó a *kproc* el valor de cero, este procedimiento se ejecutará en varias ocasiones mediante un ciclo controlado por el número máximo de procesadores permitidos cuyo valor es de 160 procesadores. A partir de este momento el valor de *kproc* se asigna a *pnumber*, el cual representa el número total de procesadores en el sistema. Por cada valor que se asigne a *pnumber*, *ForMalHeu* se encarga de llamar a otros procedimientos dependiendo de la estrategia que se desee aplicar al conjunto de tareas. Cuando la estrategia a aplicarse es *Graham* o *LPT*, se llama al procedimiento *MalblRun*. Si desde el inicio de la calendarización se debe seguir la estrategia *Gang*, *ForMalHeu* se encarga de llamar a *ForMalblMu*. Por último, si se trata de la estrategia *(a,b,c)-Scheme*, se llama al procedimiento *ForMalblAlfa*

A continuación se describe el funcionamiento de los procedimientos *ForMalblMu*, *ForMalblAlfa* y *MalblRun* mencionados anteriormente:

- *ForMalblMu*, indica el tipo de factor de ineficiencia ( $\mu$ ), que habrá de aplicarse a las tareas que se asignen a más de un procesador. Este procedimiento se llama únicamente cuando se trate de la estrategia *(a,b,c)-Scheme* o de la estrategia *Gang*. Los valores de  $\mu$  pueden ser: cóncavo, convexo o lineal. Para cada caso se llama a *MalblRun*.
- *ForMalblAlfa*, se encarga de asignar el valor de *kalfa*, que representa el número de procesadores ociosos en el sistema necesarios para pasar a la etapa 2 de la estrategia *(a,b,c)-Scheme* conocida como ST2. El valor de *kmalfa* se asigna a *kalfa*, en el caso de que su valor sea cero, *kalfa* toma todos los valores de 1 hasta *pnumber* - 1. Para cada valor que tome *kalfa*, el procedimiento *ForMalblAlfa* hace un llamado a *ForMalblMu*.
- *MalblRun*, se encarga de llamar al procedimiento *Modeling* que es el responsable de realizar la asignación de las tareas a los procesadores de acuerdo al tipo de estrategia y al valor de *b* de las tareas a calendarizar. Al finalizar *Modeling*, se conoce el tiempo que tardó la estrategia en procesar todas las tareas del conjunto.

La figura 12 ilustra gráficamente como fluye la información durante los pasos que se explicaron con anterioridad.

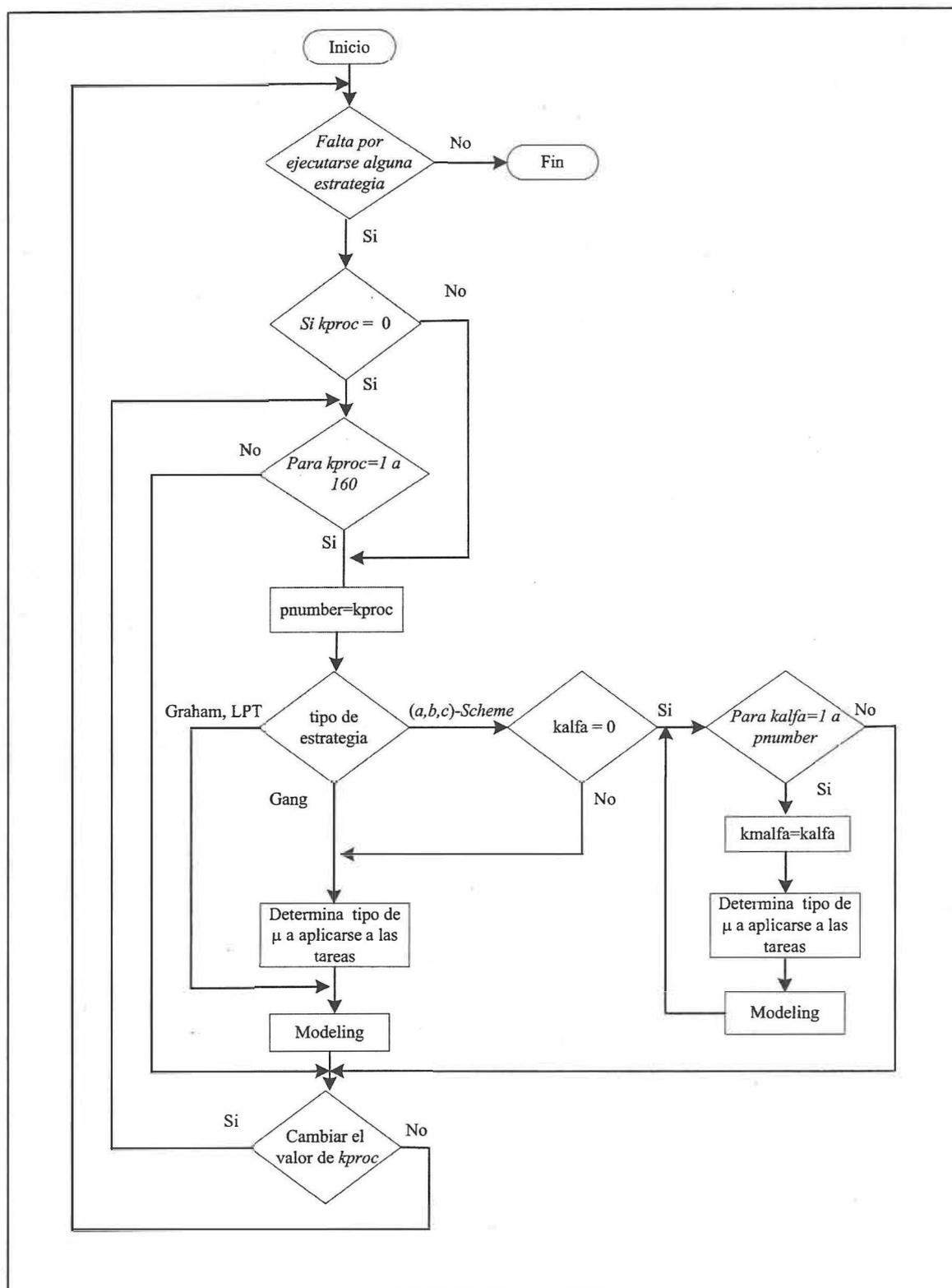


Figura 12. Diagrama de flujo general de la etapa de procesamiento del conjunto de tareas.

En la siguiente sección se describirá más detalladamente el procedimiento llamado *Modeling*.

### VII.2.2.1 Procedimiento *Modeling*

Este procedimiento es el encargado de modelar las estrategias de calendarización auxiliándose con las funciones *MallCondition* y *MallAssign*. La primera se encarga de verificar si se debe interrumpir la ejecución de las tareas para aplicar una nueva estrategia, mientras que por otra parte, *MallAssign* es la responsable de asignar las tareas a los procesadores. Se ilustra en la figura 13 el diagrama de flujo del procedimiento *Modeling*.

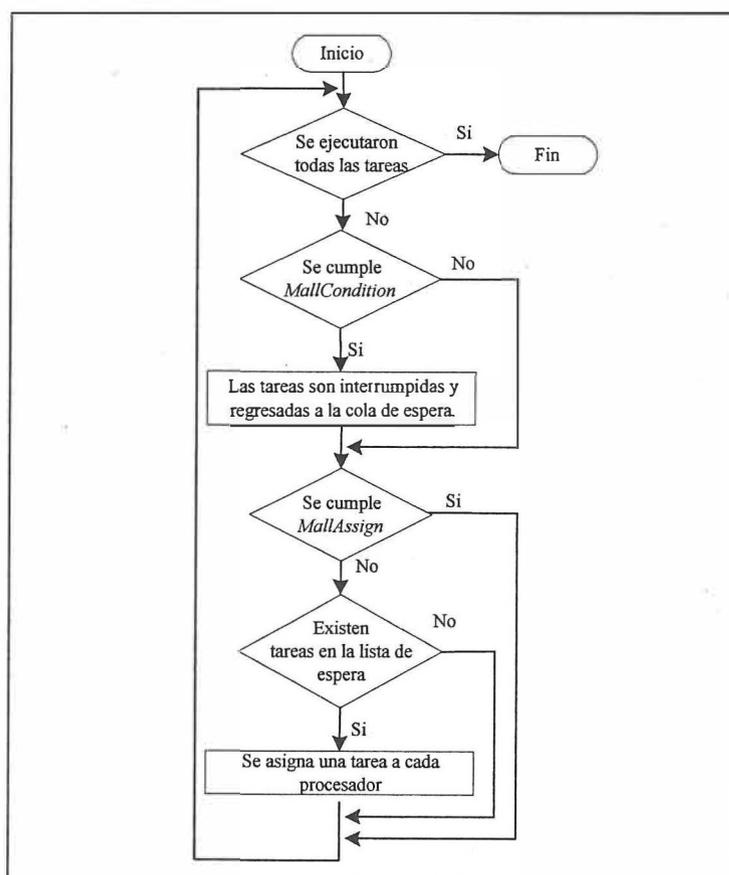


Figura 13. Diagrama de flujo del procedimiento *Modeling*.

### VII.2.2.2 Procedimiento *MallCondition*

*MallCondition* es una función lógica que recibe como parámetro de entrada el número de tareas que no han finalizado su ejecución, llamado  $n$ , y cuya salida indica si la ejecución de las tareas se debe o no interrumpir.

*ChgeStrategy* se encarga de verificar si la estrategia a aplicar permite o no que las tareas se asignen a más de un procesador. Su valor es positivo cuando una tarea puede requerir más de un procesador, de lo contrario su valor es negativo. Además, *ChgeStrategy* únicamente se puede cumplir durante la ejecución de la estrategia *Gang*, o bien de la estrategia  $(a,b,c)$ -*Scheme*, de acuerdo a lo siguiente:

- En el caso de la estrategia *Gang*, se cumple debido a que todas las tareas se ejecutan desde un inicio en más de un procesador.
- A diferencia del caso anterior, cuando se trata de la estrategia  $(a,b,c)$ -*Scheme*, la condición se cumple una vez que se tiene en el sistema a los  $b$  procesadores ociosos que permiten pasar a la segunda etapa de esta estrategia.

El diagrama de flujo de la función *MallCondition* se ilustra en la figura 14.

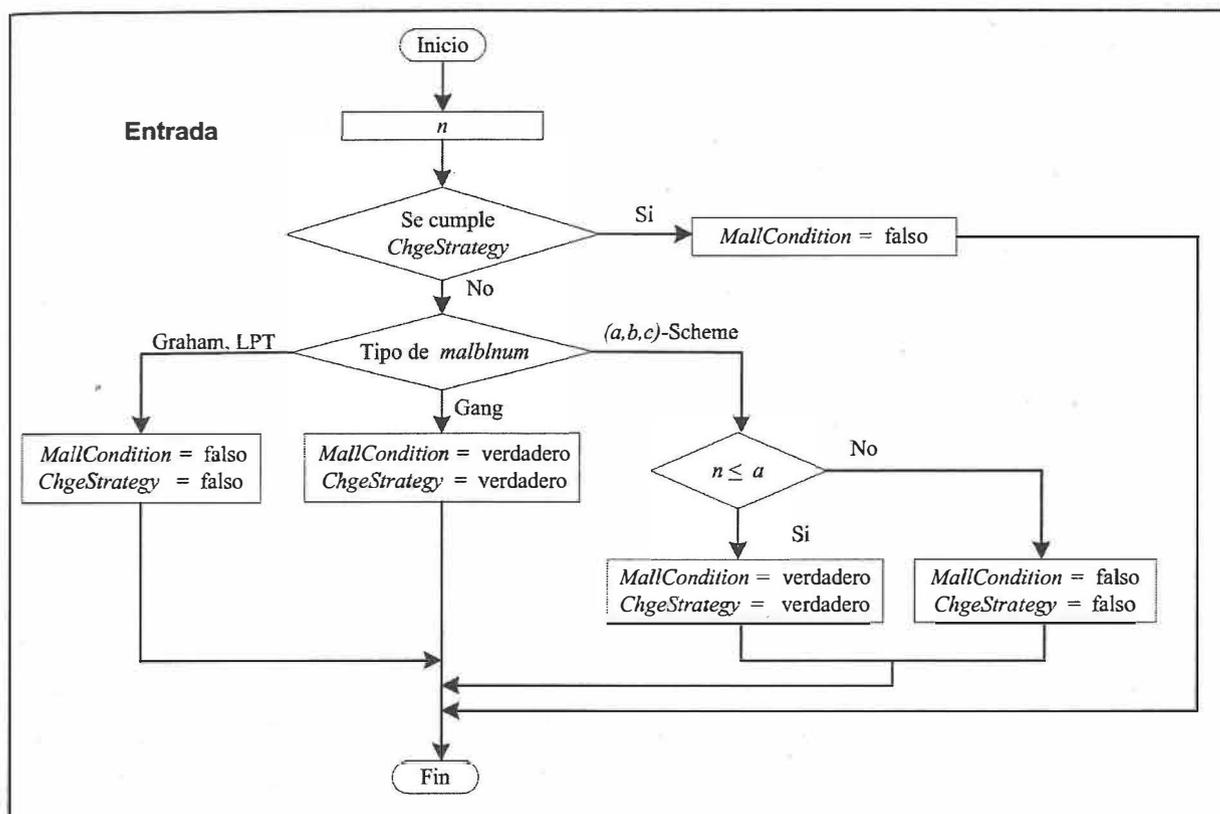


Figura 14. Representación en diagrama de flujo de la función *MallCondition*.

### VII.2.2.3 Procedimiento *MallAssign*

Para asignar las tareas a los procesadores se sigue el procedimiento descrito a continuación:

- Verificar si se hizo cambio de estrategia en algún momento durante la calendarización.
- Si se cumple la condición, se indica al sistema que no se puede volver a cambiar la estrategia. De no cumplirse la condición, existen dos opciones para la asignación de las tareas a los procesadores dependiendo de la estrategia que se esté ejecutando:
- a. Ejecutar la asignación de acuerdo a la estrategia *Gang*.
  - b. Ejecutar la asignación de acuerdo a la estrategia *(a,b,c)-Scheme*.

Se ilustra en la figura 15 la descripción en forma de diagrama de flujo del procedimiento *MallAssign*.

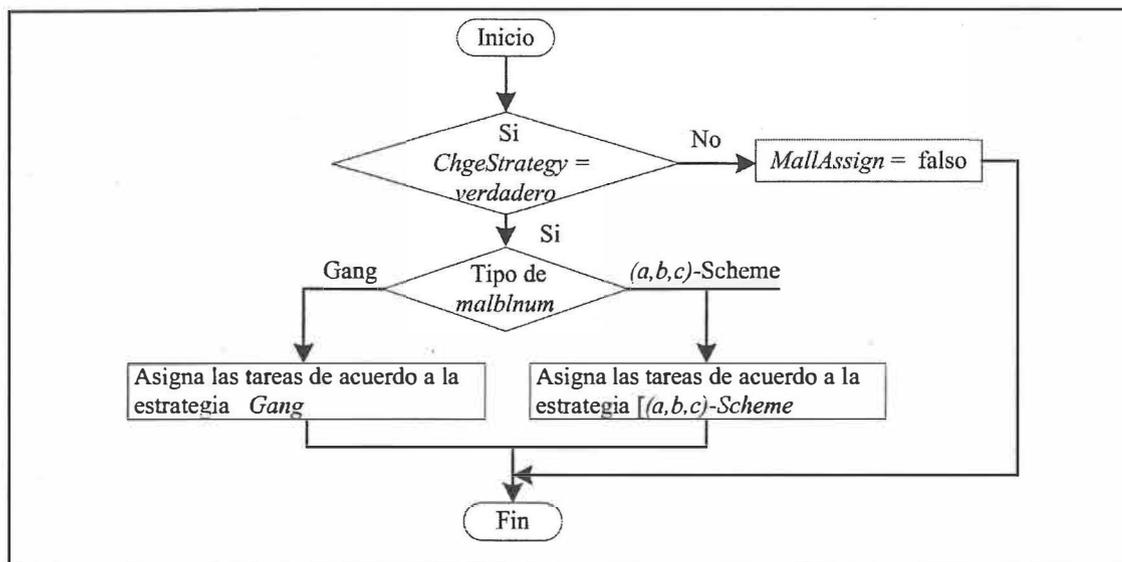


Figura 15. Representación gráfica de *MallAssign*.

Si la asignación de las tareas se realiza de acuerdo a la estrategia *Gang* desde el inicio de la calendarización, la tarea se asigna al número total de procesadores (ver figura 16).

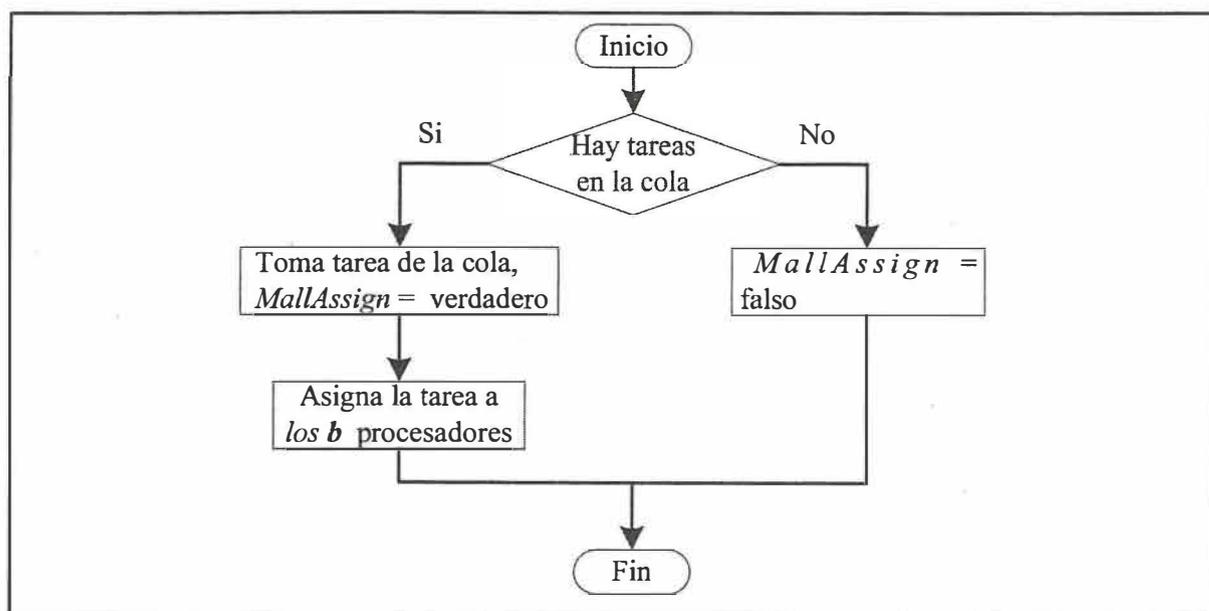


Figura 16. Asignación de tareas de acuerdo a la estrategia *Gang*.

Ahora bien, si la estrategia a aplicarse es  $(a,b,c)$ - *Scheme*, la asignación de la tarea depende del tipo de  $b$  asignado a esa tarea (ver figura 17). Podemos decir que a partir de este momento se inicia la etapa 2 de la estrategia  $(a,b,c)$ - *Scheme*. Los tipos de  $b$  pueden ser: 1, la tarea se asigna a un solo procesador;  $m$ , la tarea se asigna a todos los procesadores; *maleable*, la tarea se asigna a todos los procesadores libres. En el caso de que  $b$  sea igual a *maleable*, únicamente se interrumpe la ejecución de una tarea, mientras que el resto de las tareas las continúa ejecutando un solo procesador.

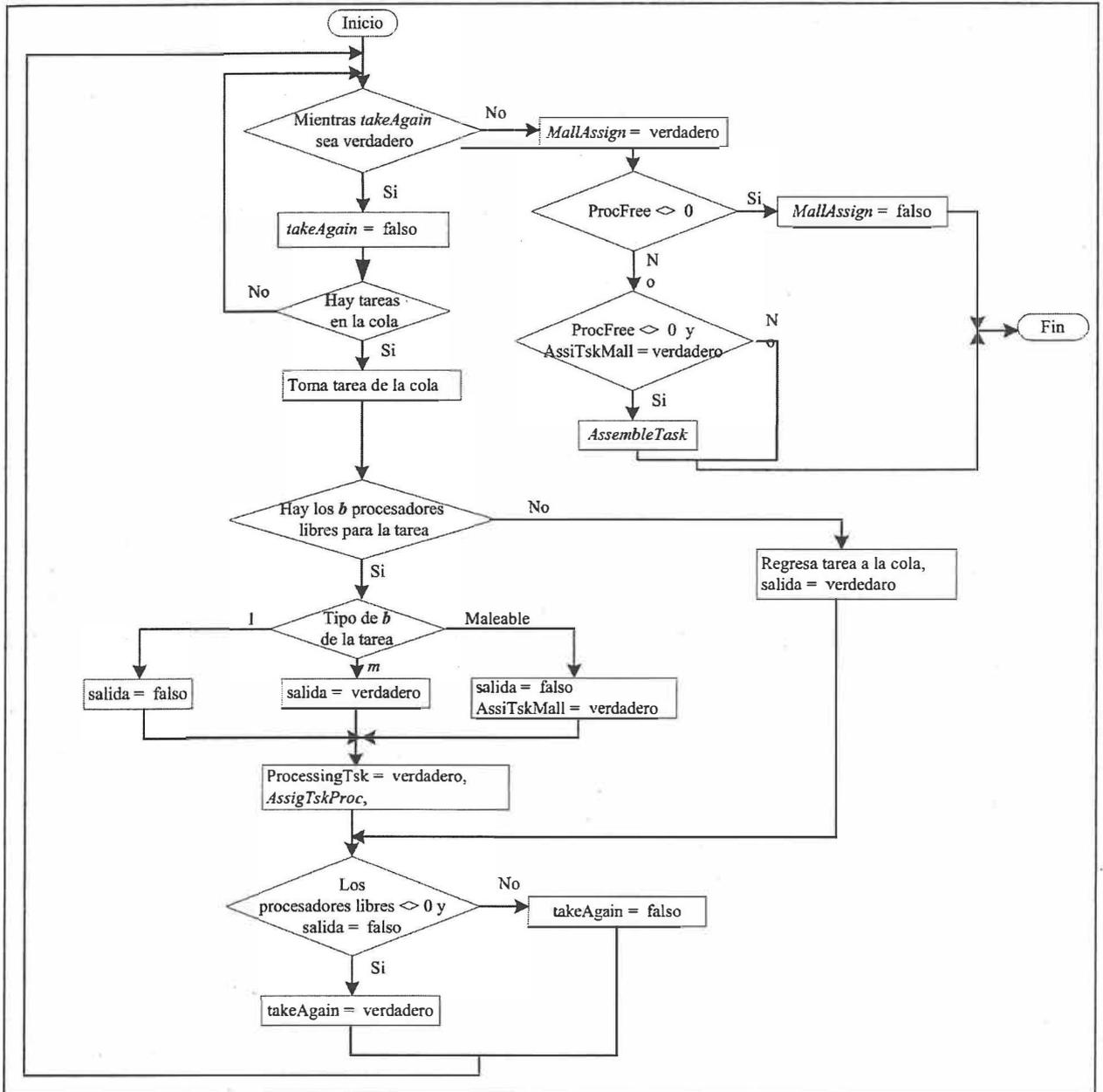


Figura 17. Asignación de tareas de acuerdo a la estrategia  $(a,b,c)$ -Scheme.

- *takeAgain*, indica si es necesario volver a tomar otra tarea de la cola. Los valores que puede tomar son: *verdadero*, tomar otra tarea de la cola de tareas, *falso*, no tomar por el momento otra tarea de la cola.

- *salida*, representa si existen o no procesadores libres después de haber asignado la tarea a los  $b$  procesadores necesarios para su ejecución. Es *verdadero* si algún procesador es libre, de lo contrario toma el valor de *falso*.
- *ProcFree*, es el número total de procesadores libres después de asignar una tarea.
- *AssembleTask*, se utiliza sólo cuando la tarea tiene una  $b$  del tipo *maleable*. Cada vez que un procesador es libre, *AssembleTask* se encarga de paralelizar la tarea en los procesadores libres. No se toma una nueva tarea hasta que la tarea paralelizada termine de ejecutarse.

### ***VII.3 Almacenamiento de los resultados***

Los resultados obtenidos por haber aplicado las diferentes estrategias para calendarizar las tareas del conjunto se almacenan de acuerdo a ciertos criterios en un conjunto de archivos con extensión “.txt”, debido a que una sola estrategia puede llegar a generar una gran cantidad de información. Los datos obtenidos dependen del valor de  $a$ ,  $b$  y del número total de procesadores en el sistema, ya que estos tres parámetros determinan la serie de combinaciones que habrá de realizar cada estrategia. Se utiliza un total de cuatro archivos para almacenar los resultados obtenidos por las estrategias aplicadas. Cada uno de los renglones de estos archivos está formado por seis columnas, las cuales corresponden a: el nombre de la estrategia aplicada, el número total de procesadores en el sistema, el valor de  $a$ , el factor de ineficiencia aplicado (cóncavo, convexo, lineal), así como el tipo de  $b$  asignado a las tareas del conjunto. De acuerdo al nombre del archivo generado con extensión “.txt”, la información de la última columna corresponde a:

- *En el archivo total.txt*, se refiere a la suma total de tiempos ociosos de todos los procesadores durante la ejecución de la estrategia de calendarización.
- *En el archivo r\_Malbl.txt*, se guarda el cociente competitivo de cada uno de las estrategias, como resultado de la división del tiempo óptimo almacenado en *C\_opt.txt* entre el tiempo de ejecución de la estrategia; esta última información proporcionada por el archivo *r\_Malbl.txt*.
- *En el archivo C\_opt.tx*, se registra el mejor tiempo de procesamiento posible para cada una de las estrategias. Es decir, la cota inferior para cada estrategia de acuerdo al número total de procesadores, así como de  $a$  y  $b$ .
- *En el archivo c\_Malbl.txt*, se hace referencia a los tiempos de procesamiento generados por cada una de las estrategias.

Por otra parte, la información del conjunto de tareas se almacena en el archivo *aaa.pas*. Los datos almacenados son: número de tarea, capa de la tarea, nivel, conivel, tiempo de procesamiento.

## VIII Resultados Experimentales

Con el objetivo de analizar el comportamiento de la estrategia adaptativa  $(a,b,c)$ -Scheme se llevan a cabo experimentos con un gran número de conjuntos de tareas, con tiempos de procesamiento no unitarios. El procesamiento de las tareas se simula con procesadores abstractos.

Los resultados arrojados por las estrategias presentadas en este trabajo se comparan con el resultado obtenido con la estrategia *LPT* (ver sección 3.2.1), la cual es determinística ya que se conoce por adelantado toda la información. Por otro lado, se calculó la desviación estándar del comportamiento promedio de las estrategias analizadas.

Antes de mostrar los resultados experimentales, es necesario detallar el criterio que se tomó para la asignación de los tiempos de procesamiento de las tareas al momento de crear los conjuntos de tareas. Es importante mencionar que han sido definidas tres diferentes funciones como factor de ineficiencia, las cuales serán utilizadas durante los experimentos. Dichas funciones serán detalladas más adelante.

### VIII.1 *Tiempo de procesamiento de una tarea (p)*

En la calendarización dinámica no se puede predecir de antemano la ocurrencia de las tareas en el sistema, así como también se desconoce su tiempo de ejecución, ya que dichas tareas ejecutan un número diferente de operaciones durante su procesamiento. Esto es el caso de tareas reales, pero en nuestro caso para llevar a cabo la simulación es necesario asignarle a cada tarea un tiempo de ejecución, el cual se genera en forma aleatoria. Para determinar el tipo de distribución que se aplicará al generar dichos valores aleatorios, se

tomó como base el comportamiento de la máquina paralela que se tiene en CICESE, llamada cicese2000. Esta computadora paralela cuenta con 10 procesadores entre los cuales se distribuye la carga de trabajo.

Los datos recabados sobre el uso de CPU de esta computadora paralela corresponden a un periodo de 6 meses, el cual está comprendido de abril del 2002 a septiembre del 2002.

El uso tanto diario como mensual de CPU se ilustra en el apéndice C, donde además se describen las características de cicese2000.

Al observar las gráficas correspondientes al uso promedio diario de cada uno de los meses que conforman el periodo analizado, véase figura 58, así como al graficar el uso promedio mensual durante ese periodo, véase figura 59, se observó que cicese2000 tiene un comportamiento parecido al de una campana. Debido a que el uso mensual fue incrementando durante los primeros meses del periodo analizado, pero después de algunos meses el uso promedio empezó a descender, formando una campana, véase figura 59. Con base en esto, se decidió utilizar una distribución Gaussiana para la generación de los números aleatorios.

Para nuestro análisis, el tiempo de ejecución de cada una de las tareas que forman los dígrafos será generado como un número aleatorio con distribución Gaussiana con media  $\mu = 70$  y desviación estándar  $\sigma = 28$ .

### **Distribución Gaussiana**

Se dice que una variable aleatoria  $X$  tiene una distribución Gaussiana con un valor esperado  $\mu_x$  y una varianza  $\sigma_x^2$  cuando  $X$  es  $N(\mu_x, \sigma_x^2)$  distribuida (Pértegas y Pita., 2001). La

función de densidad de probabilidad (FDP) de una variable aleatoria con distribución Gaussiana con parámetros  $\mu_x$  y  $\sigma_x$  es representada por la siguiente expresión:

$$f(x; \mu_x, \sigma_x) = \frac{1}{\sigma_x \sqrt{2\pi}} e^{\left\{ -\frac{1}{2} \left( \frac{x - \mu_x}{\sigma_x} \right)^2 \right\}}. \quad (74)$$

La distribución Gaussiana es continua y tiene una forma de campana, la cual es simétrica sobre su media y puede tomar valores desde negativo infinito a positivo infinito (ver figura 18).

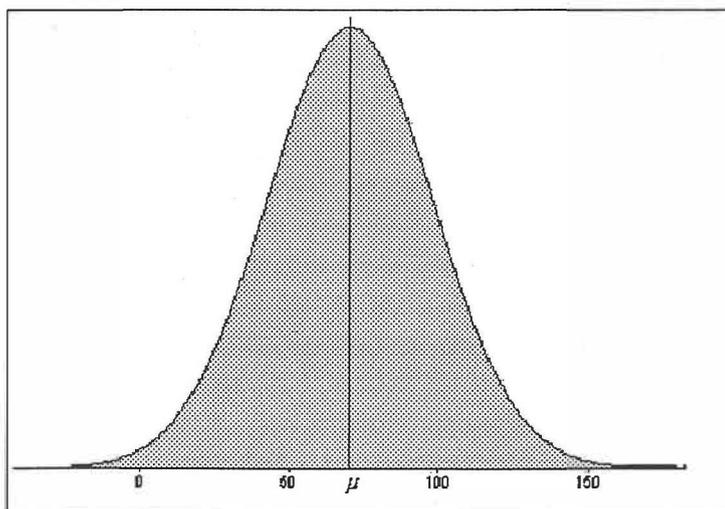


Figura 18. Gráfica de FDP de la distribución Gaussiana con  $\mu_x = 70$  y  $\sigma_x = 28$ .

A continuación se describe el comportamiento de las funciones utilizadas durante los experimentos como factor de ineficiencia.

## VIII.2 *Funciones utilizadas para calcular el factor de ineficiencia*

A fin de analizar el comportamiento del factor de ineficiencia al paralelizar tareas que requieren más de un procesador, se han definido tres funciones como factor de ineficiencia, las cuales muestran un comportamiento distinto.

Debemos tener en cuenta que el factor de ineficiencia es una penalidad por paralelizar una tarea en más de un procesador. Su objetivo fundamental es medir de la mejor forma posible el impacto de las comunicaciones incurridas a consecuencia de la paralelización. Modelar las comunicaciones de una aplicación es relativamente difícil por todos los factores que se ven involucrados.

En un modelo ideal, un sistema con  $m$  procesadores ejecuta una tarea  $m$  veces más rápido que lo que la ejecuta un sistema con un solo procesador. Esto no ocurre en la realidad. Ahora bien, el tiempo de ejecución al paralelizar una tarea maleable en  $m$  procesadores es menor al tiempo en que esa tarea se paraleliza en un solo procesador. Uno de los problemas para toda aplicación es la obtención de una buena estimación del factor de ineficiencia que permita ordenar eficientemente el grafo de tareas maleables. Este factor de ineficiencia se puede evaluar empíricamente o bien analíticamente al analizar la complejidad de las operaciones.

Si el tiempo de ejecución de una tarea que se ejecuta en un solo procesador es  $p_1$ , si ésta se ejecuta en dos procesadores su tiempo de ejecución no puede ser menor a  $\frac{p_1}{2}$ . El tiempo en que ésta se ejecuta en dos procesadores es  $\mu_2 \frac{p_1}{2}$ . Si se ejecuta en cuatro procesadores

el tiempo de ejecución es  $\mu_4 \frac{P_i}{4}$ . Como se observa, el factor de ineficiencia influye en el tiempo de ejecución de una tarea.

Con base en lo anterior, el criterio que se tomó para la selección de las funciones de ineficiencia es que el pago por paralelizar no pase de dos veces el tiempo ideal. Las funciones a utilizar serán:

- Con comportamiento *cóncavo*:

$$\mu_m = \frac{1}{e^{\left(\frac{-m}{200}\right)}} - 0.0035m. \quad (75)$$

- Con comportamiento *convexo*:

$$\mu_m = 2(0.0038m + 0.9975) - \left( \frac{1}{e^{\left(\frac{-m}{200}\right)}} - 0.0035m \right). \quad (76)$$

- Con comportamiento *lineal*:

$$\mu_m = 0.0038m + 0.9975. \quad (77)$$

En estas ecuaciones,

$m$  representa el número de procesadores que se asignan a la tarea.

El comportamiento de dichas funciones es ilustrado en la figura 19.

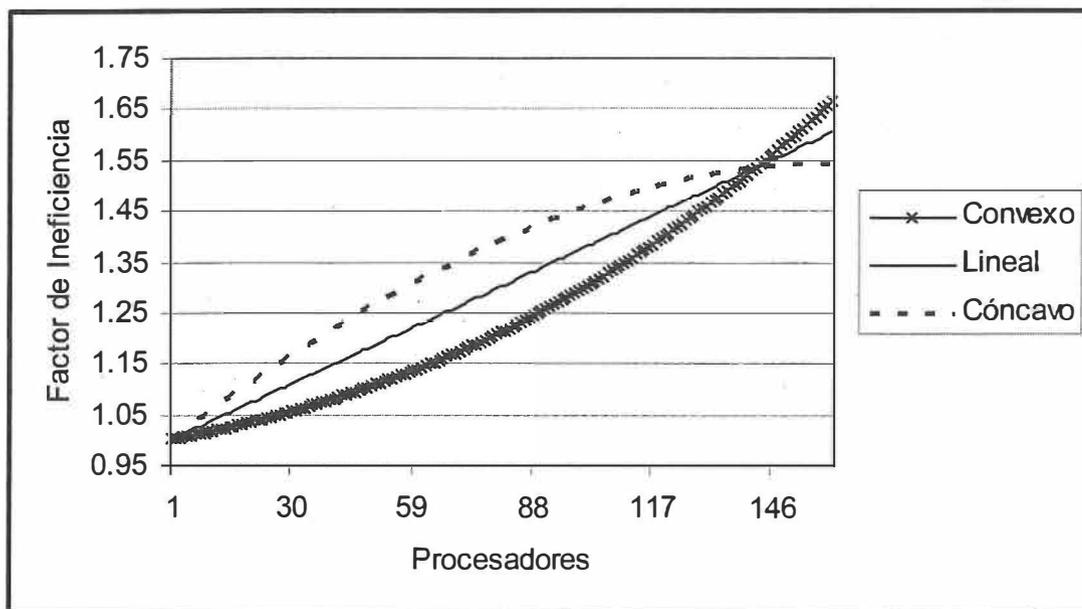


Figura 19. Comportamiento del factor de ineficiencia.

Como podemos observar, el punto de intersección de las funciones utilizadas como factor de ineficiencia es justo en 142 procesadores. Por otra parte, como mencionamos en la sección 4.2, una característica del factor de ineficiencia es el hecho de ser monótonico. De acuerdo a esto, el máximo número de procesadores a utilizar para nuestros experimentos será de 160 procesadores.

### VIII.3 Descripción de las estrategias de calendarización utilizadas

Las estrategias utilizadas con el objetivo de obtener el calendario para la ejecución del conjunto de tareas independientes son las siguientes:

- *Estrategia Graham.* Utiliza la calendarización de lista para ejecutar las tareas. El orden en que arriban las tareas al sistema determina su prioridad. La tarea que tiene

mayor tiempo en el sistema tiene mayor prioridad. Las tareas no se pueden interrumpir una vez iniciada su ejecución.

- *Estrategia Tiempo de Procesamiento Mayor (LPT, Longest Processing Time, por su nombre en inglés)*. Utiliza la calendarización de lista. El criterio de prioridad de las tareas es de acuerdo a su tiempo de procesamiento. A mayor tiempo de procesamiento, mayor prioridad. Una vez que se asigna la tarea a un procesador, ésta no se puede interrumpir.
- *Estrategia Gang*. Se crea una lista con las tareas de acuerdo a su llegada al sistema. La primera en llegar se atiende primero y se asigna a todos los procesadores. Una vez que inicia la ejecución de una tarea, ésta no se puede interrumpir.
- *Estrategia (a,b,c)-Scheme*. A diferencia de las anteriores, esta estrategia permite hacer cambio de estrategia. Una tarea se asigna a un procesador conforme éstas arriban al sistema utilizando para ello la estrategia *Graham*, siempre y cuando no existan procesadores ociosos. El cambio de estrategia se lleva a cabo una vez que se tienen  $a$  procesadores ociosos en el sistema. En la simulación, el usuario asigna el valor de  $a$  antes de iniciar el programa. Las estrategias aplicadas en este momento pueden ser:
  - *Estrategia Gang*. Todas las tareas se interrumpen. Cada tarea se asigna al número total de procesadores y se ejecuta conforme arriba al sistema.
  - *Estrategia Maleable*. Únicamente una tarea se interrumpe, el resto de las tareas continúa su ejecución. La tarea interrumpida se asigna al número total de procesadores libres en el sistema.

Debemos tener en cuenta que el pago por asignar una tarea a más de un procesador es abstraído implícitamente por la función de ineficiencia.

Siguiendo con el objetivo de este documento, los resultados de los calendarios generados por la estrategia determinística LPT se comparan con los de las estrategias no determinísticas mencionadas anteriormente, en busca del mejor resultado.

#### ***VIII.4 Análisis de los resultados experimentales***

La estrategia  $(a,b,c)$ -Scheme representa una nueva opción para llevar a cabo la calendarización de un conjunto de tareas, ya que durante su procesamiento se pueden aplicar dos estrategias distintas. Es precisamente la necesidad de llevar a cabo un análisis del desempeño de esta estrategia lo que motivó la realización de los experimentos.

Cada uno de los calendarios generados muestra un tiempo de procesamiento distinto. El hecho de generar calendarios distintos tiene como objetivo la búsqueda del calendario que muestre mayor eficiencia, la mejor utilización de los recursos del sistema, que reduzca los tiempos ociosos durante la calendarización y, sobretodo, el tiempo total de procesamiento de las tareas.

Para que una aplicación se paralelice eficazmente por medio de  $(a,b,c)$ -Scheme, es necesario encontrar el momento correcto para hacer el llamado cambio de estrategia y evitar que existan muchos procesadores ociosos durante el procesamiento de las tareas. Además, es necesario determinar la forma adecuada de asignar los procesadores a cada tarea, de manera que se obtenga el calendario con el menor tiempo total de procesamiento.

Tomando en cuenta que no se conoce el resultado óptimo de la estrategia  $(a,b,c)$ -Scheme, durante la experimentación comparamos los resultados obtenidos con los resultados

generados por la estrategia de calendarización de lista llamada *LPT*. El objetivo es probar experimentalmente el comportamiento de  $(a,b,c)$ -*Scheme* en relación a una estrategia determinística donde se conoce de antemano toda la información sobre las tareas. Al tomar la estrategia determinística como parámetro de comparación se pretende observar si en promedio  $(a,b,c)$ -*Scheme* logra mejorar o se aleja del desempeño de *LPT*, tomando en cuenta que la nueva estrategia no conoce de antemano toda la información sobre las tareas.

Después de realizar una serie de experimentos (ver Apéndice A) donde se calendarizó un conjunto de tareas maleables independientes por medio de la estrategia  $(a,b,c)$ -*Scheme*, observamos que el desempeño promedio que muestra esta nueva estrategia es más eficiente que el desempeño que se obtiene al calendarizar las tareas únicamente bajo la estrategia *Gang*. Este mismo comportamiento se repitió al variar los valores de  $a$ ,  $b$  y  $c$ . Por lo tanto, podemos decir que a pesar de que el cambio de estrategia se lleve a cabo cuando existen ya sea uno o más procesadores ociosos, la estrategia  $(a,b,c)$ -*Scheme* muestra, en promedio, mejor desempeño que la técnica de distribución *Gang*; no debemos olvidar que en esta última no se tiene ningún procesador ocioso debido a que todos los procesadores se asignan a cada una de las tareas durante el proceso de calendarización.

Además, se observó que el desempeño que muestra  $(a,maleable,c)$ -*Scheme* al incrementar el valor de  $a$  (hasta cierto valor) mejora al mostrado por  $(a,m,0)$ -*Scheme*. Conforme aumenta el valor de  $a$  e incrementa el número de procesadores en el sistema, se mejora el desempeño tanto de  $(a,maleable,1)$ -*Scheme* como de  $(a,m,0)$ -*Scheme*.

A continuación se muestran algunos resultados experimentales obtenidos en el análisis del cociente de desempeño de  $(a,b,c)$ -*Scheme* con  $0 \leq a \leq m$ ,  $b=m$ , y  $c=0$ . Para los

experimentos se generaron 100 casos compuestos por 500 tareas. Para cada  $a$  se calculó de manera independiente el cociente de desempeño.

La siguiente figura muestra el cociente de desempeño promedio cuando se varía el parámetro  $a$  con un número fijo de procesadores, donde  $\mu$  es una función de  $m$  (cóncava, convexa o lineal).

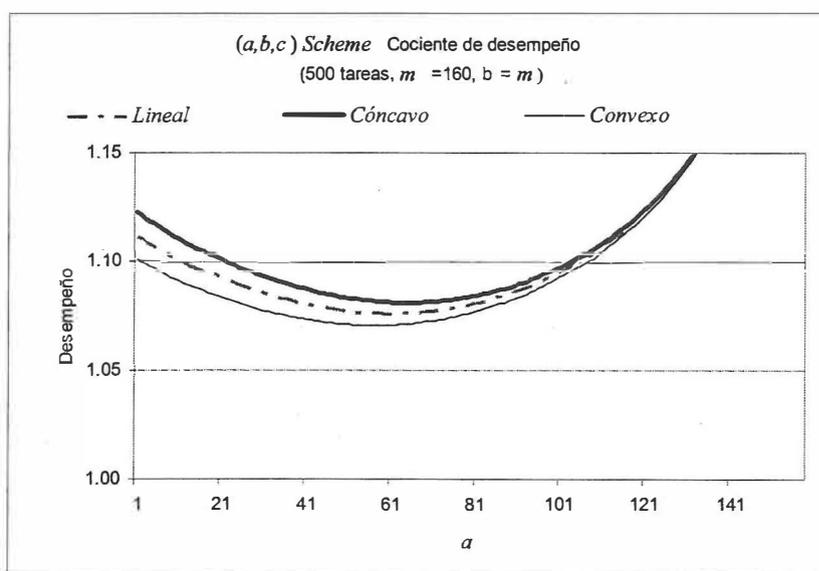


Figura 20. Cociente de desempeño promedio de  $(a,b,c)$ -Scheme con 100 experimentos, donde  $n=500$ ,  $m=160$ .

Por otra parte, la figura 21 muestra el cociente de desempeño promedio cuando se varía el número de procesadores  $m$  con un valor fijo de  $a=30$ , donde  $\mu$  es una función de  $m$  (cóncava, convexa o lineal).

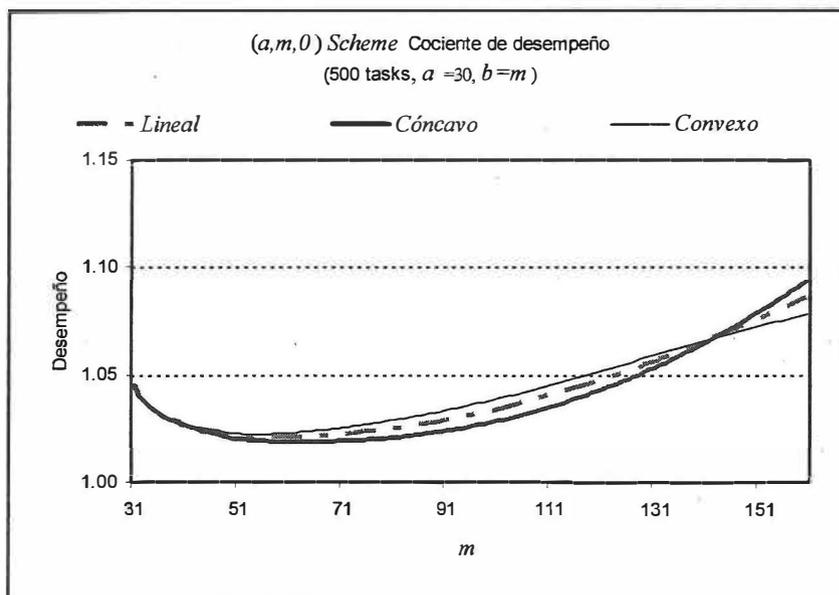


Figura 21. Cociente de desempeño promedio de  $(a,b,c)$ -Scheme (100 experimentos,  $n=500$ ,  $a=30$ ).

La siguiente figura muestra el cociente de desempeño promedio de Gang versus el número de procesadores.

Obsérvese que la paralelización de todas las tareas aplicando Gang nos da el peor caso al compararlo con el método dos pisos que retarda la paralelización de las tareas hasta un cierto umbral determinado por  $a$  en  $(a,b,c)$ -Scheme. (véase la figura 22).

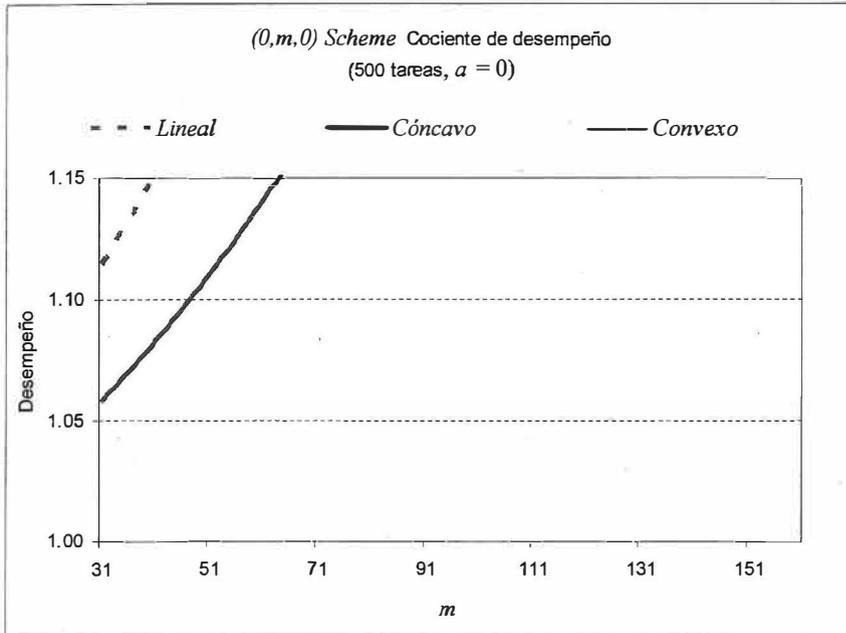


Figura 22. Cociente de desempeño promedio de la estrategia de distribución Gang (100 experimentos,  $n=500$ ,  $m$  varía de 31 a 160).  $\mu$  es una función lineal, cóncava y convexa.

Otros de los resultados experimentales que se obtuvieron son los correspondientes al desempeño promedio de la estrategia  $(a, maleable, 1)$ -Scheme al variar el parámetro  $a$ , aplicando los tres factores de ineficiencia (función lineal, cóncava y convexa) y tomando un número fijo de procesadores ( $m=160$ ), como lo muestra la figura 23. A medida que incrementa el valor de  $a$ , el desempeño mostrado decrementa.

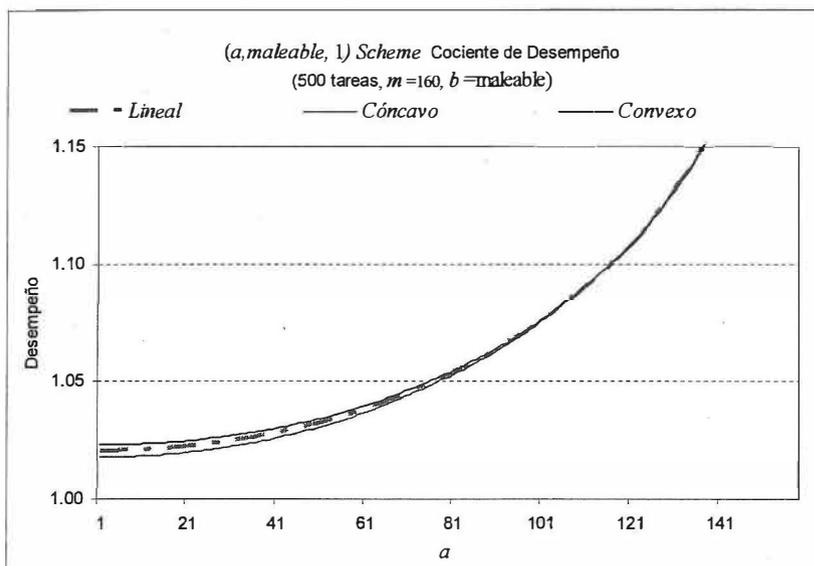


Figura 23. Cociente de desempeño promedio de la estrategia ( $a, maleable, 1$ )-Scheme (100 experimentos,  $n=500$ ,  $m=160$ ).

Ahora observemos el desempeño de ( $a, maleable, 1$ )-Scheme cuando  $a=30$  y varía el número de procesadores en el sistema, aplicando una  $\mu$  lineal, cóncava y convexa. Véase figura 24.

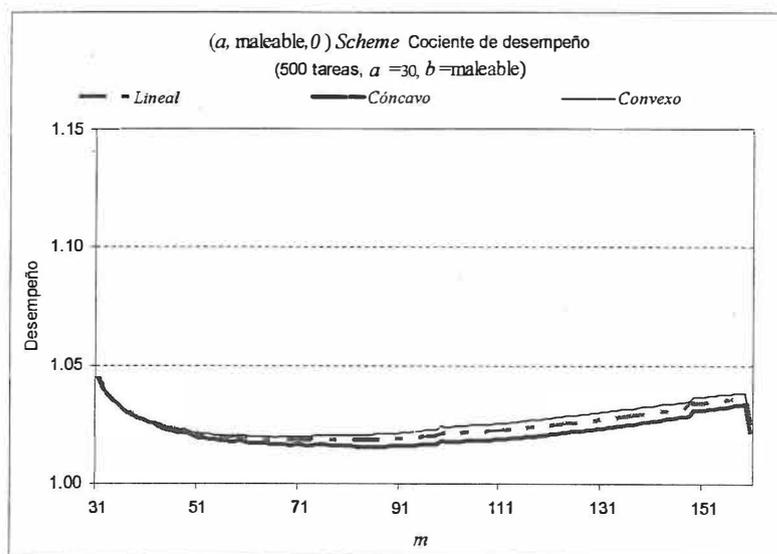


Figura 24. Cociente de desempeño promedio de la estrategia ( $a, maleable, 1$ )-Scheme (100 experimentos,  $n=500$ ,  $a=30$ ).

Por otra parte, al incrementar el parámetro  $a$ , la diferencia entre el desempeño de  $(a, \text{maleable}, 1)$ -Scheme y  $(a, m, 0)$ -Scheme es cada vez más pequeña después de cierto valor alcanzado por  $a$ , como lo muestra la figura 25.

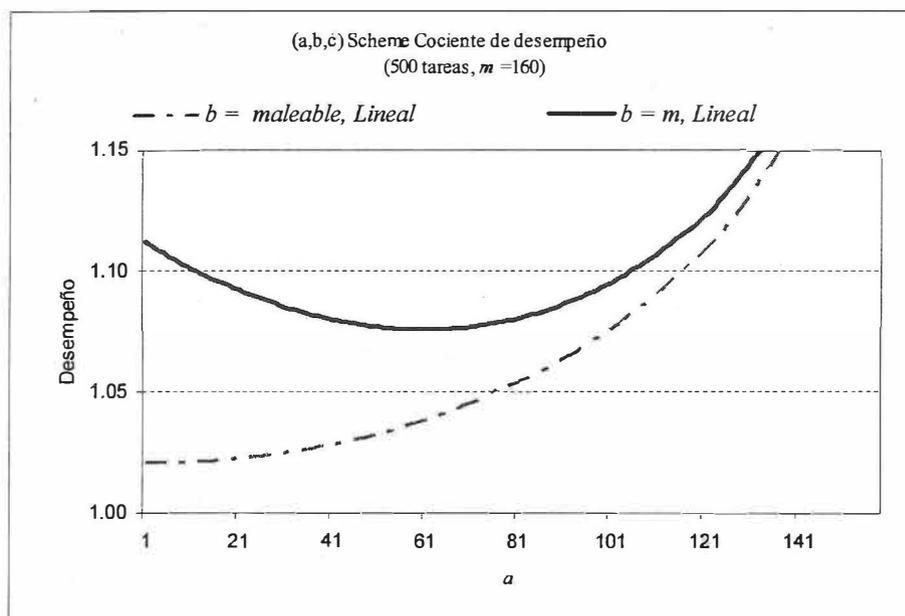


Figura 25. Cociente de desempeño promedio de las estrategias  $(a, \text{maleable}, 1)$ -Scheme y  $(a, m, 0)$ -Scheme (100 experimentos,  $n=500$ ,  $m = 160$ ).  $\mu$  es una función lineal.

A medida que se incrementa el número de procesadores en el sistema, el desempeño de  $(a, \text{maleable}, 1)$ -Scheme y  $(a, m, 0)$ -Scheme mejora hasta un cierto límite, después del cual el desempeño de ambos casos disminuye conforme aumenta el número de procesadores en el sistema, tal como se observa en la siguiente figura.

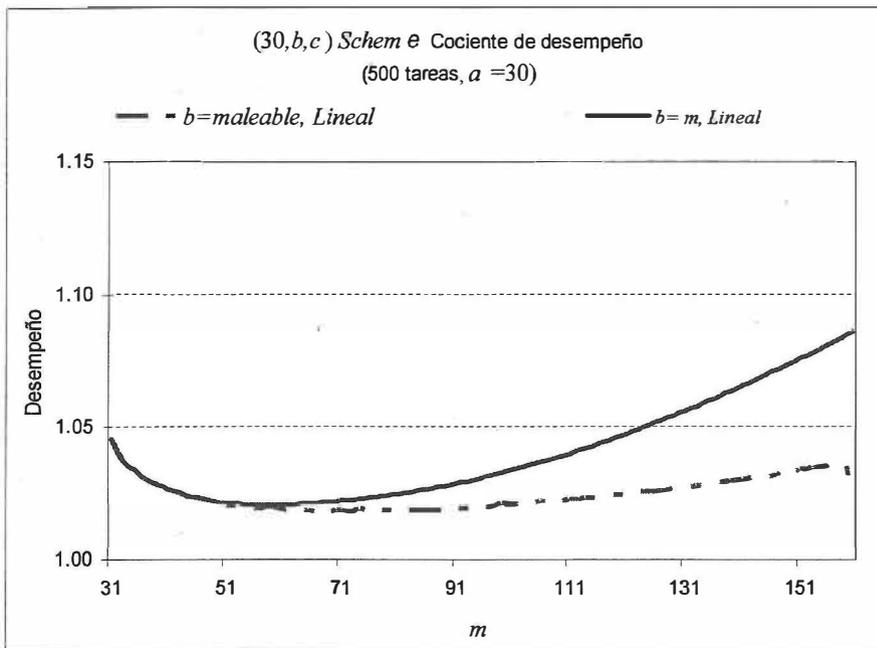


Figura 26. Cociente de desempeño promedio de  $(30, \text{maleable}, 1)$ -Scheme y  $(30, m, 0)$ -Scheme (100 experimentos,  $n=500$ ,  $m$  varía de 31 a 160).  $\mu$  es una función lineal.

Ahora bien, si comparamos los resultados de la estrategia *LPT* con los mostrados por  $(a, \text{maleable}, 1)$ -Scheme, podemos observar que esta última muestra, en promedio, ser más eficiente a medida que se tiene mayor número de procesadores en el sistema.

Obsérvese que al incrementar  $a$  y después de que ésta alcanza cierto valor, y conforme incrementa el número de procesadores en el sistema, durante ciertos intervalos se mejora el desempeño de  $(a, m, 0)$ -Scheme y ésta logra ser más eficiente que la estrategia *LPT*.

Al incrementar  $a$  al igual que el número de procesadores en el sistema, se reducen cada vez más los intervalos en los que  $(a, \text{maleable}, 1)$ -Scheme y  $(a, m, 0)$ -Scheme llegan a ser más eficientes que *LPT*, hasta que llega el momento en que ambas son menos eficientes que *LPT*.

Otro punto que observamos, es el hecho de que disminuye el desempeño de  $(1,m,0)$ -Scheme conforme incrementa el número de procesadores en el sistema.

Conforme aumenta el valor de  $a$ , se reduce el intervalo en el que  $(a,m,0)$ -Scheme lograr mejor desempeño que  $LPT$  y este intervalo disminuye conforme mayor sea el número de procesadores que se tenga. Entre más cercano es el valor de  $a$  al valor de 160, menor es la posibilidad de que  $(a,m,0)$ -Scheme pueda mejorar el desempeño de  $LPT$ .

Por otro lado, analizando las gráficas de los experimentos donde se tomó como número fijo 160 procesadores, observamos lo siguiente:

- Al incrementar el valor de  $a$ , el desempeño que muestran  $(a,maleable,1)$ -Scheme y  $(a,m,0)$ -Scheme es cada vez menos eficiente.
- El desempeño de las estrategias  $(a,maleable,1)$ -Scheme y  $(a,m,0)$ -Scheme es más eficiente que el desempeño que muestra la estrategia *Gang*, sin importar cual de las tres funciones descritas como  $\mu_m$  se haya aplicado.
- $(a,maleable,1)$ -Scheme muestra mejor desempeño que  $(a,m,0)$ -Scheme. Durante todos los experimentos, el desempeño de  $(a,m,0)$ -Scheme es menos eficiente que el de  $LPT$ , conforme aumenta  $a$  mayor es la diferencia entre sus desempeños.
- Cuando se tiene aproximadamente más de la mitad de los procesadores trabajando,  $(a,maleable,1)$ -Scheme logra mejor desempeño que la estrategia  $LPT$ . A medida que aumenta el valor de  $a$ , que nos indica el momento de hacer cambio de estrategia, es más eficiente el desempeño de  $LPT$  que el de  $(a,maleable,1)$ -Scheme.

De acuerdo a la graficas mostradas en el apéndice A, podemos mencionar que el método de dos pasos propuesto por  $(a,b,c)$ -Scheme logró mejorar en algunos intervalos el desempeño de una estrategia de calendarización determinística.

Los resultados muestran que entre los dos casos propuestos por  $(a,b,c)$ -Scheme el caso  $(a,maleable,1)$ -Scheme logró alcanzar un mejor desempeño.

Al analizar experimentalmente un sistema compuestos por 160 procesadores, se observó que al asignarle al parámetro  $a$  un valor entre 1 y 80 aproximadamente la estrategia logró mejorar el desempeño de  $LPT$ . Y a medida que se reducía el valor de  $a$ , mostraba mejor desempeño con respecto a  $LPT$ .

El hecho de que las tareas analizadas se puedan asignar en más de un procesador y cuyo número no es fijo resultó ser una gran ventaja en los resultados que muestra la estrategia analizada. Lo anterior se debe a que se reducen los tiempos ociosos que pueden ocurrir durante la calendarización y es menor el pago al paralelizar una tarea conforme los procesadores que están disponibles.

## IX Conclusiones y trabajo futuro

Antes de dar nuestras conclusiones, es importante recordar que el problema de calendarización planteado en este documento se centra en conjuntos de tareas independientes cuyo tiempo de procesamiento se desconoce de antemano. Para llevar a cabo este trabajo fue necesario realizar el estudio de las diferentes estrategias de calendarización que habrían de utilizarse, así como de las características del modelo de tareas maleables y del factor de ineficiencia.

Ahora bien, el motivo principal que llevó al análisis de la estrategia  $(a,b,c)$ -Scheme se centra en la característica tan especial que presenta al permitir que las tareas sean calendarizadas por dos estrategias distintas durante su procesamiento.

En este trabajo se analizó, tanto teórica como experimentalmente, el desempeño de la estrategia  $(a,b,c)$ -Scheme. Teóricamente se mostró el cociente de desempeño de la estrategia de acuerdo a ciertos valores asignados a los parámetros  $a$  y  $b$ , utilizando  $c = 0$ . Por otra parte, se realizó una comparación del desempeño de la estrategia por medio de una serie de experimentos en los cuales varían los valores asignados a los parámetros  $a$ ,  $b$  y  $c$ .

Como se esperaba, la estrategia  $(a,b,c)$ -Scheme en promedio logró mejorar el resultado mostrado por *LPT* a pesar de que no siempre se cumplió esta situación, ya que se encontraron situaciones donde no fue conveniente realizar el cambio de estrategia, debido a que al hacerlo no se logró superar el resultado de *LPT*.

De forma experimental se comprobó que  $(a,b,c)$ -Scheme tiene mejor desempeño en promedio si al momento de hacer el cambio de estrategia los procesadores se asignan a una

tarea conforme éstos se liberan, a diferencia del hecho de asignar todos los procesadores a una sola tarea.

Se estudió el mejor momento para realizar el cambio de estrategia de forma que la calendarización se logre en el menor tiempo posible y con el menor número de procesadores ociosos. En cuanto a este aspecto se obtuvieron algunos momentos donde es posible lograrlo en forma eficiente. Es necesario llevar a cabo más experimentos con diferente número de tareas y con ello tratar de determinar los valores más adecuados para resolver la pregunta de “¿cuándo es el momento más adecuado para el cambio de estrategia?”. Hasta el momento, y en base los experimentos realizados podemos, ver que esto se cumple siempre y cuando al parámetro  $a$  no se le asigne un valor mayor del 40 % del total de los procesadores.

En lo que respecta a la forma más adecuada de seleccionar la tarea a paralelizar, los resultados teóricos únicamente se basan en una sola forma de llevarlo a cabo; para efectos de experimentación se trabajó con las dos primeras opciones permitidas por el parámetro  $c$  y se observó que la segunda genera un mejor resultado.

Para llevar a cabo la experimentación se diseñó e implementó una herramienta encargada de llevar a cabo la calendarización de las tareas bajo la estrategia estudiada. Se experimentó con un conjunto de tareas para la obtención del desempeño promedio de la estrategia  $(a,b,c)$ -Scheme, así como de su desviación estándar.

Se asignó a los parámetros  $a$ ,  $b$ , y  $c$  valores distintos a los analizados teóricamente, a fin de observar de manera experimental el desempeño promedio de la estrategia para esas características.

Por primera vez una estrategia de calendarización permite que una tarea se ejecute por más de un procesador a la vez y aplicando más de una técnica de calendarización.

Una vez presentada la base teórica de la nueva estrategia y de las técnicas que ésta aplica, así como el haber contribuido con los primeros experimentos basados en dicha estrategia, podemos decir que se ha cumplido con el objetivo de este trabajo de investigación. Sabemos que aún existe mucho trabajo de investigación por hacer sobre esta nueva técnica.

### ***IX.1 Aportaciones***

Por ser la estrategia  $(a,b,c)$ -Scheme una estrategia recientemente propuesta, se considera que nuestra principal aportación es la presentación de los experimentos realizados de algunos de los casos permitidos por esta estrategia. Con esto, se demuestra de forma experimental que efectivamente esta nueva estrategia, bajo ciertas circunstancias, puede lograr una calendarización más eficiente de las tareas con un menor número de procesadores ociosos y una mejor utilización de los recursos.

En base a los experimentos quedó demostrado que el asignar todos los procesadores a cada una de las tareas de acuerdo a estrategias conocidas para computadoras reales, no siempre va a representar una mejor estrategia de calendarización, a consecuencia del pago que hay que realizar por la paralelización.

Se mostró experimentalmente cuál de las dos opciones presentadas por esta nueva estrategia genera un calendario más cercano al óptimo.

Se desarrolló el calendarizador capaz de realizar experimentos con este tipo de estrategia.

Se llevó a cabo un análisis sobre la fórmula a utilizarse como factor de ineficiencia para los experimentos.

Se mostró el comportamiento de la nueva estrategia más allá de los casos desarrollados teóricamente.

## ***IX.2 Trabajo Futuro***

Existen aún muchos aspectos por investigar en relación a esta nueva estrategia, tanto teóricos como experimentales, por ello se propone el desarrollar a profundidad todas las posibilidades mencionadas por esta estrategia tanto para asignar procesadores, así como de la forma de seleccionar las tareas y del momento de llevar a cabo el cambio de estrategia.

Tratar de encontrar una mejor fórmula a aplicar como factor de ineficiencia en los experimentos.

Realizar experimentos con un conjunto de tareas mayor.

Encontrar el caso óptimo, peor caso y caso promedio para cada una de las opciones presentadas por  $(a,b,c)$ -Scheme.

Otro punto que es importante realizar en un futuro es el probar el desempeño de esta estrategia en aplicaciones manejadas en el mundo real, más allá de la experimentación aquí realizada. Analizar el desempeño de la estrategia al aplicarla quizá en un cluster de computadoras.

Se propone la implementación de las opciones contempladas por la estrategia de calendarización dentro de la herramienta *Scheduling*.

Ahora bien, otro aspecto que se propone es el hecho de que este nuevo método de dos pasos se aplique no únicamente a tareas maleables independientes, sino que además permita trabajar con tareas maleables dependientes.

Por último, con el objetivo de contar con una herramienta más completa que apoye en el análisis experimental de las distintas estrategias de calendarización, se propone el agregar a la herramienta la opción de mostrar gráficamente el desempeño de cada una de las estrategias aplicadas a un conjunto de tareas.

## X Referencias y Literatura citada

- Chen, G., y Lai, T. 1998. "Preemptive scheduling of independent jobs on a hypercube". *Information Processing Letters*. 28(4): 201-206 p.
- Blazewicz, J., Drozdowski, M., y Ecker, K. 2000. "Handbook on Parallel and Distributed processing: Management of Resources in Parallel Systems". Springer-Verlag. Primera edición. Berlin. 635 pp.
- Blazewicz, J., Drozdowski, M., y Markiewicz, M, 1999. "Divisible task scheduling - Concept and verification". *Parallel Computing*. 25(1): 87-98 p.
- Blazewicz, J., Machowiak, M., Mounié, G., Trystram, D., y Weglarz, J. 2000. "Scheduling malleable tasks with convex processing speed functions". *Computacion y sistemas*. 12(1):158-165 p.
- Drozdowski, M. 1996. "Scheduling multiprocessor tasks - An overview". *European Journal of Operations Research*. 94:215-230 p.
- Feitelson, D., Rudolph, L., Schweigelshohn, U., Sevcik, KC., y Wong, P. 1997. "Theory and practice in parallel job scheduling". *IPPS'97 Workshop on Job Scheduling Strategies for Paralle Processing*. Geneva. Feitelson, D., y Rudolph, L. 1-34 p.
- Feitelson, D., y Rudolph, L. 1996. "Toward convergence in job schedulers for parallel supercomputers". En Feitelson, D., y Rudolph, L. (eds). *Lecture Notes in Computer Science*. Springer- Verlag, Berlin, 1-26 p.
- Jansen, K. y Porkolab, L. 1999. "Linear time approximation schemes for scheduling problems". *Conference Proceedings in 10<sup>th</sup> Annual ACM-SIAM Symposium on Discrete Algorithms*. Baltimore. Meinel, C., y Bern., J. 490-498 p.

- Lepère, R., Mounié, G., Trystram, D., y Robič, B. 2000. "Malleable Tasks: An efficient model for solving actual parallel applications". *Parallel Computing: Fundamentals and Applications. Proceeding of the International Conference Parallel Computing. Delft. Joubert, G.* 598-605 p.
- Ludwig, W. 1995. "Algorithms for scheduling malleable and nonmalleable parallel tasks". Ph.D. thesis University of Wisconsin – Madison, Department of Computer Sciences. 128 pp.
- Mounié, G. 2000. "Ordonnancement efficace d'applications parallèles: les tâches malléables monotones". PhD thesis Institut National Polytechnique de Grenoble. 135 pp.
- Mounié, G., Rapine, C., y Trystram, D. 1999. "Efficient Approximation Algorithms for Scheduling Malleable Tasks". *ACM Symposium on Parallel Algorithms and Architectures. Saint Malo. Sitaraman., R.* 23-32 p.
- Pértegas, S., y Pita, S. "Metodología de la investigación: la distribución normal". Coruña. Unidad de Epidemiología Clínica y Bioestadística, Complejo Hospitalario Juan Canalejo. (Dirección electrónica, 11 de noviembre, 2001: [http://www.fisterra.com/material/investiga/distr\\_normal/distr\\_normal.html](http://www.fisterra.com/material/investiga/distr_normal/distr_normal.html)).
- Rapine, C., Scherson, I., y Trystram, D. 1998. "On-Line Scheduling of Parallelizable Jobs". Pritchard, D., y Reeve, J., (eds). *Lecture Notes in Computer Science. Springer-Verlag, Germany*, 322-327 p.
- El-Rewini, H., Lewis, T., y Ali, H. 1994. "Task Scheduling in Parallel and Distributed Systems". Prentice Hall. Eaglewood Cliffs, New Jersey. 290 pp.

- Silva, F., Campos, L., y Scherson, I. 1998. "Improvements in Gang Scheduling for Parallel Supercomputers". Conference Proceeding Parallel Computing Workshop. Singapur. P2-H1-P2-H5 p.
- Silva, F., Campos, L., y Scherson, I. 1998. "A Lower Bound for Dynamic Scheduling of Data Parallel Programs". Conference Proceeding Euro-Par. Southampton, UK. 367-372 p.
- Tchenykh, A., Trystram, D., y Rapine, C. 2002. "Adaptive Strategy for On-line Scheduling". International Institute of Informatics and Systemics Press. XVI: 354-359 p.
- Turek, J., Wolf, J., Pattipati, K., y Yu, P. 1992. "Scheduling Parallelizable Tasks: Putting it All on the Shelf". Performance Evaluation Review, 20 (1): 225-236 pp.
- Turek, J., Wolf, J., y Yu, P. 1992. "Approximate algorithms for scheduling parallelizable tasks". Conference Proceeding in 4<sup>th</sup> Annual ACM Symposium on Parallel Algorithms and Architectures. 323-332 p.
- Zomaya, A. 1996. "Parallel & Distributed Computing Handbook". McGraw-Hill. Primera edición. New York. 1232 pp.

## Apéndice A. Resultados experimentales con conjunto de 500 tareas y tiempos de procesamiento no unitarios.

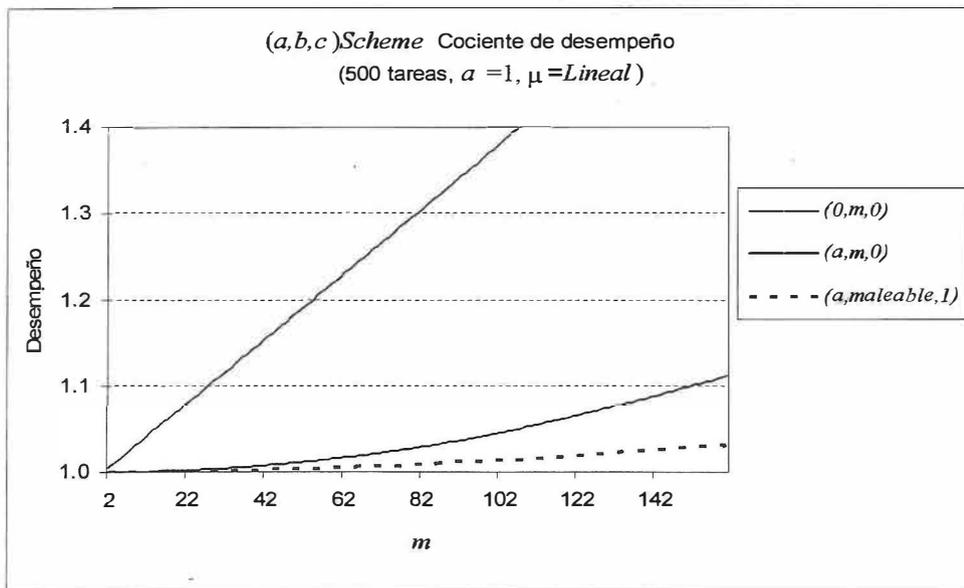


Figura 27. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a = 1$  y diferentes valores de  $b$ .  $m$  varía de 2 a 160 procesadores, aplicando un factor de ineficiencia lineal.

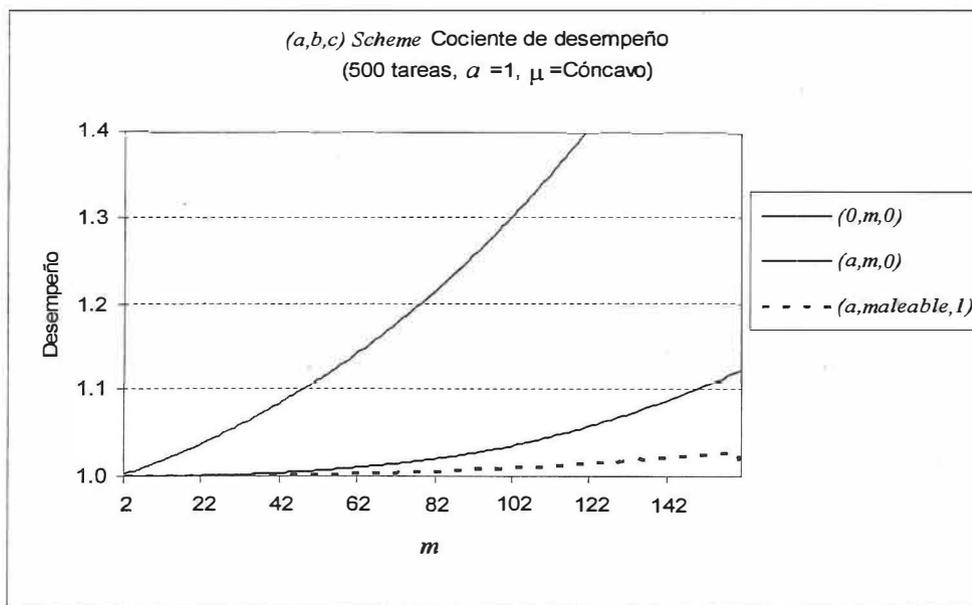


Figura 28. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a = 1$  y diferentes valores de  $b$ .  $m$  varía de 2 a 160 procesadores, aplicando un factor de ineficiencia cóncavo.

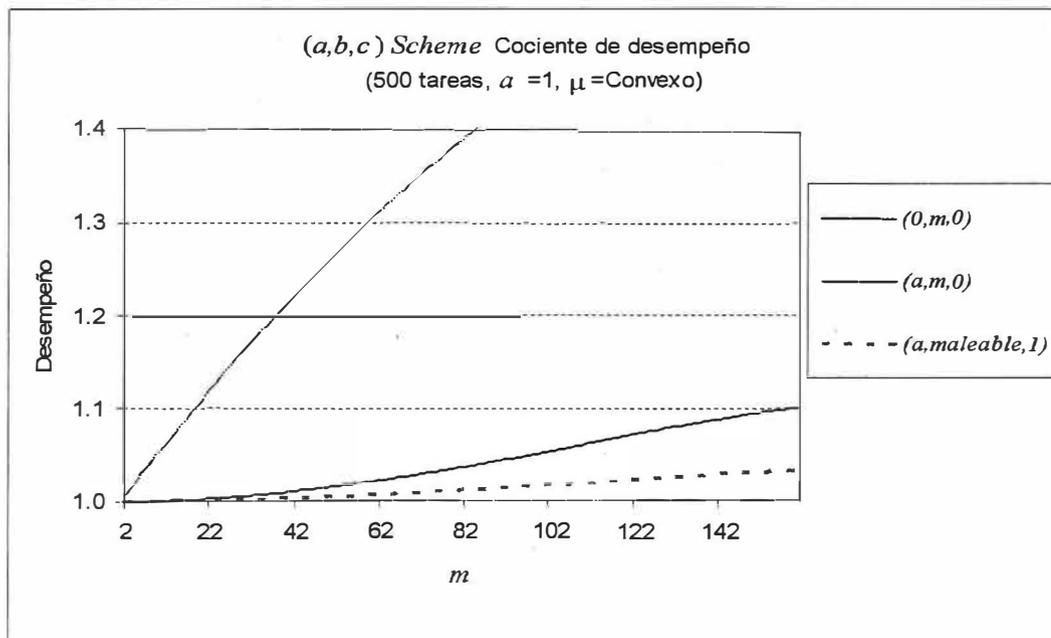


Figura 29. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a = 1$  y diferentes valores de  $b$ .  $m$  varía de 2 a 160 procesadores, aplicando un factor de ineficiencia convexo.

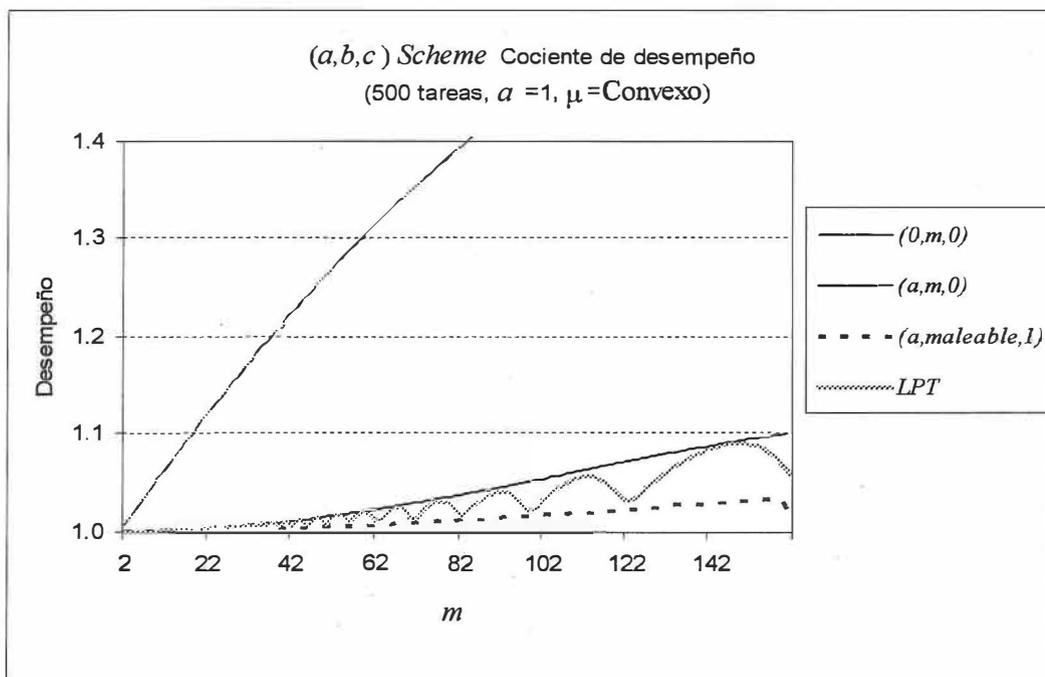


Figura 30. Desempeño promedio de 100 experimentos de LPT y  $(a,b,c)$ -Scheme para  $a = 1$  y diferentes valores de  $b$ .  $m$  varía de 2 a 160 procesadores, aplicando un factor de ineficiencia convexo.

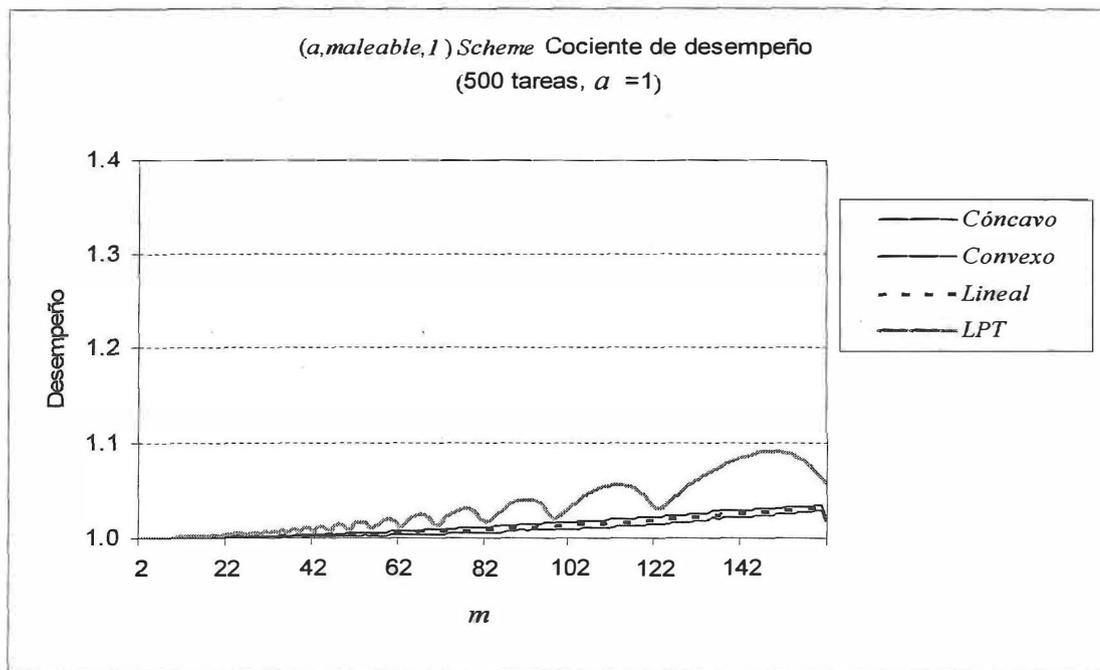


Figura 31. Desempeño promedio de 100 experimentos de  $(a,maleable,1)$ -Scheme para  $\alpha=1$ .  $m$  varía de 2 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal.

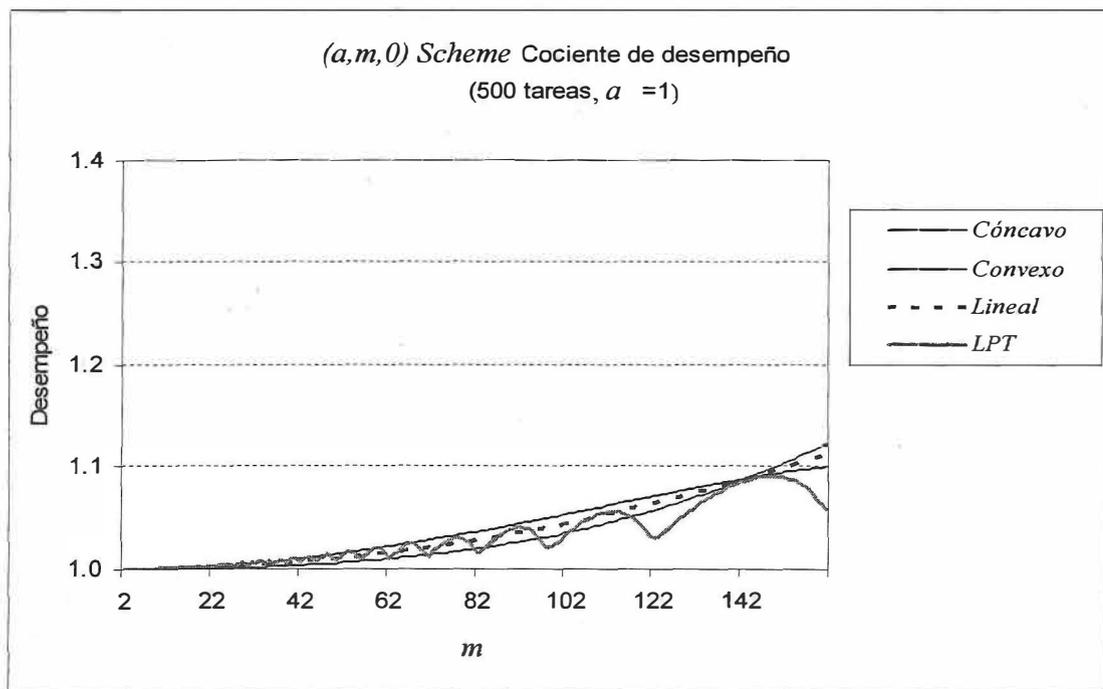


Figura 32. Desempeño promedio de 100 experimentos de  $(a,m,0)$ -Scheme para  $\alpha=1$ .  $m$  varía de 2 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal.

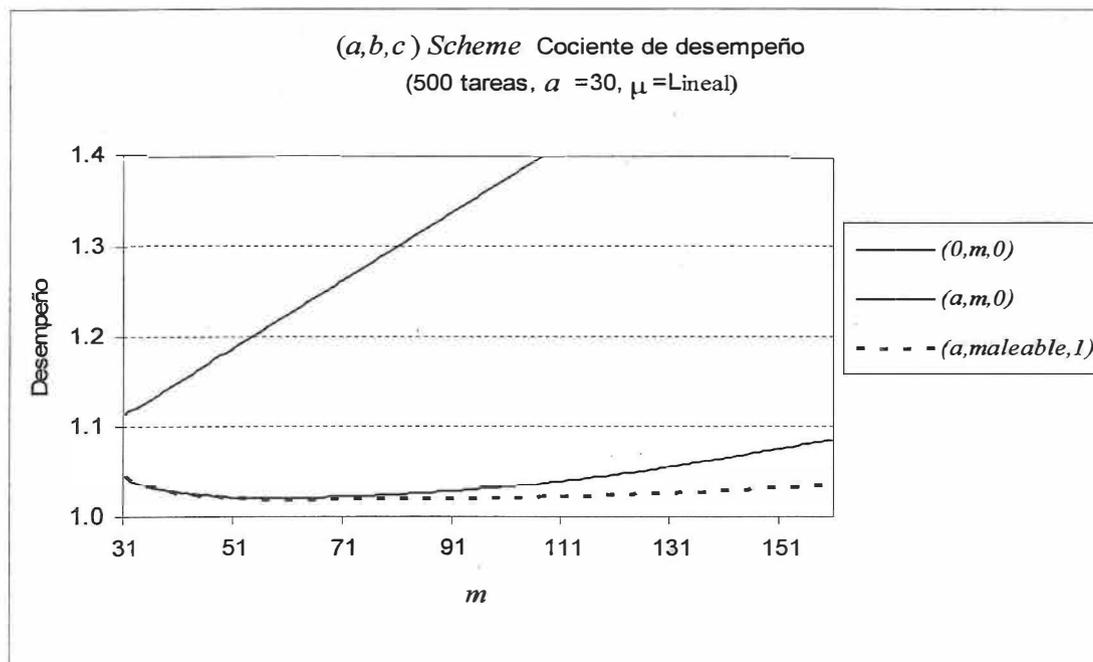


Figura 33. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a=30$  y diferentes valores de  $b$ .  $m$  varía de 31 a 160 procesadores, aplicando un factor de ineficiencia lineal.

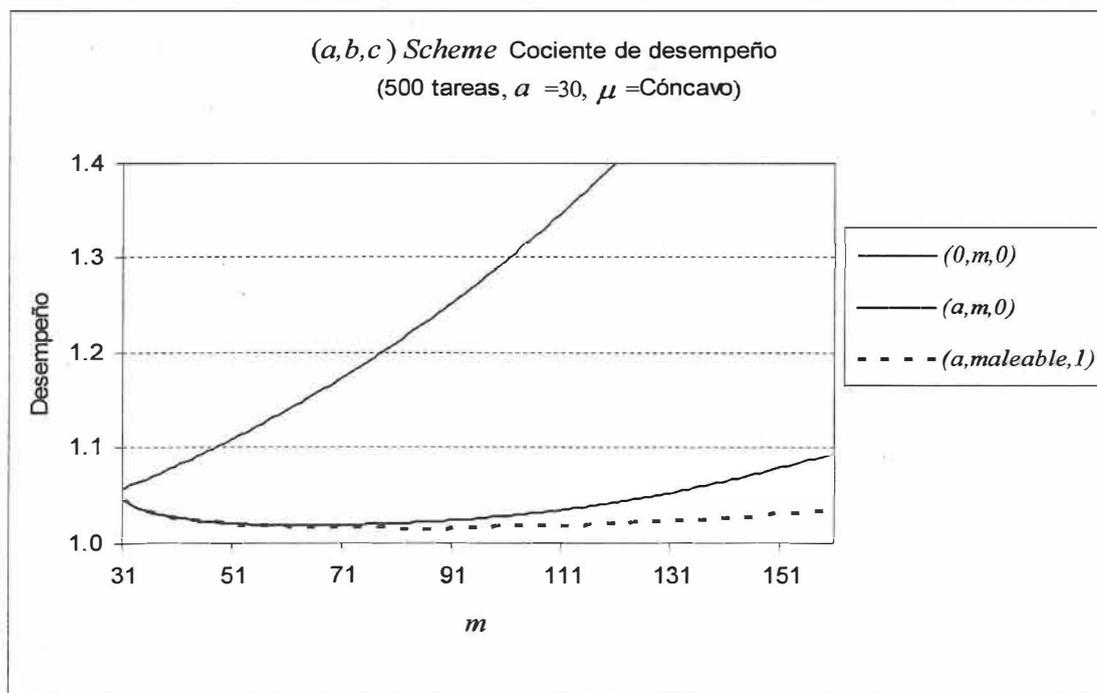


Figura 34. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a=30$  y diferentes valores de  $b$ .  $m$  varía de 31 a 160 procesadores, aplicando un factor de ineficiencia cóncavo.

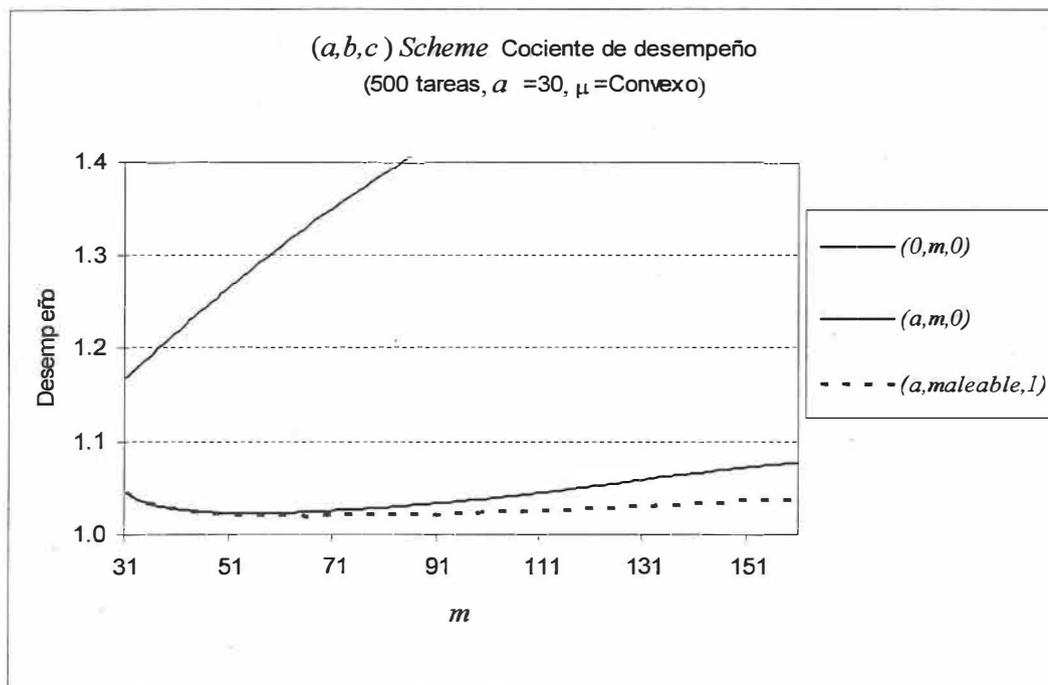


Figura 35. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a=30$  y diferentes valores de  $b$ ,  $m$  varía de 31 a 160 procesadores, aplicando un factor de ineficiencia convexo.

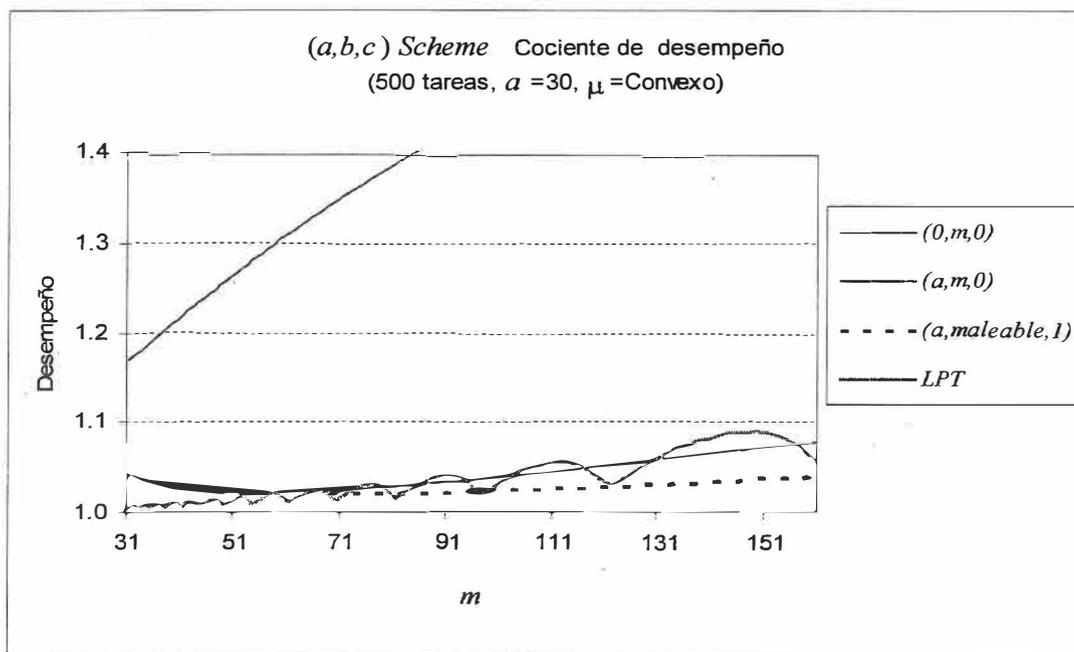


Figura 36. Desempeño promedio de 100 experimentos de LPT y  $(a,b,c)$ -Scheme para  $a=30$  y diferentes valores de  $b$ .  $m$  varía de 31 a 160 procesadores, aplicando un factor de ineficiencia convexo.

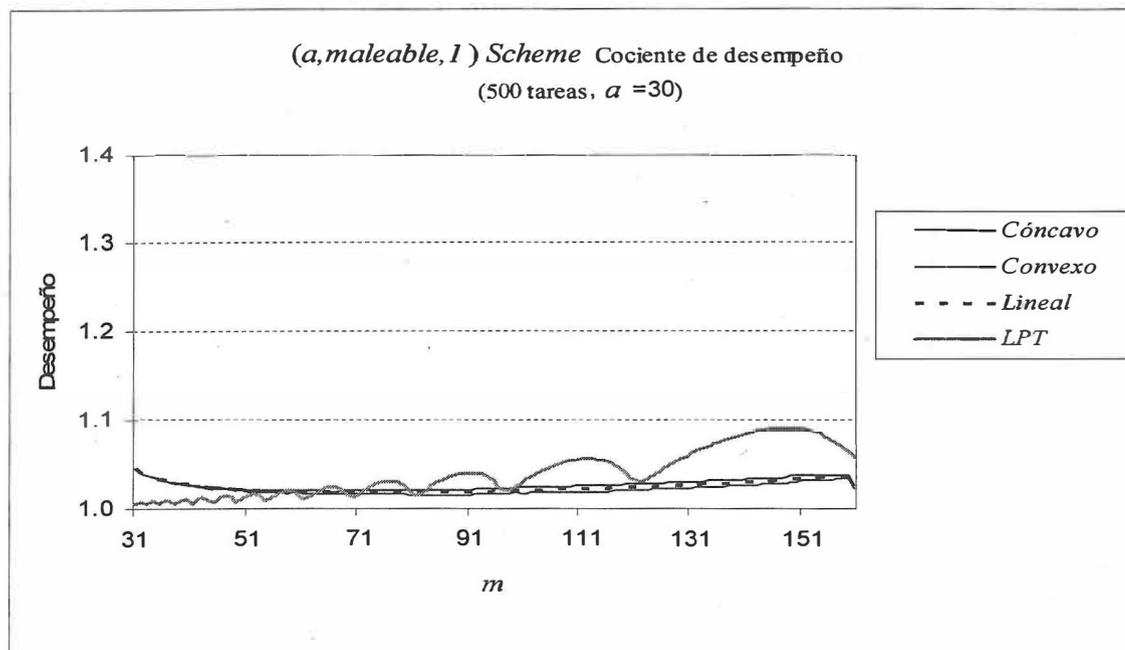


Figura 37. Desempeño promedio de 100 experimentos de  $(a, maleable, 1)$ -Scheme para  $\alpha=30$ .  $m$  varía de 31 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal.

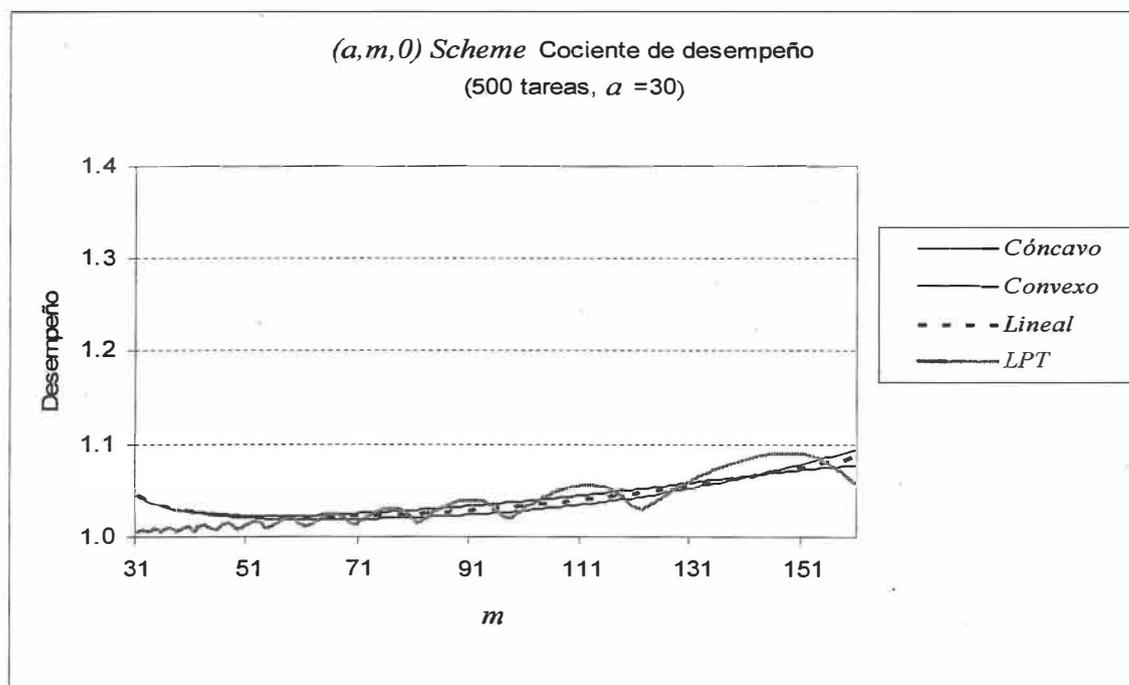


Figura 38. Desempeño promedio de 100 experimentos de  $(a, m, 0)$ -Scheme para  $\alpha=30$ .  $m$  varía de 31 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal.

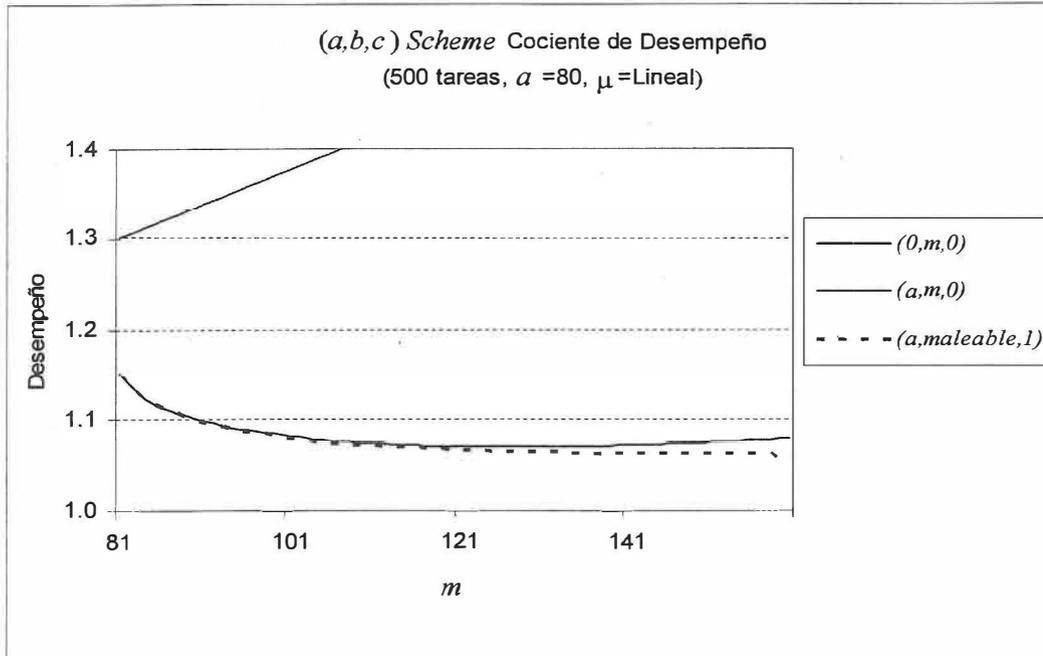


Figura 39. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a=80$  y diferentes valores de  $b$ .  $m$  varía de 81 a 160 procesadores, aplicando un factor de ineficiencia lineal.

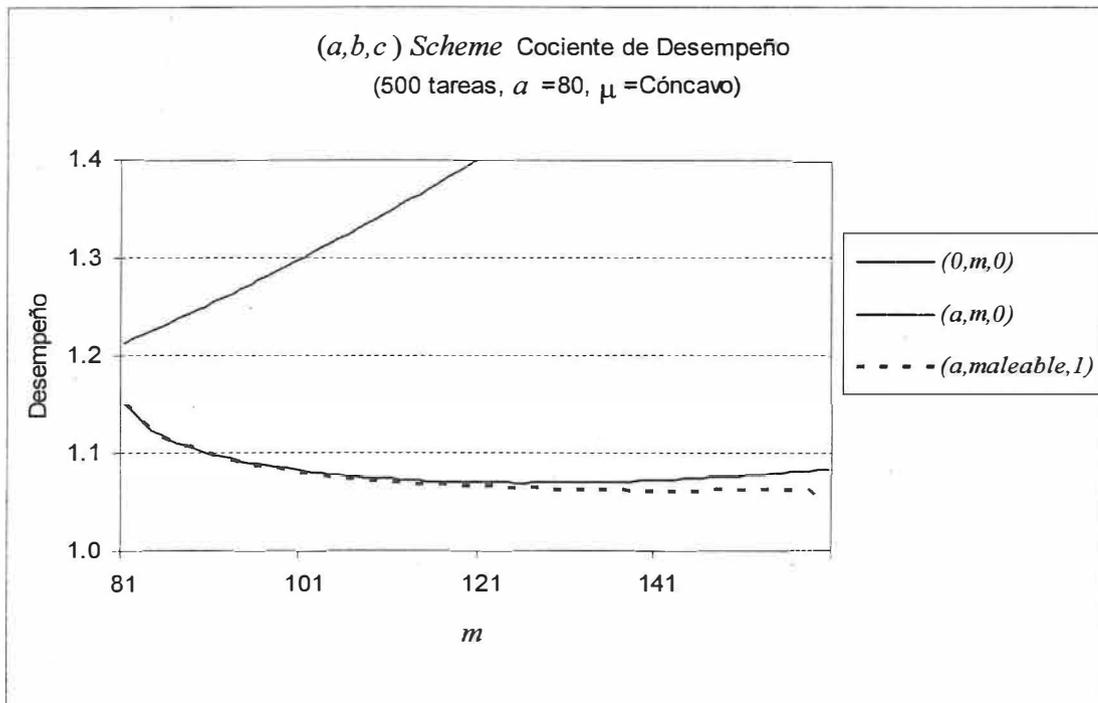


Figura 40. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a=80$  y diferentes valores de  $b$ .  $m$  varía de 81 a 160 procesadores, aplicando un factor de ineficiencia cóncavo.

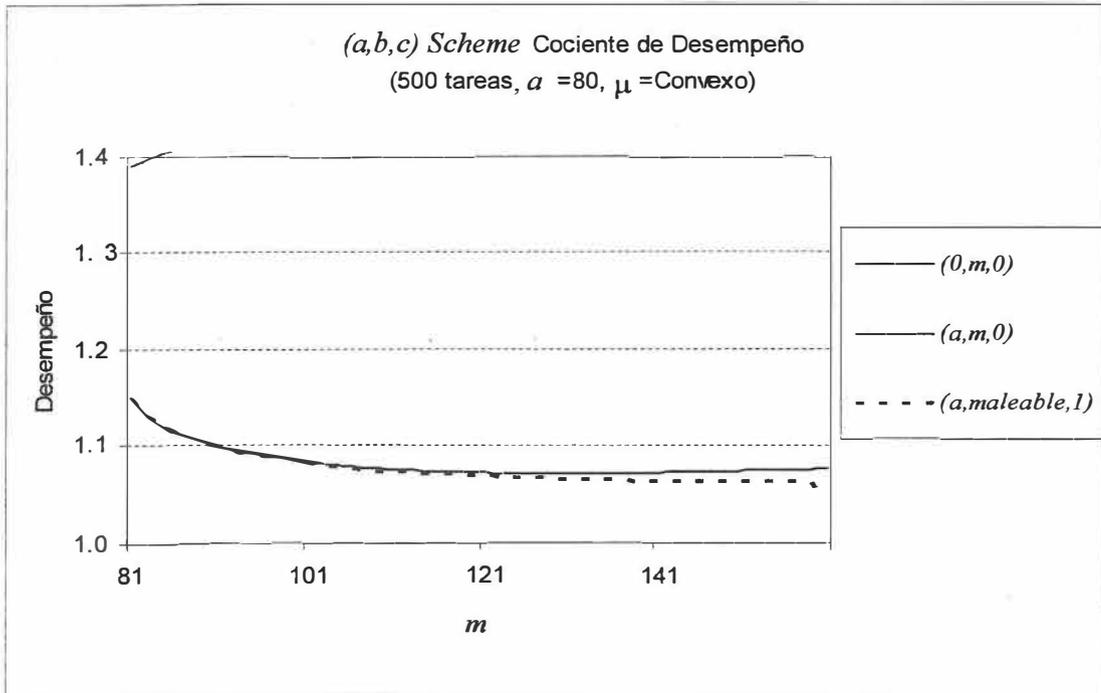


Figura 41. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a=80$  y diferentes valores de  $b$ .  $m$  varía de 81 a 160 procesadores, aplicando un factor de ineficiencia convexo.

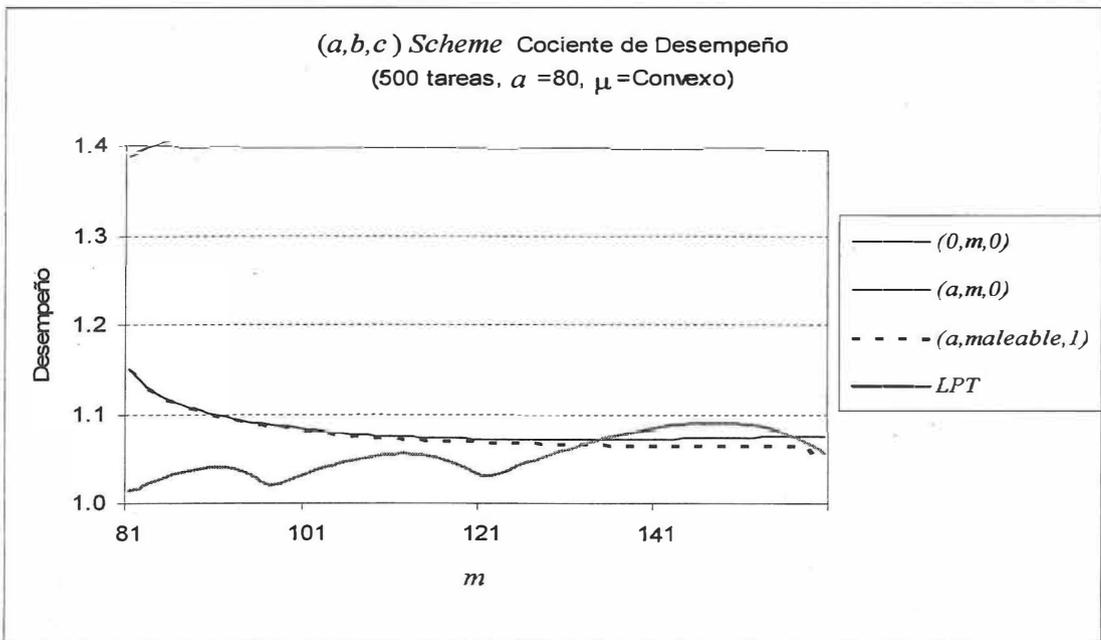


Figura 42. Desempeño promedio de 100 experimentos de LPT y  $(a,b,c)$ -Scheme para  $a=80$  y diferentes valores de  $b$ .  $m$  varía de 81 a 160 procesadores, aplicando un factor de ineficiencia convexo.

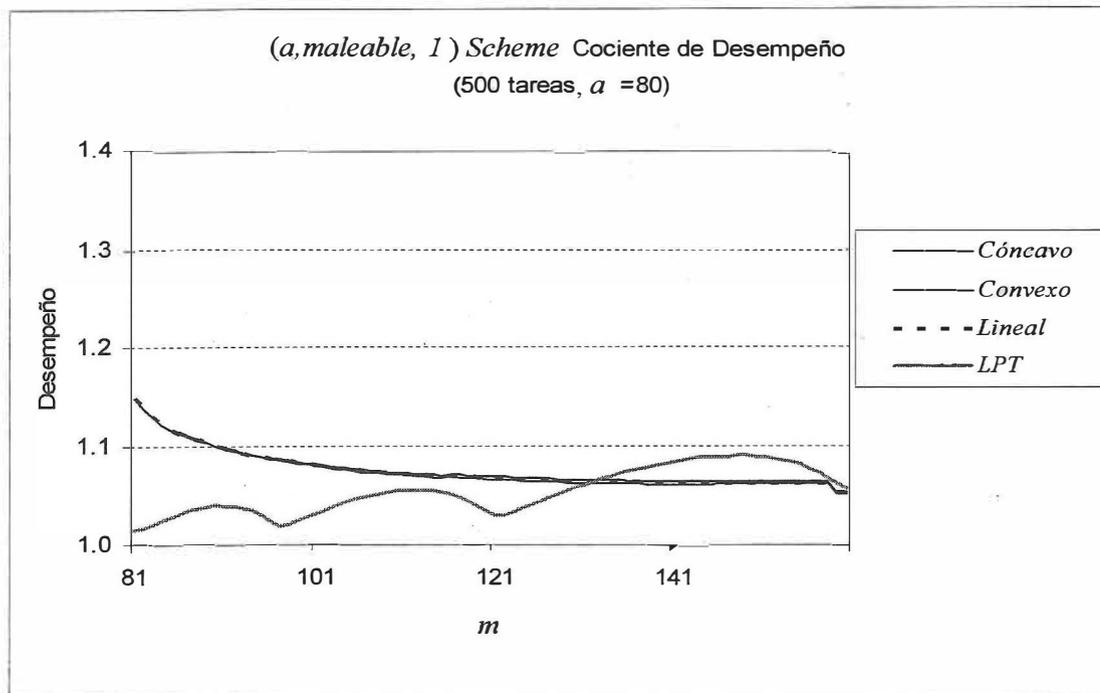


Figura 43. Desempeño promedio de 100 experimentos de  $(a, \text{maleable}, 1)$ -Scheme para  $a=80$ .  $m$  varía de 81 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal.

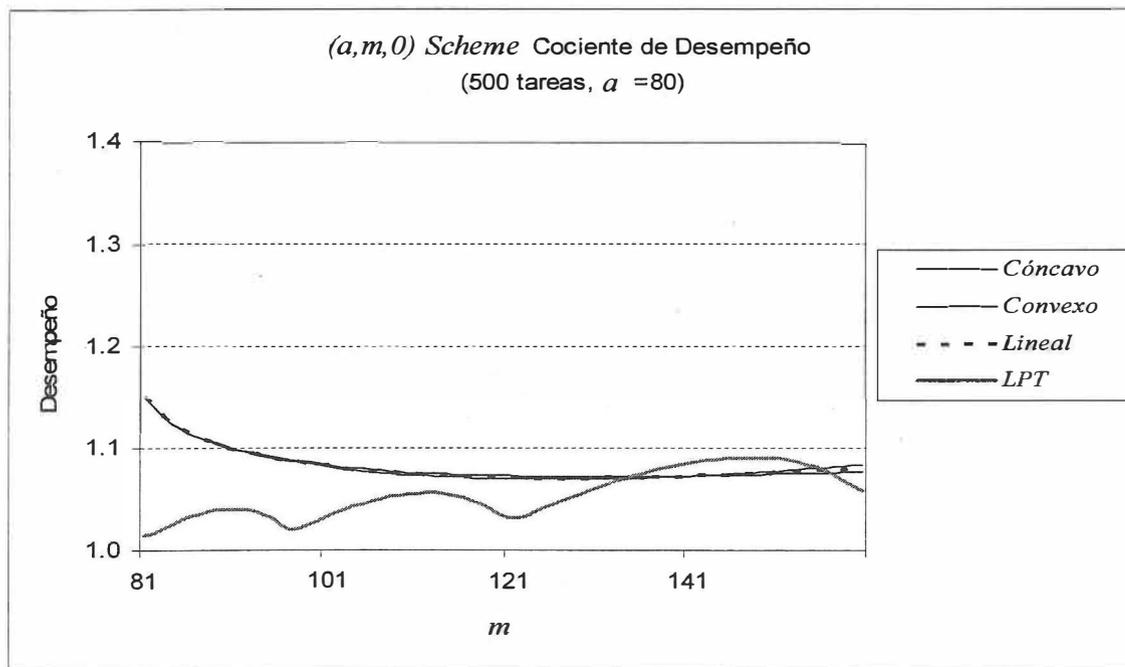


Figura 44. Desempeño promedio de 100 experimentos de  $(a, m, 0)$ -Scheme para  $a=80$ .  $m$  varía de 81 a 160 procesadores, aplicando los factores de ineficiencia cóncavo, convexo y lineal.

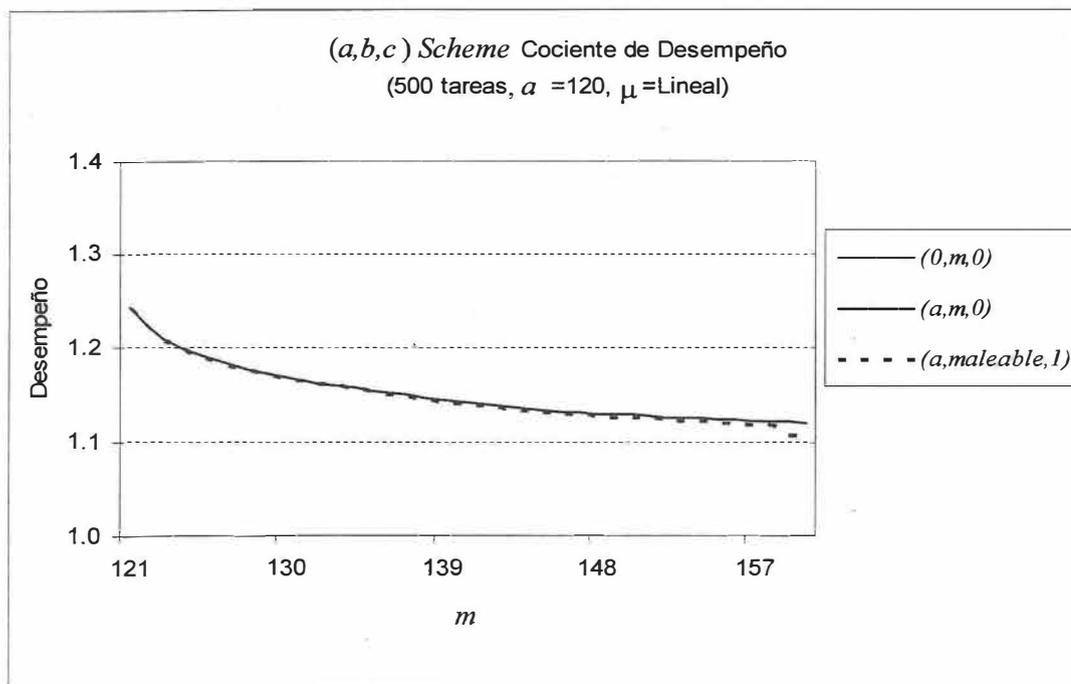


Figura 45. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a=120$  y diferentes valores de  $b$ .  $m$  varía de 121 a 160 procesadores, aplicando un factor de ineficiencia lineal.

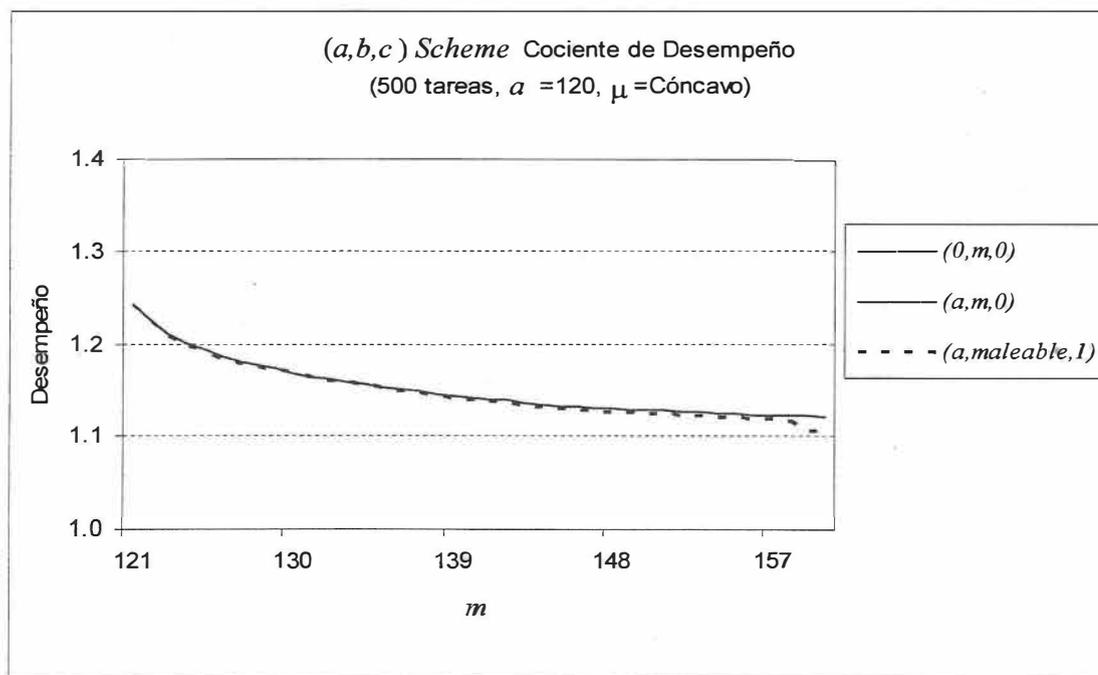


Figura 46. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a=120$  y diferentes valores de  $b$ .  $m$  varía de 121 a 160 procesadores, aplicando un factor de ineficiencia cóncavo.

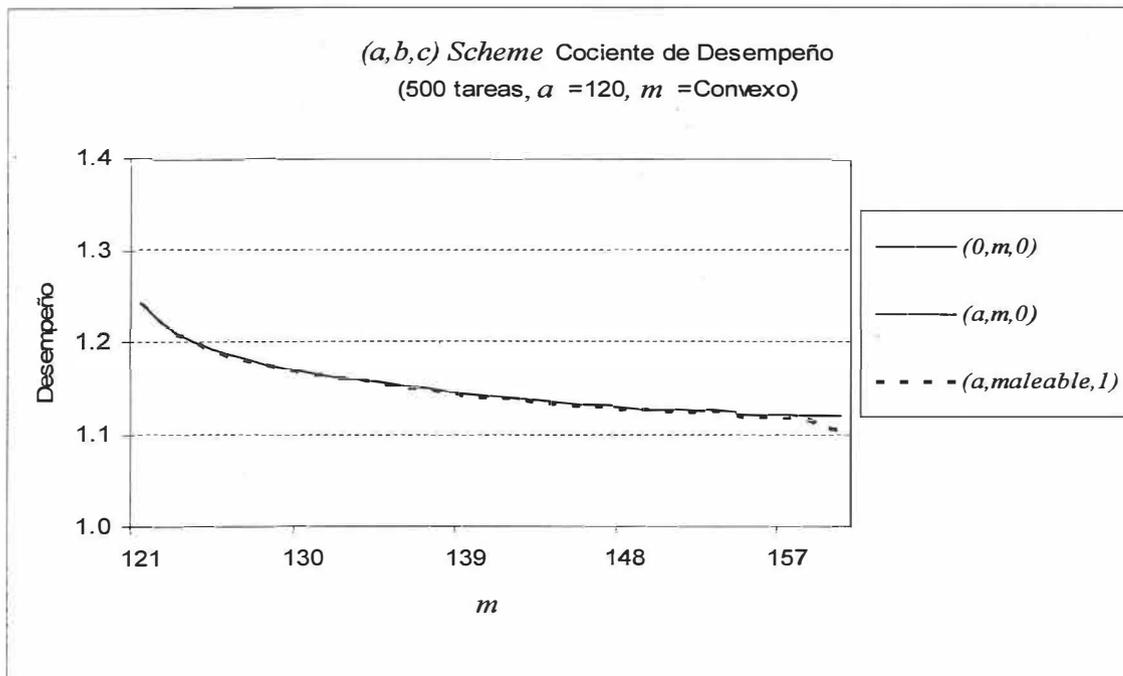


Figura 47. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme para  $a=120$  y diferentes valores de  $b$ .  $m$  varía de 121 a 160 procesadores, aplicando un factor de ineficiencia convexo.

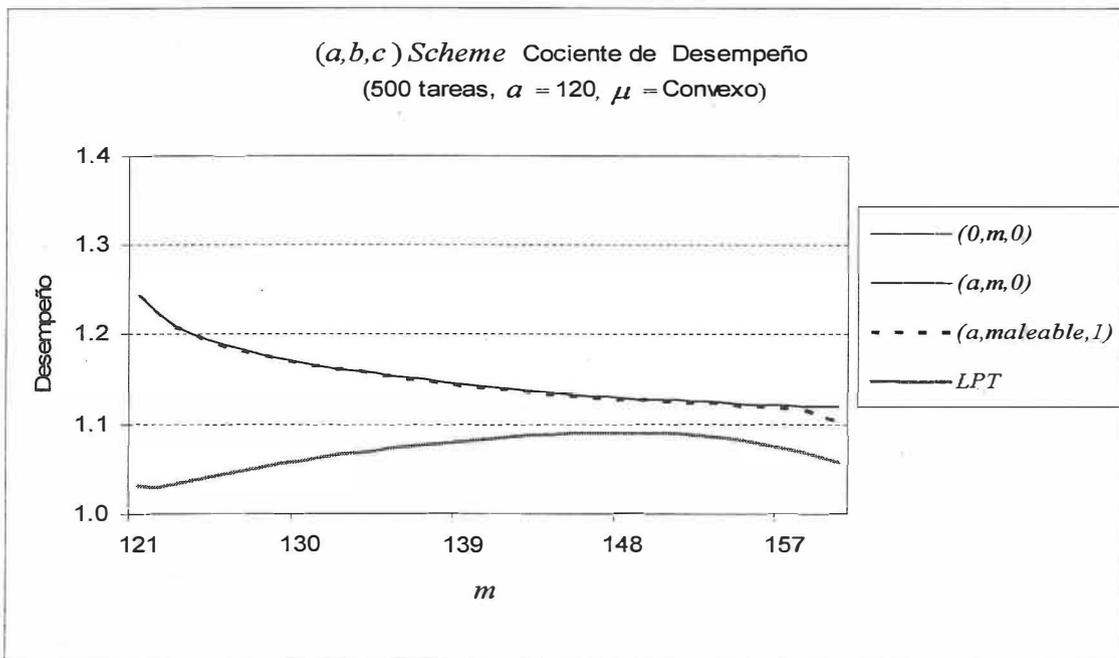


Figura 48. Desempeño promedio de 100 experimentos de LPT y  $(a,b,c)$ -Scheme para  $a=120$  y diferentes valores de  $b$ .  $m$  varía de 121 a 160 procesadores, aplicando un factor de ineficiencia convexo.

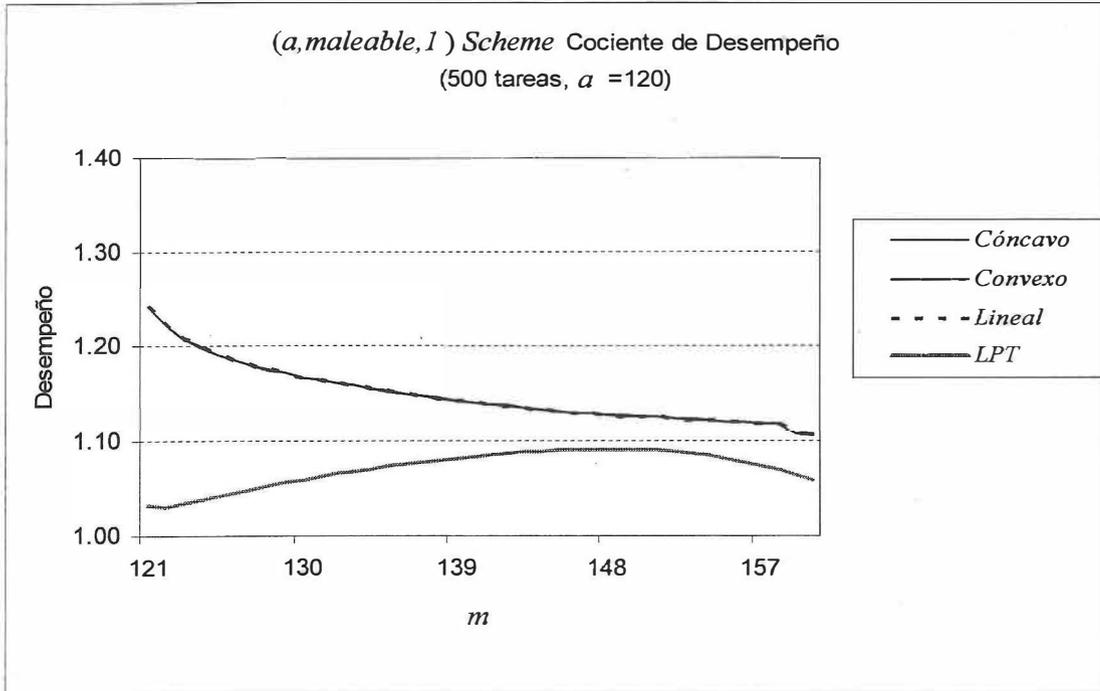


Figura 49. Desempeño promedio de 100 experimentos de  $(a, maleable, 1)$ -Scheme para  $a=120$ .  $m$  varía de 121 a 160 procesadores, aplicando los factores de ineficiencia lineal, cóncavo y convexo.

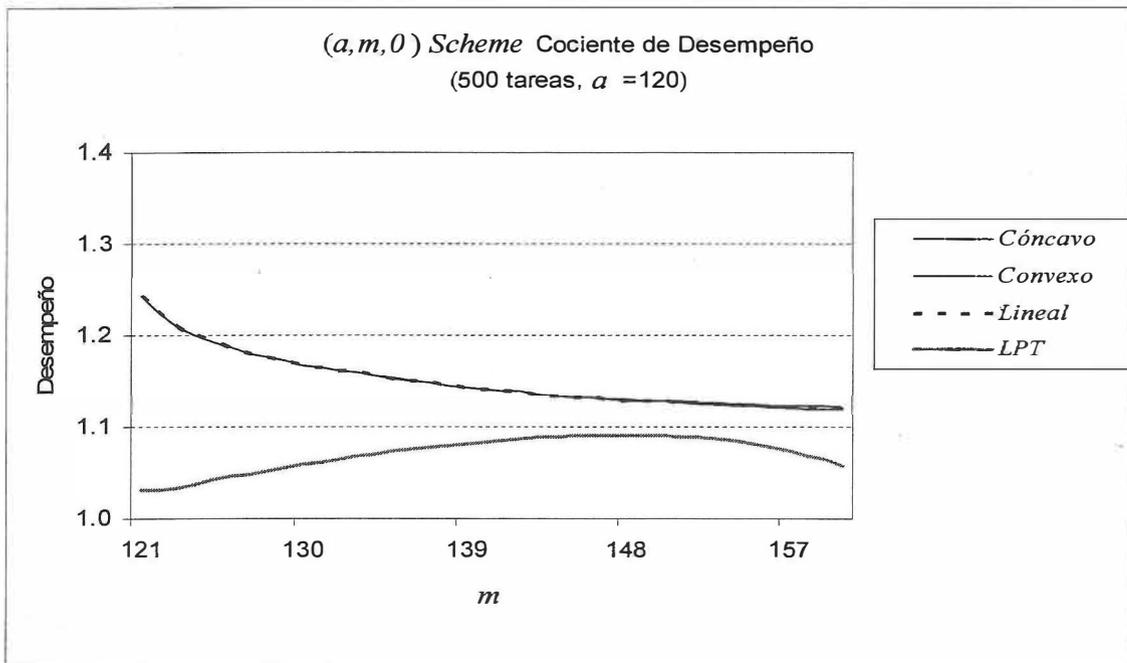


Figura 50. Desempeño promedio de 100 experimentos de  $(a, m, 0)$ -Scheme para  $a=120$ .  $m$  varía de 121 a 160 procesadores, aplicando los factores de ineficiencia lineal, cóncavo y convexo.

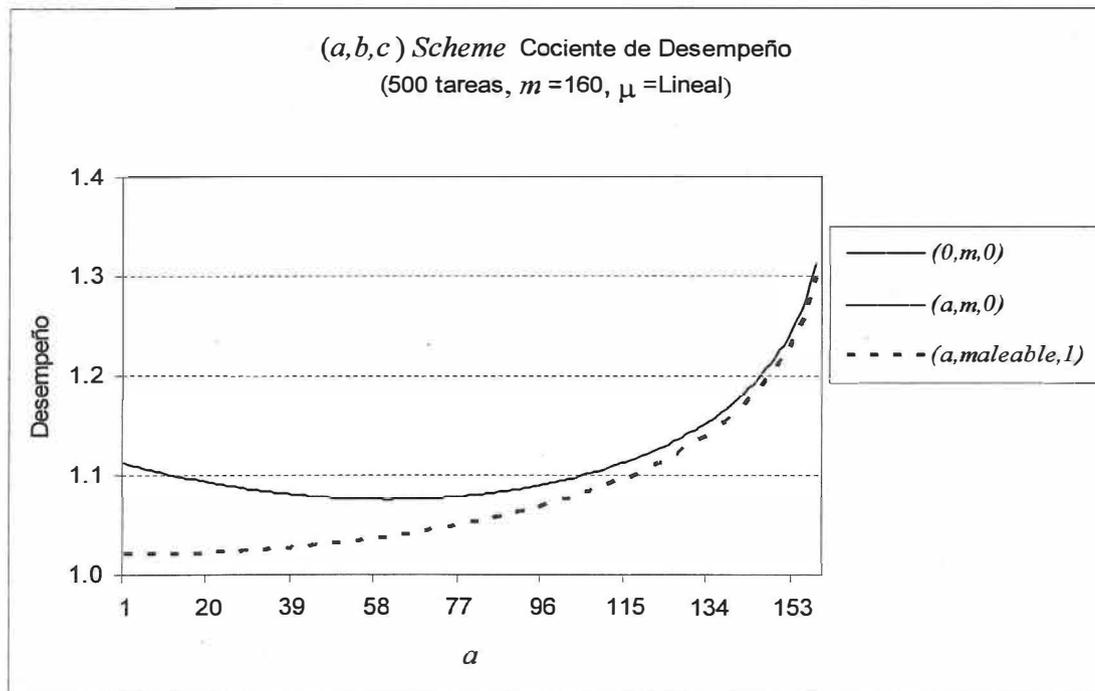


Figura 51. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme donde  $a$  varía de 1 a 159 con diferentes valores de  $b$ .  $m=160$ , aplicando un factor de ineficiencia lineal.

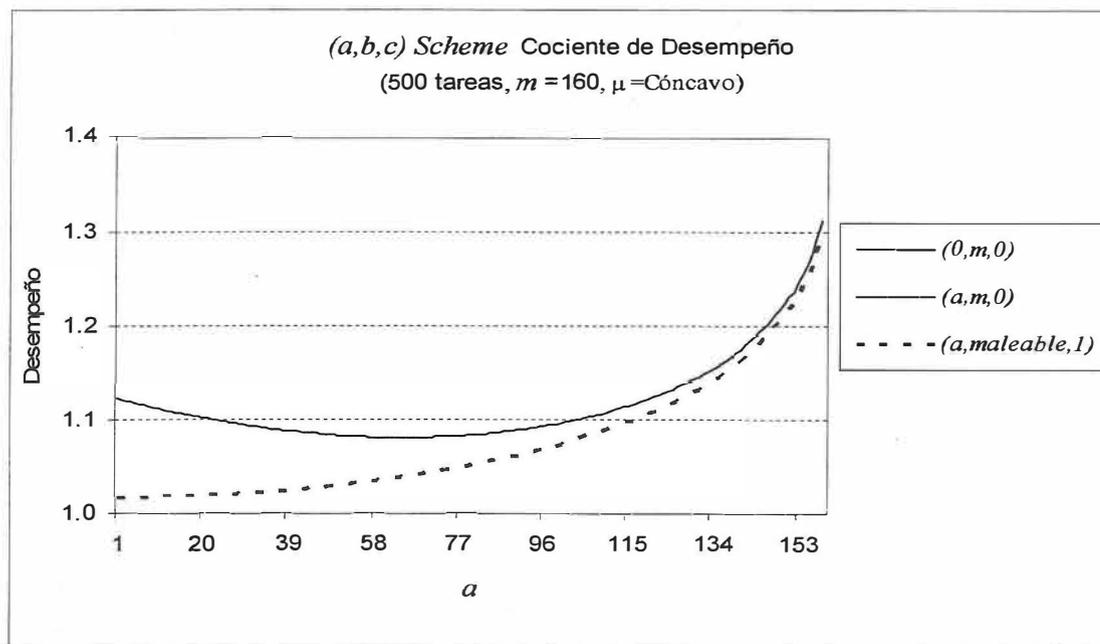


Figura 52. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme donde  $a$  varía de 1 a 159 con diferentes valores de  $b$ .  $m=160$ , aplicando un factor de ineficiencia cóncavo.

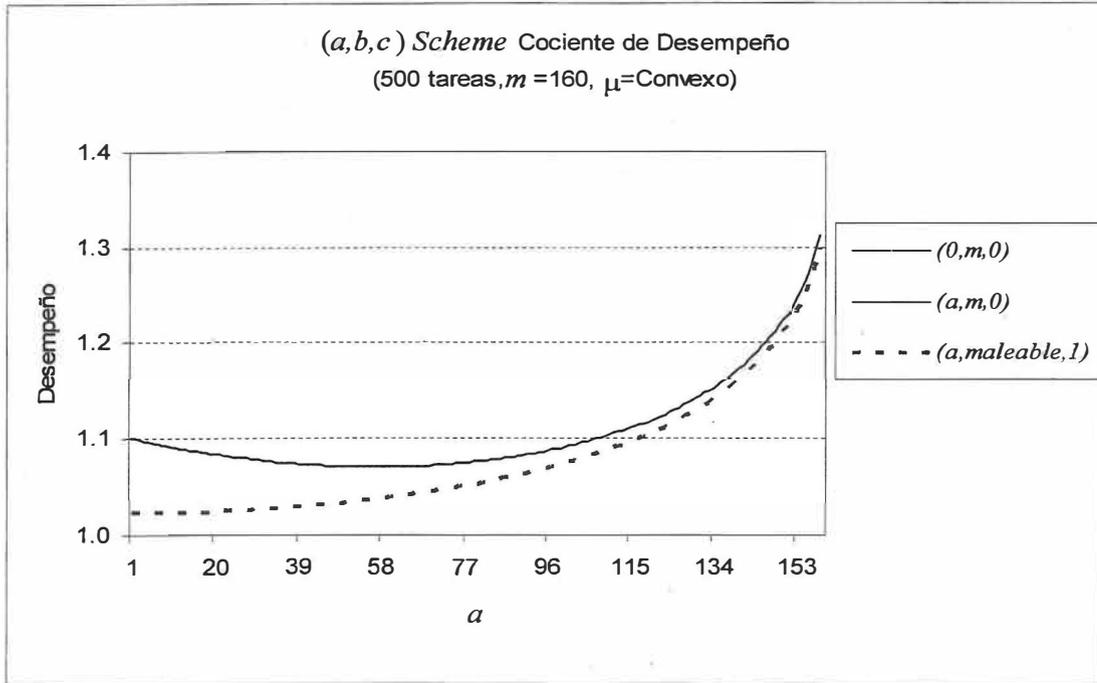


Figura 53. Desempeño promedio de 100 experimentos de  $(a,b,c)$ -Scheme donde  $a$  varía de 1 a 159 con diferentes valores de  $b$ .  $m=160$ , aplicando un factor de ineficiencia convexo.

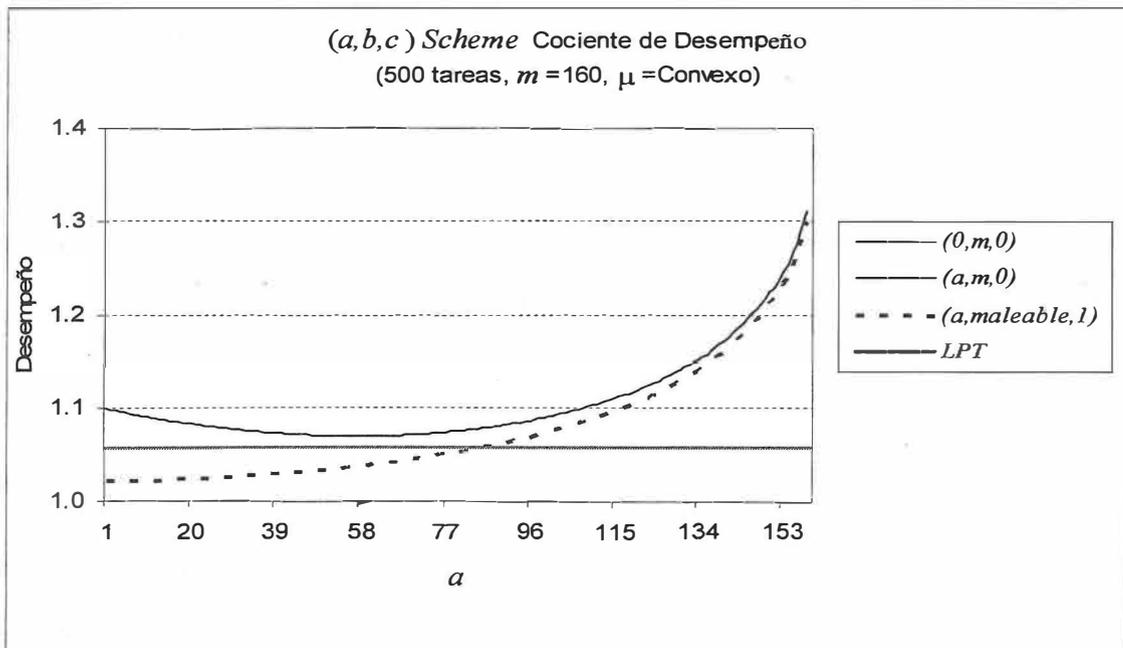


Figura 54. Desempeño promedio de 100 experimentos de LPT y  $(a,b,c)$ -Scheme donde  $a$  varía de 1 a 159 con diferentes valores de  $b$ .  $m=160$ , aplicando un factor de ineficiencia convexo.

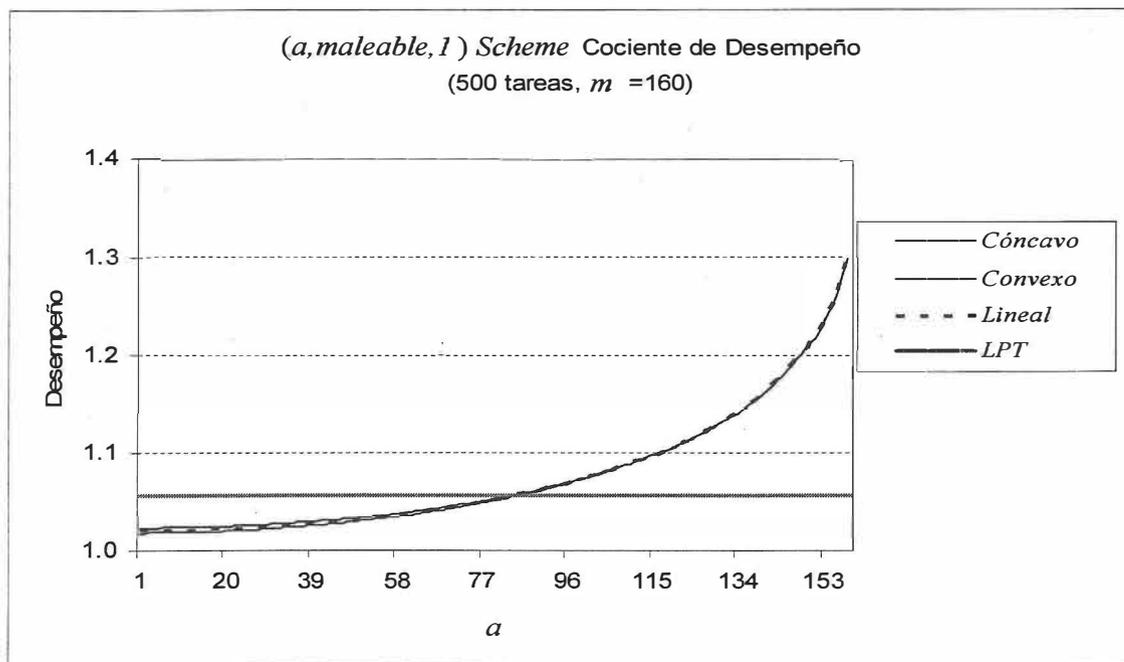


Figura 55. Desempeño promedio de 100 experimentos de  $(a, \text{maleable}, 1)$ -Scheme, donde  $a$  varía de 1 a 159 con diferentes valores de  $b$ .  $m=160$ , aplicando los factor de ineficiencia cóncavo, convexo y lineal.

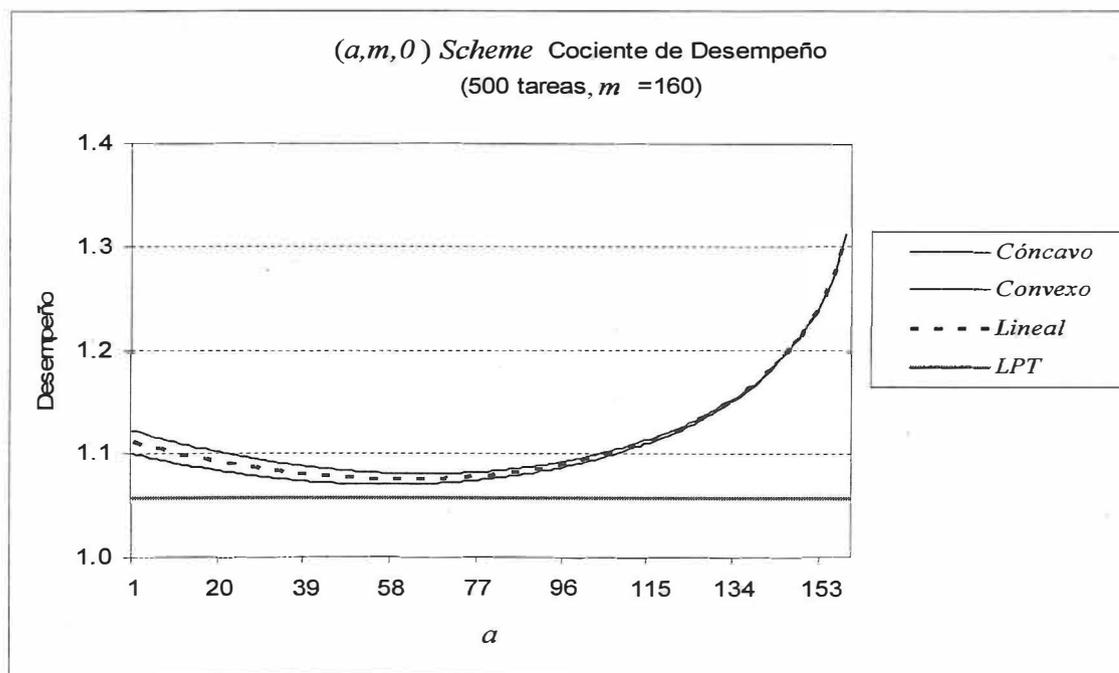


Figura 56. Desempeño promedio de 100 experimentos de  $(a, m, 0)$ -Scheme donde  $a$  varía de 1 a 159.  $m=160$ , aplicando los factor de ineficiencia cóncavo, convexo y lineal.

## Apéndice B. Herramienta *Scheduling*.

A continuación se describe el funcionamiento de la herramienta llamada *Scheduling* que se utilizó para realizar los experimentos.

Para llevar a cabo los experimentos relacionados a las estrategias descritas en este documento, es necesario introducir cierta información para crear los conjuntos de tareas e indicar que se va a trabajar con las estrategias analizadas en este documento. Lo anterior se debe a que la herramienta está diseñada para realizar una serie de estrategias distintas a las descritas en este documento, por tal motivo no se trabaja con todas las opciones que muestra la herramienta.

Iniciaremos nuestra descripción siguiendo un orden de izquierda a derecha y de arriba hacia abajo. Como se puede observar en la figura 57, la pantalla principal de nuestra herramienta está dividida en tres grandes secciones: *Parameters*, *Mallable* y *Results*. La primera sección se refiere a los datos necesarios para generar los conjuntos de tareas. La segunda sección se refiere a los tipos de estrategias que habrán de aplicarse al momento de realizar la calendarización, y por último tenemos la sección donde habrán de mostrarse los resultados de dichas estrategias.

Ahora bien, para indicar al sistema el número total de tareas que deberá contener cada conjunto de tareas, el usuario deberá introducir dicho número en el espacio denominado como *Tasks*.

A continuación tenemos *Time*; aquí el usuario puede elegir entre asignar a las tareas tiempos de procesamiento unitarios o no unitarios (en nuestro caso trabajaremos con tiempos no unitarios, los cuales se asignan de acuerdo a lo descrito en la sección 8.1).

En cuanto al tipo de conjunto de tareas que debe generarse, este se selecciona en *Graph*. Como recordaremos, nuestro interés se centra en conjunto de tareas independientes, los cuales se representa por la opción *Independ*.

Al momento de seleccionar conjuntos de tareas de tipo independiente, el sistema automáticamente despliega la opción *Malb*, que engloba todas las estrategias analizadas en este documento. Al activar *Malbl*, aparece la sección Maleable, que contiene las estrategias:

- *Graham*, cada una de las tareas se asignan a un solo procesador, de acuerdo al orden en que arribaron al sistema.
- *MLPT*, las tareas se ordenan de mayor a menor de acuerdo a su tiempo de procesamiento y se asignan cada una de éstas a un solo procesador. Esta opción hace referencia a la estrategia llamada *LPT*.
- *Gang*, desde un inicio cada una de las tareas se asigna al número total de procesadores en el sistema, de acuerdo al orden en que arribaron al sistema.
- *(a,b)-Scheme*, se refiere a la estrategia *(a,b,c)-Scheme*, donde la calendarización de las tareas se lleva a cabo por medio de dos pasos.

Cada una de estas estrategias se puede seleccionar independientemente.

En caso de seleccionar las estrategias *Gang* o *A-b Scheme*, es necesario indicar el tipo de factor de ineficiencia que habrá de aplicarse al paralelizar las tareas. Como se definió en la sección 8.2, existen tres tipos de funciones que se pueden aplicar como factor de ineficiencia: lineal (li), cóncava (cc) y convexa (cv). Para los experimentos el usuario debe

indicar con cuáles de éstas desea trabajar. En caso de seleccionar más de una función, el sistema realizará los experimentos correspondientes aplicando cada función.

Al seleccionar la estrategia  $(a,b)$ -Scheme, el usuario deberá proporcionar la información solicitada en *Beta* y *Alfa*. Ahora bien, *Beta* corresponde a la forma como habrán de asignarse los procesadores a cada tarea una vez que se haga el cambio de estrategia, momento indicado por *Alfa*. En *Beta* el usuario puede seleccionar entre:

- *single*, asignar un solo procesador a cada tarea.
- *total*, asignar el número total de procesadores a cada tarea.
- *malbl*, asignar los procesadores libres a una tarea.

Únicamente se puede seleccionar una opción de *Beta* a la vez.

En lo que respecta a *Alfa*, existen dos maneras de indicar al sistema el número de procesadores ociosos que debe existir para cambiar de estrategia. Obsérvese que justo debajo de la palabra *Alfa* se tienen dos cuadros, en el de la izquierda el usuario puede introducir directamente el número de procesadores ociosos y en el de la derecha el usuario indica el porcentaje de procesadores que habrán de estar ociosos para hacer cambio de estrategia. Estas dos opciones no se pueden combinar. Si el usuario desea hacer todos los experimentos variando *Alfa* de 1 hasta el número total de procesadores, sólo deberá introducir en el cuadro izquierdo el valor numérico cero como lo indica la figura 57 y el sistema automáticamente realizará los experimentos.

Un dato importante que el usuario debe proporcionar antes de iniciar los experimentos corresponde al número total de procesadores en el sistema, éste se introduce en *Proc*. Aquí el usuario introduce un número específico, o bien puede introducir el valor numérico de

cero, para indicar al sistema que realice todos los experimentos considerando que el número total de procesadores en el sistema varía de 1 hasta el número total de tareas menos 1.

La herramienta *Scheduling*, cuenta con la opción que permite al usuario seleccionar una o varias estrategias a ejecutarse y observar su resultado, con la posibilidad de poder aplicar las veces que él decida dichas estrategias o bien aplicar estrategias distintas sin cambiar de conjunto de tareas. Para poder utilizar dicha opción e indicar al sistema que no cambie de conjunto de tareas, el usuario una vez que genere el conjunto de tareas con el que desea trabajar deberá desactivar el botón que se encuentra en la parte superior derecha de la pantalla (ver figura 57), justo debajo de *Run/Stop/Save*. Cuando este botón está activado indica que se debe generar un nuevo conjunto de tareas, de lo contrario continúa trabajando con el mismo conjunto de tareas.

Después de haber introducido todos los datos necesarios para generar el conjunto de tareas, además de haber especificado las estrategias a aplicarse durante la calendarización, el sistema está listo para generar el conjunto de tareas e iniciar su calendarización; para esto es necesario presionar el botón llamado *Experim*. Una vez que el sistema genera el conjunto de tareas, sus datos se despliegan en la parte inferior de la pantalla (véase figura 57). Los datos desplegados son: número de tarea, capa, nivel, co-nivel, tiempo de procesamiento de la tarea y tipo de  $b$  asignado a la tarea, las dos últimas columnas no muestran ningún dato por tratarse de un conjunto de tareas independientes.

Graph Statistics: Arbitrary\_NonUnitTime\_Tasks\_100\_Exp

Exp	Heu	Proc	Time1	Time2
1	2	15	5:46:04 PM	5:46:04 PM

Experiment	Layer	Edge	Num_nrc	Leaf	MaxSucc	MaxWide	Layer_MaxWide	MaxPath
1	1	0	100	100	0	100	1	0
MIN	1	0	100	100	0	100	1	0

N	Layer	Level	CoLevel	Time	BetaT	[Ancestors]-List	[Successors]-List
1.	1	90	0	90	malbl	[0]=	[0]=
2.	1	97	0	97	malbl	[0]=	[0]=
3.	1	46	0	46	malbl	[0]=	[0]=
4.	1	87	0	87	malbl	[0]=	[0]=
5.	1	84	0	84	malbl	[0]=	[0]=
6.	1	83	0	83	malbl	[0]=	[0]=
7.	1	8	0	8	malbl	[0]=	[0]=
8.	1	4	0	4	malbl	[0]=	[0]=
9.	1	32	0	32	malbl	[0]=	[0]=
10.	1	80	0	80	malbl	[0]=	[0]=
11.	1	76	0	76	malbl	[0]=	[0]=
12.	1	114	0	114	malbl	[0]=	[0]=
13.	1	47	0	47	malbl	[0]=	[0]=
14.	1	97	0	97	malbl	[0]=	[0]=
15.	1	72	0	72	malbl	[0]=	[0]=
16.	1	40	0	40	malbl	[0]=	[0]=
17.	1	49	0	49	malbl	[0]=	[0]=
18.	1	122	0	122	malbl	[0]=	[0]=
19.	1	80	0	80	malbl	[0]=	[0]=
20.	1	95	0	95	malbl	[0]=	[0]=
21.	1	73	0	73	malbl	[0]=	[0]=
22.	1	67	0	67	malbl	[0]=	[0]=
23.	1	103	0	103	malbl	[0]=	[0]=
24.	1	21	0	21	malbl	[0]=	[0]=
25.	1	83	0	83	malbl	[0]=	[0]=
26.	1	82	0	82	malbl	[0]=	[0]=
27.	1	48	0	48	malbl	[0]=	[0]=
28.	1	88	0	88	malbl	[0]=	[0]=
29.	1	73	0	73	malbl	[0]=	[0]=
30.	1	96	0	96	malbl	[0]=	[0]=
31.	1	82	0	82	malbl	[0]=	[0]=
32.	1	36	0	36	malbl	[0]=	[0]=
33.	1	54	0	54	malbl	[0]=	[0]=
34.	1	27	0	27	malbl	[0]=	[0]=

Figura 57. Pantalla principal de la herramienta *Scheduling*.

Es necesario que el sistema primeramente termine de mostrar todos los datos del conjunto de tareas, para poder solicitar al sistema que muestre los resultados de las estrategias seleccionadas. Una vez que termine de mostrar los datos, se deberá presionar el botón *Mallable* para desplegar los resultados. Como se puede observar en la figura 58, los resultados están distribuidos a través de cinco columnas:

- *Malbl\_Experim*, indica el tipo de estrategia, total de tareas en el conjunto, número de procesadores en el sistema, tipo de factor de ineficiencia aplicado, valor de  $a$  y tipo de  $b$ .
- *Idle*, indica el número total de veces que estuvieron ociosos los procesadores durante todo el proceso de calendarización.

- *Expl\_Copt*, muestra el tiempo óptimo de procesamiento de las tareas aplicando una determinada estrategia.
- *Expl\_Ca*, muestra el tiempo total de procesamiento de las tareas aplicando determinada estrategia bajo las características especificadas en *Malbl\_Experim*.
- *Expl\_R*, muestra el cociente de desempeño entre lo mostrado por *Expl\_Ca* y *Expl\_Copt*.

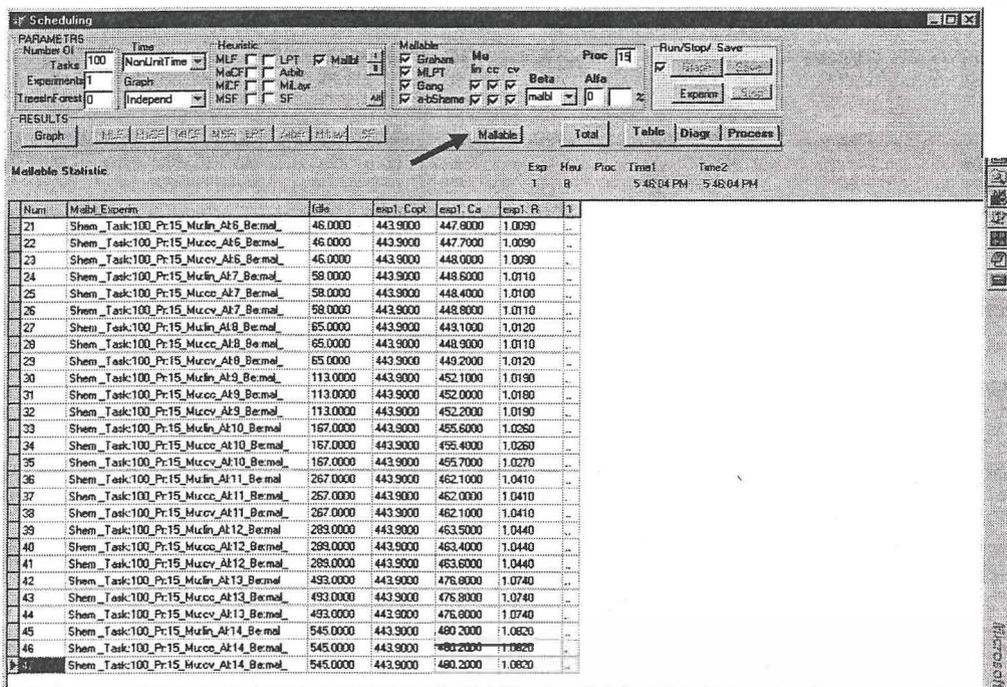


Figura 58. Pantalla con resultados de la calendarización.

Si el usuario desea almacenar los datos del conjunto de tareas y los resultados de la calendarización, es necesario presionar en la parte superior derecha de la pantalla el botón que se encuentra señalado con un círculo en la figura 59.

PARAMETERS

Number Of: Tasks 100 Time: NonJrnTime

Experiments: 1 Graph

TreeInFormat: 0 Independ

HEURISTIC

MLF  LPT  Maltbi

MaCF  ABBR

MCF  MILAY

MSF  SF

Malleable

Ma  In  Co  cv

Beta  Alfa

multi  %

Run/Stop/ Save

Save Save

Experim Stop

RESULTS

Graph MLF MaCF MSF MCF LPT Subt Maltbi SF

Malleable Total Table Diag Process

Malleable Statistic

Num	Maltbi_Experim	fide	exp1_Eopt	exp1_Ca	exp1_R	t
21	Shem_Task100_Pr15_Mulin_Alt6_Bermal	46.0000	443.9000	447.8000	1.0090	..
22	Shem_Task100_Pr15_Mucoc_Alt6_Bermal	46.0000	443.9000	447.7000	1.0090	..
23	Shem_Task100_Pr15_Mucov_Alt6_Bermal	46.0000	443.9000	448.0000	1.0090	..
24	Shem_Task100_Pr15_Mulin_Alt7_Bermal	58.0000	443.9000	448.6000	1.0110	..
25	Shem_Task100_Pr15_Mucoc_Alt7_Bermal	58.0000	443.9000	448.4000	1.0100	..
26	Shem_Task100_Pr15_Mucov_Alt7_Bermal	58.0000	443.9000	448.8000	1.0110	..
27	Shem_Task100_Pr15_Mulin_Alt8_Bermal	65.0000	443.9000	449.1000	1.0120	..
28	Shem_Task100_Pr15_Mucoc_Alt8_Bermal	65.0000	443.9000	449.3000	1.0110	..
29	Shem_Task100_Pr15_Mucov_Alt8_Bermal	65.0000	443.9000	449.2000	1.0120	..
30	Shem_Task100_Pr15_Mulin_Alt9_Bermal	113.0000	443.9000	452.1000	1.0190	..
31	Shem_Task100_Pr15_Mucoc_Alt9_Bermal	113.0000	443.9000	452.0000	1.0190	..
32	Shem_Task100_Pr15_Mucov_Alt9_Bermal	113.0000	443.9000	452.2000	1.0190	..
33	Shem_Task100_Pr15_Mulin_Alt10_Bermal	167.0000	443.9000	456.6000	1.0260	..
34	Shem_Task100_Pr15_Mucoc_Alt10_Bermal	167.0000	443.9000	456.4000	1.0260	..
35	Shem_Task100_Pr15_Mucov_Alt10_Bermal	167.0000	443.9000	456.7000	1.0270	..
36	Shem_Task100_Pr15_Mulin_Alt11_Bermal	257.0000	443.9000	462.7000	1.0410	..
37	Shem_Task100_Pr15_Mucoc_Alt11_Bermal	7.0000	443.9000		1.0410	..
38	Shem_Task100_Pr15_Mucov_Alt11_Bermal	257.0000	443.9000	462.1000	1.0410	..
39	Shem_Task100_Pr15_Mulin_Alt12_Bermal	289.0000	443.9000	463.5000	1.0440	..
40	Shem_Task100_Pr15_Mucoc_Alt12_Bermal	289.0000	443.9000	463.4000	1.0440	..
41	Shem_Task100_Pr15_Mucov_Alt12_Bermal	289.0000	443.9000	463.6000	1.0440	..
42	Shem_Task100_Pr15_Mulin_Alt13_Bermal	493.0000	443.9000	476.9000	1.0740	..
43	Shem_Task100_Pr15_Mucoc_Alt13_Bermal	493.0000	443.9000	476.9000	1.0740	..
44	Shem_Task100_Pr15_Mucov_Alt13_Bermal	493.0000	443.9000	476.9000	1.0740	..
45	Shem_Task100_Pr15_Mulin_Alt14_Bermal	545.0000	443.9000	480.2000	1.0820	..
46	Shem_Task100_Pr15_Mucoc_Alt14_Bermal	545.0000	443.9000	480.2000	1.0820	..
47	Shem_Task100_Pr15_Mucov_Alt14_Bermal	545.0000	443.9000	480.2000	1.0820	..

Figura 59. Activación de ventanas para almacenar datos o salir del sistema.

Al presionar este botón automáticamente aparece una ventana llamada *Confirm*, que permite almacenar los resultados de la calendarización y del conjunto de tareas (ver figura 60).

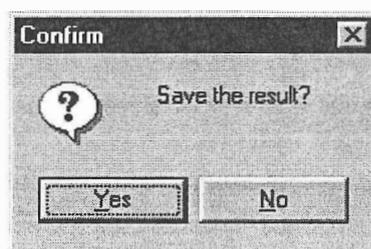


Figura 60. Ventana para almacenar los datos del conjunto de tareas.

Una vez que se indica si se desea o no almacenar la información, el sistema despliega una segunda ventana encargada de preguntar si se desea o no dar por terminada la ejecución (ver figura 51).

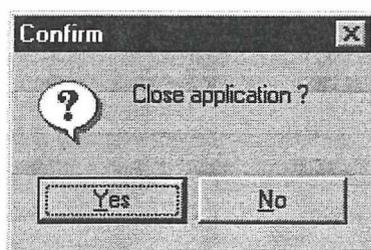


Figura 61. Ventana para terminar la ejecución de la herramienta.

## **Apéndice C. Computadora paralela cicese2000.**

En 1996, CICESE adquirió un servidor tipo gabinete ORIGIN2000 de Silicon Graphics, que se utiliza para el desarrollo de aplicaciones e investigaciones de procesamiento en paralelo y de compilación de programas que requieran velocidad en cómputo, gran cantidad de memoria y de una respuesta inmediata de resultados.

Antes de mostrar el uso de CPU de la computadora paralela *cicese2000*, durante un periodo de 6 meses que comprende de abril a septiembre del 2002, primeramente se describen las características de dicho servidor. Hablaremos de su sistema operativo, gabinete, procesadores, memoria, escalabilidad, medios de almacenamiento, tarjeta de base de E/S.

### **Sistema Operativo**

ORIGIN2000 cuenta con el sistema operativo Irix 6.4 (exclusivo de Silicon Graphics):

- Está basado en UNIX System V, Release 4.
- Propio para arquitecturas de 64 bits (Origin2000).
- Se pueden ejecutar aplicaciones de versiones anteriores (de Irix 5 y 6).
- Soporta escalabilidad de hasta 32 procesadores.

### **Sistema de archivo**

El sistema de archivos es XFS, cuyas algunas características son:

- Es un sistema de archivos de 64 bits.
- Soporta bloques lógicos de 512 bytes hasta 64 KB.
- Presenta avanzada tecnología de autorecuperación cuando falla un sistema de archivos.

- Tiene un promedio de lectura-escritura al disco de más de 500 MB por segundo.

### **Gabinete**

El gabinete que está formado por 2 *módulos* manejados cada uno por su propio Sistema Controlador de Módulos (MSC), y un Panel de Control llamado Sistema Controlador Multimódulos (MMSC) que maneja y controla ambos *módulos*.

Un *módulo* contiene: 4 nodos IP27, 1 Midplane, 2 ruteadores, 11 ranuras para tarjetas de E/S, 1 tarjeta base de E/S, 1 Controlador de Módulos (MSC) con 5 ranuras para discos SCSI.

La comunicación entre los *módulos* se lleva por medio de un cable de alta velocidad llamado CrayLink Interconnect cuya velocidad es de 800 MBs/seg.

Un *nodo* esta formador por: 2 procesadores R10000, 16 ranuras de Memoria RAM y 1 Hub.

### **Procesadores**

En ORIGIN2000 cada *módulo* está comprendido por 4 *nodos* y cada *nodo* cuenta con 2 procesadores.

cicese2000 cuenta con 10 procesadores R10000 de 195 MHZ. El módulo 1 posee 1 nodo con 2 procesadores y el módulo 2 tiene 4 nodos con 8 procesadores totalizando 10 procesadores en los 5 nodos.

Cada procesador tiene la capacidad máxima de efectuar 390 millones de operaciones de punto flotante por segundo, totalizando 3.90 GFlops en la supercomputadora cicese2000

### **Memoria**

cicese2000 cuenta con una memoria principal de 1280 MB (1.25 GB) de memoria física, distribuida entre los 5 nodos (cada nodo con 256 MB) y se comparte lógicamente, se

conoce como Memoria Compartida Distribuida (DSM). Para un procesador la memoria aparece como un sencillo espacio direccionable de 1280 MB. Esta memoria es compartida para todos los procesos de los usuarios.

Cada procesador tiene 64 KB de caché primario (32 KB de instrucciones y 32 KB de datos) y 4 MB de caché secundario.

### **Escalabilidad**

Un solo nodo puede tener de 64 MB hasta 4 GB de memoria principal, lo que significa que un gabinete con sus 8 nodos puede alcanzar 32 GB de memoria principal, y si juntamos 8 gabinetes se tendrán 256.

El gabinete tiene capacidad de hasta 16 procesadores y su máximo crecimiento es de 8 gabinetes totalizando 128 procesadores.

### **Medios de almacenamiento**

Son 5 discos SCSI cuya capacidad de almacenamiento suman 40.9 GB., distribuidos en los 2 módulos. Es decir, se tiene un disco principal del sistema de 4.5 GB y 4 discos más de 9.1 GB.

Se cuenta con una Unidad de Cinta (DAT) para cintas de 4 mm. y con una Unidad de CDROM.

Cada módulo posee 5 ranuras para discos SCSI.

### **Tarjeta base de E/S**

Esta tarjeta se localiza en la primera ranura de E/S en la parte posterior del módulo.

Cicese2000 cuenta con 2 Tarjeta Base E/S (una x módulo). La tarjeta base de E/S que viene con cada módulo contiene lo siguiente: 1 Conector Fast Ethernet 100 Base-T, 2 Puertos serial de 9 pins, 1 Puerto externo UltraSCSI (68 pins), 2 Interruptores de tiempo real.

Se tiene disponible 11 ranuras de E/S en cada módulo para agregar tarjetas de red, UltraSCSI, Canal de Fibra, ATM, HIPPI-Serial.

*cicese2000* cuenta con una tarjeta de red 10/100 Base T Fast Ethernet integrada en la tarjeta Base de E/S.

La información presentada anteriormente fue recopilada de la dirección de internet:

<http://telematica.cicese.mx/computo/super/cicese2000/>

Una vez descritas las características de *cicese2000*, a continuación mostraremos gráficamente el uso promedio de CPU mostrado durante el periodo de abril a septiembre del 2002. Es importante mencionar que hasta el momento *cicese2000* cuenta con aproximadamente 154 usuarios internos y 7 externos.

De acuerdo a la información proporcionados por la persona encargada del manejo de *cicese2000*, Julián Delgado, del Departamento de Telemática de CICESE, el uso promedio diario de CPU es almacenado en un archivo mensual de nombre “rpt.xxx”, donde las x’s corresponden a las tres primeras letras de cada mes. A continuación mostraremos los datos almacenados de abril a septiembre del 2002.

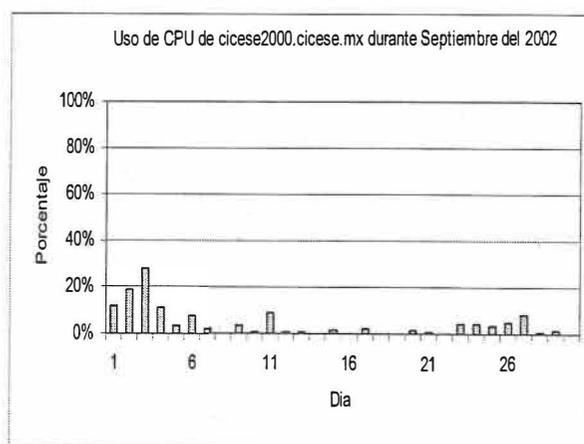
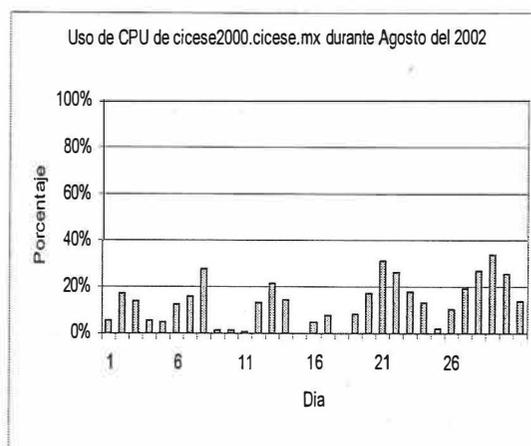
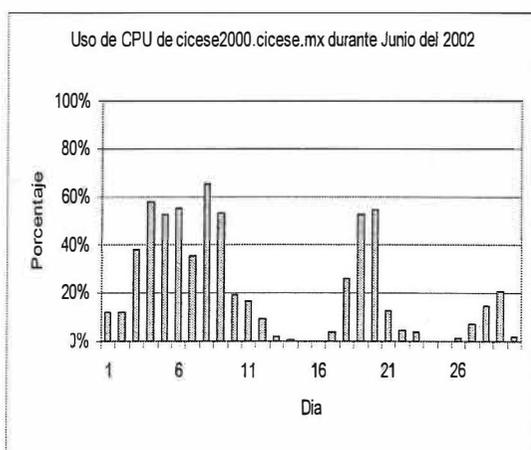
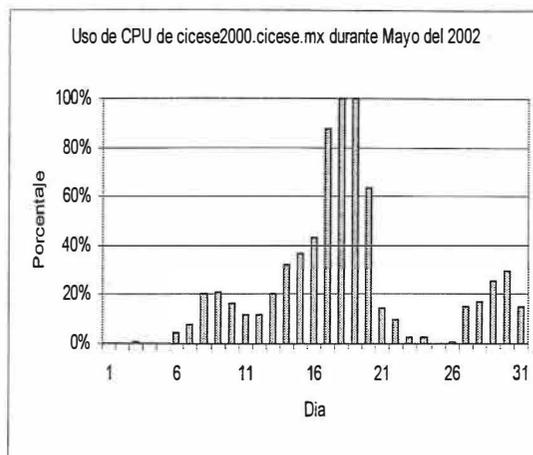
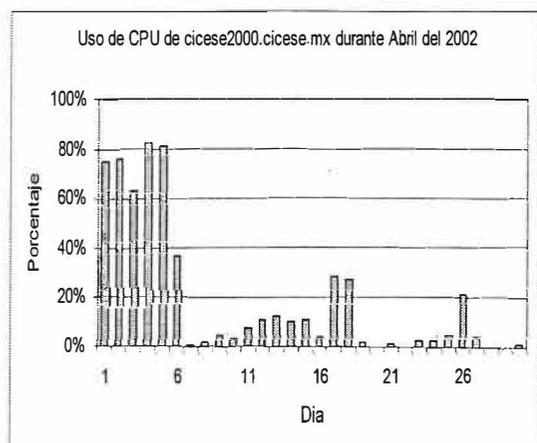


Figura 62. Uso diario de CPU de *cicese2000* de abril a septiembre del 2002.

La siguiente figura nos muestra el uso mensual promedio de CPU de *cicese2000* durante un periodo de 6 meses.

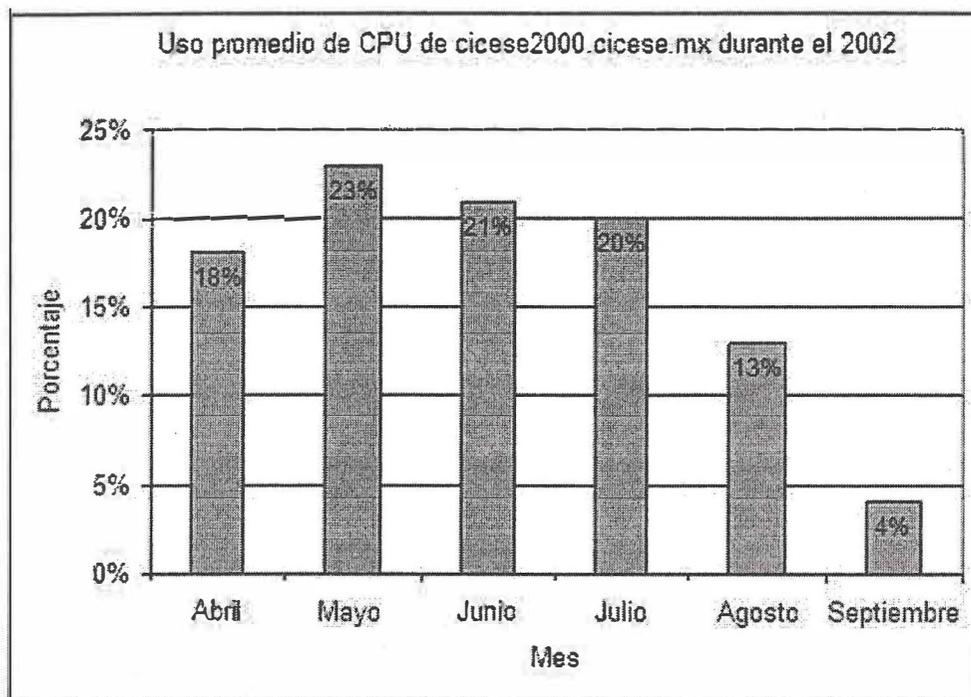


Figura 63. Uso mensual de CPU de *cicese2000* de abril a septiembre del 2002.