CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

TESIS

PLANIFICACIÓN Y ASIGNACIÓN DE TAREAS EN UN SISTEMA DE MULTICOMPUTADORAS USANDO ALGORITMOS EVOLUTIVOS MULTIOBJETIVO

PRESENTA

Apolinar Velarde Martinez

PARA OBTENER EL GRADO DE DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN

TUTOR

Dra. Eunice Esther Ponce de León Sentí

COMITÉ TUTORAL

Dra. Elva Díaz Díaz
Dr. Alejandro Padilla Díaz

Aguascalientes, Ags. 13 de Enero de 2014

UNIVERSIDAD AUTONOMA
DE AGUASCALIENTES

M. en C. José de Jesús Ruiz Gallegos
Decano del Centro de Ciencias Básicas

PRESENTE

Por medio del presente como tutor designado del estudiante APOLINAR VELARDE MARTINEZ con ID 125686 quien realizó el trabajo de tesis titulado PLANIFICACION Y ASIGNACION DE TAREAS EN UN SISTEMA DE MULTICOMPUTADORAS USANDO ALGORITMOS EVOLUTIVOS MULTIOBJETIVO, y con fundamento en el Articulo 175, Apartado II del Reglamento General de Docencia, me permito emitir el VOTO APROBATORIO, para que él pueda proceder a imprimirlo, y así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

ATENTAMENTE
"Se Lumen Proferre"
Aguascalientes, Ags., a 10 de diciembre de 2013

Dra. Eunice Esther Ponce de León Sentí
Directora de Tesis

ccp.- Interesado
ccp.- Secretaria de Investigación y Posgrado
ccp.- Jefatura del departamento de Ciencias de la Computación
ccp.- Consejero Académico

UNIVERSIDAD AUTONOMA
DE AGUASCALIENTES

M. en C. José de Jesús Ruiz Gallegos
Decano del Centro de Ciencias Básicas

PRESENTE

Por medio del presente como tutor designado del estudiante APOLINAR VELARDE MARTINEZ con ID 125686 quien realizó el trabajo de tesis titulado PLANIFICACION Y ASIGNACION DE TAREAS EN UN SISTEMA DE MULTICOMPUTADORAS USANDO ALGORITMOS EVOLUTIVOS MULTIOBJETIVO, y con fundamento en el Articulo 175, Apartado II del Reglamento General de Docencia, me permito emitir el VOTO APROBATORIO, para que él pueda proceder a imprimirlo, y así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

ATENTAMENTE
"Se Lumen Proferre"
Aguascalientes, Ags., a 10 de diciembre de 2013

Dra. Elva Díaz Díaz
Asesor de Tesis

ccp.- Interesado
ccp.- Secretaria de Investigación y Posgrado
ccp.- Jefatura del departamento de Ciencias de la Computación
ccp.- Consejero Académico

UNIVERSIDAD AUTONOMA
DE AGUASCALIENTES

M. en C. José de Jesús Ruiz Gallegos
Decano del Centro de Ciencias Básicas

PRESENTE

Por medio del presente como tutor designado del estudiante **APOLINAR VELARDE MARTINEZ** con ID **125686** quien realizó el trabajo de tesis titulado **PLANIFICACION Y ASIGNACION DE TAREAS EN UN SISTEMA DE MULTICOMPUTADORAS USANDO ALGORITMOS EVOLUTIVOS MULTIOBJETIVO**, y con fundamento en el Articulo 175, Apartado II del Reglamento General de Docencia, me permito emitir el VOTO APROBATORIO, para que él pueda proceder a imprimirlo, y así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

ATENTAMENTE
"Se Lumen Proferre"
Aguascalientes, Ags., a 10 de diciembre de 2013

Dr. Alejandro Padilla Díaz
Asesor de Tesis

ccp.- Interesado
ccp.- Secretaria de Investigación y Posgrado
ccp.- Jefatura del departamento de Ciencias de la Computación
ccp.- Consejero Académico

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

ANIVERSARIO UAA

Centro de Ciencias Básicas

M. en C. APOLINAR VELARDE MARTÍNEZ
ALUMNO (A) DEL DOCTORADO EN CIENCIAS EXACTAS, SISTEMAS Y DE LA INFORMACIÓN,
P R E S E N T E .

Estimado (a) alumno (a) Velarde:

Por medio de este conducto me permito comunicar a Usted que habiendo recibido los votos aprobatorios de los revisores de su trabajo de tesis y/o caso práctico titulado: **"PLANIFICIACIÓN Y ASIGNACIÓN DE TAREAS EN UN SISTEMA DE MULTICOMPUTADORAS USANDO ALGORITMOS EVOLUTIVOS MULTIOBJETIVO",** hago de su conocimiento que puede imprimir dicho documento y continuar con los trámites para la presentación de su examen de grado.

Sin otro particular me permito saludarle muy afectuosamente.

A T E N T A M E N T E
Aguascalientes, Ags., 11 de diciembre de 2013
"SE LUMEN PROFERRE"
EL DECANO SUSTITUTO

M. en C. JOSÉ DE JESÚS RUIZ GALLEGOS

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES
CENTRO DE CIENCIAS BÁSICAS

c.c.p.- Interesado
c.c.p.- Secretaría de Investigación y Posgrado
c.c.p.- Jefatura del Departamento de Ciencias de la Computación
c.c.p.- Consejero Académico
c.c.p.- Archivo
JJRG,mjda

Dedicado a

Mis hijos: Ernesto Apolinar y Luis Felipe, sirva de muestra el presente para darnos cuenta que nada ni nadie, puede, ni debe truncar los sueños de un hombre.

A mi esposa: Maria Isabel, por estar a mi lado en todo momento.

A mi Madre: Eliazar, por el cariño, el amor y el apoyo que nos has brindado siempre, te queremos mucho.

Agradecimientos

Un agradecimiento especial a mi directora de tesis, la Doctora Eunice Esther Ponce de León Sentí, por todo el apoyo a este proyecto de investigación y las facilidades otorgadas para la presentación del mismo en los distintos escenarios internacionales.

Al Consejo de Ciencia y Tecnología del Estado de Aguascalientes CONCYTEA, por el apoyo recibido para poder continuar mis estudios de doctorado en la Universidad Autónoma de Aguascalientes.

Índice general

Índice de figuras

Índice de cuadros

RESUMEN

La asignación de tareas en un sistema de multicomputadoras, ha sido un problema tratado con diferentes estrategias que buscan optimizar el uso de los procesadores, sin utilizar métodos de planificación previa de tareas en la cola, produciendo por tanto largos tiempos de respuesta a los trabajos que esperan por ejecutarse; las investigaciones realizadas se basan en solucionar un solo problema, tal como: optimización de recursos, minimización de costos de comunicación o minimización de la fragmentación externa, pero no buscan conjuntar los diferentes objetivos que tanto la planificación y la asignación presentan.

En este trabajo abordamos el problema de la planificación de tareas y el problema de la asignación de tareas en un sistema de Multicomputadoras con arquitectura en malla 2D. La propuesta de solución a este problema, establece como primer paso la identificación de los distintos objetivos involucrados en ambos problemas. En un segundo paso, se plantean un algoritmo de planificación y un algoritmo de asignación de tareas; a través de una búsqueda en la cola de espera, el primer algoritmo realiza una planificación previa a la asignación para identificar la lista de tareas que caben en la malla, y el segundo es un algoritmo de distribución marginal univariante (UMDA *Univariate Marginal Distribution Algorithm*) para identificar las mejores asignaciones en la malla de procesadores mediante una asignación cuadrática dinámica. Ambos algoritmos permiten una evaluación de los cinco objetivos propuestos, para determinar el frente de Pareto y obtener la mejor planificación de las tareas en la cola de espera, y de esta forma elegir la mejor asignación en la malla de procesadores.

Las fases de experimentación del presente trabajo son dos: la primera experimentación es, para obtener los resultados al aplicar la evaluación de los cinco objetivos de manera aislada con diferentes cargas de trabajo en el sistema objetivo, y también para realizar una comparación del método propuesto con dos de los métodos más comunes de asignación de tareas: la asignación lineal y el método de curvas de Hilbert, y una segunda experimentación, presenta un proceso de comparación de las diferentes funciones objetivo, lo que permite evaluar de manera conjunta el problema multiobjetivo.

A través de esta evaluación se crea el frente de Pareto que permite realizar una comparación de los resultados de todas las funciones y elegir la que presente los mejores resultados para realizar la asignación en la malla de procesadores. Los resultados obtenidos, que se explican de manera detallada en el capitulo de conclusiones, muestran que el problema debe tratarse como un problema multiobjetivo dinámico, el cual presenta diferentes entornos cada vez que se realiza una planificación y asignación de las tareas en un sistema paralelo.

ABSTRACT

Task assignment of a multi-computer system is a problem that has been treated by using different strategies, that look to optimize the use of the processors without using previous task planning in the queue, that causes long wait times in their execution. The recents researches are based on solving one problem at a time, for example; optimizing resources, minimizing costs and communication or minimizing external fragmentation. However, they do not aim to coordinate different objectives, where as planning and assignment do.

In this investigation we address the problem of planning and task assignment of a multicomputer system that uses a 2D architectural mesh. The proposed solution to this problem establishes the identification of the distinct objectives that are involved as the first step, that in which is defined as the following points:

1. Minimize the number of allocations to the processor mesh that carries out the task allocation algorithm.
2. Minimize the task wait time in the queue.
3. Maximize the use of the processor meshes, or in other words, diminish the percentage of processors that remain free after the assignment algortihm has placed one or more tasks in the processor mesh (external fragmentation).
4. Diminish task starvation. Try to avoid a discrimination in the assignment of tasks that require a large amount of processors (large tasks), this happens because tasks that require only a small amount of processors (small tasks) are being continually assigned.
5. Maximize the adjacency between processors, (assign coordination of free processors to those which are closest) in order to minimize the communication route distance and there by avoid interference between them, ideally to obtain a solid parallel algorithm that diminishes communication time and maximizes processing times.

Step two. A planning algorithm and a task assignment algorithm are planted by means of a search engine within the queue. The first algorithm carries out a planning previous to the assignment, to identify the list of tasks that fit within the mesh. The second one, is a Univariate Marginal Distribution Algorithm (UMDA) that identifies the best assignments in the mesh processors, by means of a quadratic in the mesh processors by means of a quadratic dynamic assignment.

Both algorithms allow for an evaluation of five proposed objectives in order to determine the Pareto front, and to obtain the best planning of the tasks in the queue. In this form, they are able to choose the best assignation within the processor mesh.

The experimental phases of the present investigation are the following two: the first, is to obtain the application results of the five objectives in an isolated manner, using different work loads in the objective system and to also carry out a comparison of the method, using two of the most common task assignment methods. The lineal assignment and Hilbert's curve method allow us to show that:

1. Previous plannings can be carried out, the execution flow is agilized and ingression of tasks are made into the queue.

2. We can see that there is a quadratic task assignment within the processor mesh and that external fragmentation can be diminished using an evolutionary algorithm.

3. A different planning policy is used than that of the FIFO, this produces the initiation of tasks that should be treated with a strategy that comes from within the algorithm, to avoid a large number of tasks being put into the initiation.

A second experiment presents comparison process of the different objective functions, which allows the multiobjective problem to be evaluated in a coordinated way. Through this evaluation the Pareto front is created, that allows us to carry out a comparison of the results of all the functions, and to be able to choose the one that gives us the best results in order to carry out the assignment within the processor mesh.

INTRODUCCIÓN

En este trabajo de investigación, se aborda el problema de la planificación y asignación de trabajos paralelos en un sistema de multicomputadoras, como un problema multiobjetivo. En los trabajos que abordan el problema de la asignación de tareas, se plantea la resolución de un problema en particular, tal como el tiempo de respuesta que el sistema otorga a un trabajo o el tiempo de espera de las tareas en la fila de espera. Cuando se realiza la asignación de trabajos paralelos a los recursos computacionales todos lo objetivos deben converger en conjunto para determinar los mejores tiempos que se buscan mejorar, y realizar una planificación previa de los trabajos antes de realizar la asignación de los recursos a las tareas, que esperan por ejecutarse. El método que se propone para la resolución de dicho problema multiobjetivo, es a través de un algoritmo evolutivo UMDA.

Capítulo 1

PLANTEAMIENTO DEL PROBLEMA DE ESTUDIO

1.1. Descripción general del problema de investigación y su contexto

En el ámbito de la computación paralela y distribuida, se presentan una serie de problemas a los que a través del tiempo se han propuesto soluciones con diferentes técnicas y métodos. La asignación de procesadores a las tareas que se ejecutan dentro de una malla de procesadores, es uno de los problemas propuestos. Así, en el contexto del cómputo paralelo y las técnicas metaheurísticas, se plantea el problema general del problema de investigación de la forma siguiente:

El problema general de investigación consiste en desarrollar un algoritmo metaheurístico eficiente para planificar y asignar de manera óptima tareas en un sistema de multicomputadoras en una malla 2D.

El contexto general del problema de investigación se fundamenta en 2 áreas de investigación: la computación evolutiva y el cómputo paralelo, que a modo introductorio se definen a continuación. La computación evolutiva se define como una técnica de resolución de problemas de búsqueda y optimización inspirada en la teoría de la evolución de las especies y la selección natural [5]. Estos algoritmos reúnen características de búsqueda aleatoria con caracterśticas de búsqueda dirigida que provienen del mecanismo de selección de los individuos más aptos. La unión de ambas características les permite abordar los problemas de una forma muy particular, ya que tienen capacidad para acceder a cualquier región del espacio de búsqueda, capacidad de la que carecen otros métodos de búsqueda exhaustiva, a la vez que exploran el espacio de soluciones de

una forma mucho más eficiente que los métodos puramente aleatorios. Indudablemente, un algoritmo diseñado de forma específica para la resolución de un problema concreto será más eficiente que un algoritmo evolutivo, que es una técnica general de resolución [5].

El cómputo paralelo, definido como la ejecución de dos o más procesos al mismo tiempo usando más de un procesador, permite que un programa se ejecute rápidamente porque existen más procesadores que trabajan en forma simultánea y coordinada, en diferentes partes o subtareas de un problema. La meta es reducir al mínimo el tiempo total de cómputo distribuyendo la carga de trabajo entre los procesadores disponibles.

1.1.1. Contexto de la investigación

La asignación y planificación de tareas en una malla de procesadores surge de la necesidad de asignar de manera óptima las tareas y las subtareas a conjuntos de procesadores para la ejecución de trabajos inherentemente paralelos en tiempos más cortos en relación con sistemas de un solo procesador.

La asignación y planificación de tareas es inherentemente un problema multiobjetivo en el cual se pretende:

- Disminuir el número de asignaciones a la malla de procesadores que realiza el algoritmo de asignación de tareas.
- Disminuir el tiempo de permanencia de las tareas en la cola de espera.
- Maximizar el uso de procesadores de la malla, es decir, disminuir el porcentaje de procesadores que permanecen libres después que el algoritmo de asignación coloca una o más tareas en la malla de procesadores (fragmentación externa).
- Disminuir la inanición de tareas, es decir evitar una discriminación en la asignación de las tareas que requieren una gran cantidad de procesadores (tareas grandes), provocada porque las tareas que requieren una poca cantidad de procesadores (tareas pequeñas) estén siendo asignadas continuamente.
- Maximizar la contigüidad entre procesadores (asignar el conjunto de procesadores libres lo mas cercanos posible) para minimizar la distancia en la ruta de comunicación y evitar la interferencia entre los mismos; esto para obtener un buen algoritmo paralelo que disminuya el tiempo de comunicación y maximice el tiempo de procesamiento.

De esta forma, al detectar un conjunto de objetivos contrapuestos en el desarrollo de este trabajo, se busca encontrar una solución óptima para encontrar la mejor solución al realizar una asignación en los espacios vacíos de un malla de procesadores. El contexto de la investigación planea también encontrar los objetivos que tienen un mayor grado de contraposición al ser evaluados durante la fase de planificación, para de esta manera poder otorgar un cierto grado de importancia al objetivo que tenga mayor peso, al tomar la decisión de escoger el conjunto de tareas que se ejecutarán.

1.1.2. Contexto del algoritmo de estimación de la distribución

Los algoritmos de estimación de la distribución (EDA *Estimation of Distribution Algorithms* por sus siglas en inglés), son algoritmos heurísticos de optimización que basan su búsqueda (al igual que los algoritmos genéticos) en el carácter estocástico de la misma.

Las características de los algoritmos de estimación de la distribución pueden enumerarse de la siguiente forma [37]:

- Es un paradigma de la computación evolutiva.
- Se basan en los algoritmos genéticos.
- Generalizan los algoritmos genéticos al remplazar los operadores de cruce y mutación por el aprendizaje y muestreo de la distribución de probabilidad de los mejores individuos de la población en cada iteración del algoritmo.
- Las variables involucradas en el dominio del problema son explotadas y capturadas explícita y efectivamente.

En la literatura actual no encontramos una aplicación en la que se considere la detección de múltiples objetivos y la aplicación de los algoritmos EDA a la planificación y asignación de tareas en un sistema de multicomputadoras, por lo que consideramos que el presente trabajo es el primero en este campo de investigación. La gran mayoría de los trabajos desarrollados se basan en la detección de procesadores libres a través de cálculos geométricos en la malla e ignoran la planificación, utilizando únicamente la política de asignación del primero en llegar el primero en salir (FIFO de sus siglas en inglés *First Input First Output*). Aunque FIFO es una política de asignación con un porcentaje nulo de inanición de tareas en la cola, produce mucho tiempo de espera de las tareas.

1.2. Relevancia y justificación de la investigación

Los Algoritmos de Estimación de la Distribución y en específico los algoritmos de distribución marginal univariante (UMDA por sus siglas en Inglés *Univariate Marginal Distribution Algorithm*) no han sido aplicados al problema multiobjetivo de la planificación y asignación de procesadores; sin embargo el problema de asignación de procesadores en sistemas paralelos se ha abordado con múltiples técnicas (geométricas, de búsqueda y aleatorias). La relevancia de este trabajo estriba en el aspecto de mezclar la inteligencia artificial y los sistemas de cómputo paralelo para demostrar la posibilidad de aplicar técnicas heurísticas en ambientes distribuidos.

Las características por las que esta investigación usa el algoritmo UMDA son:

- No son por sí mismos un problema al momento de elegir los mejores individuos de una población.
- Ofrecen una mejor perspectiva sobre lo que hace la genética en una población de individuos.
- Usan técnicas desarrolladas en estadística, inteligencia artificial y agrupamiento.
- Ofrecen una mejor calidad en las soluciones.
- No hacen suposiciones relacionadas con la biología o técnicas basadas en el campo de la computación evolutiva.

1.3. Descripción general del enfoque de investigación

Esta investigación se ubica dentro de las ciencias computacionales, específicamente en la disciplina de la inteligencia artificial y dentro de ella en el área de la computación evolutiva.

El problema a resolver es la planificación y asignación de tareas y subtareas en un sistema de multicomputadoras con arquitectura en malla 2D.

El objetivo principal es el diseño de un Metaplanificador, constituido por un algoritmo planificador de tareas y un algoritmo asignador de procesadores, que utilizan metaheurísticas para permitir a trabajos paralelos constituidos por subtareas, ejecutarse dentro de una arquitectura de cómputo paralelo.

El producto obtenido es un algoritmo de planificación y asignación de tareas de una forma óptima dentro de la malla de procesadores mediante:

- Programación paralela.
- Metaheurística evolutiva: EDAs.
- Optimización Multiobjetivo (MOO por sus siglas en inglés *Multi Objective Optimization*).
- C estándar como lenguaje de programación.

Esta investigación contribuye en las áreas de cómputo paralelo y computación evolutiva específicamente en el problema de la planificación y la asignación de tareas.

1.4. Formulación de la investigación

En esta sección se plantea la descripción del problema de investigación, el objetivo general del proyecto y los objetivos específicos que se persiguen al desarrollar este trabajo de investigación.

1.4.1. Descripción del problema de investigación

El problema de investigación consistió en el planteamiento de un método, que modela el problema de asignación cuadrática para la asignación de tareas a procesadores, así como también un método para modelar el problema de la planificación de tareas que esperan en una cola, para ser ingresadas a una malla de procesadores en un sistema de multicomputadoras.

Usando una clase de los algoritmos EDA, en especifico, el algoritmo UMDA, se realiza el cálculo de la distribución de probabilidad conjunta de las tareas seleccionadas en la cola de espera que son susceptibles de ser asignadas a la malla de procesadores. En seguida, un algoritmo de asignación de tareas en la malla, se activa para localizar la mejor ubicación de las tareas seleccionadas. Ambos algoritmo trabajan de manera conjunta para lograr mejorar los tiempos de respuesta que el sistema otorga a los usuarios.

Para demostrar la efectividad del metodo que se propone, se diseñaron experimentos que muestran la eficiencia de los algoritmos propuestos, con respecto a los algoritmos del estado del arte reportados en la literatura.

1.4.2. Objetivo general de la investigación

Diseñar un metaplanificador, constituido por un algoritmo planificador de tareas y un algoritmo asignador de procesadores, mediante metaheuristicas permitan a trabajos paralelos compuestos por subtareas, ejecutarse dentro de una arquitectura de cómputo paralelo conformada por un sistema de multicomputadoras en una malla 2D, utilizando una red directa de comunicación.

1.4.3. Objetivos específicos de la investigación

Los objetivos específicos de este trabajo de investigación son los siguientes:

- Determinar si las técnicas heurísticas permiten optimizar los tiempos de ejecución y los recursos computacionales de un sistema de multicomputadoras en malla 2D.
- Evaluar una estrategia híbrida, basada en metaheuristicas multiobjetivo, de planificación de tareas y asignación de procesadores para la ejecución de trabajos paralelos, constituidos por tareas que se comunican utilizando el paso de mensajes en una malla de n procesadores de un sistema de multicomputadoras.
- Desarrollar, implementar y aplicar un algoritmo heurístico para la planificación de tareas intrínsecamente paralelas, a ser ejecutadas en un sistema de multicomputadoras basado en mallas utilizando un simulador y un entorno real.
- Desarrollar, implementar y aplicar un algoritmo heurístico de asignación de procesadores para minimizar la fragmentación interna y externa, minimizar la transferencia de mensajes y optimizar recursos dentro del sistema de multicomputadoras utilizando un simulador.

1.5. Difusión y aprobación del trabajo

- Participación como cartel en el coloquio del doctorado en Ciencias Exactas, Sistemas y de la Información y de la Maestría en Ciencias con opciones en Computación y Matemáticas Aplicadas, realizado en la Universidad Autónoma de Aguascalientes, los días 18 y 19 de junio de 2012, en Aguascalientes, Ags. México.

  La presentación del proyecto de investigación en este evento tuvo como objetivo la primera revisión de la propuesta de tesis doctoral en un evento local. En este documento se presentó la fundamentación teórica del proyecto, así como un breve resumen de la primer fase de la investigación.

- Participación como ponente en el Consorcio Doctoral en el 9^{TH} MEXICAN INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (MICAI 2010) llevado a cabo el 9 de Noviembre de 2010 en la Ciudad de Pachuca Hidalgo, México.

  Se presentó este proyecto con el objetivo de dar a conocer la investigación en un evento internacional, y hacer saber que ya había sido aceptado como propuesta doctoral en la Universidad Autónoma de Aguascalientes, México. En esta presentación únicamente se mostraron y explicaron los objetivos del proyecto, la primera sección investigada del estado del arte. Una vez presentado el proyecto se recibieron observaciones en el área del problema de la asignación cuadrática dado que éste se muestra como un problema NPcompleto. En términos generales, el proyecto fue aceptado como un área incipiente de aplicación de las metaheurísticas.

- Participación como ponente en el 10^{TH} MEXICAN INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (MICAI 2011) llevado a cabo del 26 de Noviembre al 4 de Diciembre de 2011 en la Ciudad de Puebla, México.

  Los resultados iniciales obtenidos con el método propuesto en el evento anterior, fueron presentados en este evento con el nombre: *Dynamic Quadratic Assignment to model Task Assignment Problem to Processors in a 2D mesh*. Los resultados presentados se consideraron como parciales, puesto que evaluaron una parte de los objetivos propuestos inicialmente.

- Publicación en el evento EVOLVE 2013, celebrado entre el 11-13 Julio de 2013 en la Ciudad de Leiden, Paises Bajos.

  La presentación del trabajo: *Planning and Allocation Tasks in a Multicomputer System as a Multi-objective Problem* en este evento internacional, fue para presentar el trabajo completamente terminado. Este articulo presentó los resultados obtenidos en la evaluación de los objetivos y mostró las conclusiones obtenidas con las experimentaciones realizadas.

- Propuesta para el desarrollo de un capítulo de libro, en el Postproceedings of EVOLVE A Bridge between Probability, SetOriented Numerics, and Evolutionary Computation 2013.

  El objetivo del desarrollo de esta publicación es para presentar los resultados obtenidos, tanto por el método propuesto como por la extensión al mismo, así como también las conclusiones obtenidas con las experimentaciones realizadas y mostrar que el problema es visto como un problema multiobjetivo dinámico.

1.6. Aportaciones del trabajo de investigación

Las aportaciones de este trabajo de investigación son:

- Un metaplanificador que permita realizar la planificación y asignación de tareas a una malla de procesadores dentro de un sistema de multicomputadoras.
- Un algoritmo para evaluar los objetivos contrapuestos que se presentan durante la planificación y asignación de tareas en una malla 2D.
- La aplicación de un algoritmo evolutivo en un entorno de programación paralela, a través de la cual se permitirá desarrollar aplicaciones de programas multiobjetivo en un ambiente paralelo.

1.7. Contenido de la tesis

Capítulo 1. Planteamiento del problema de estudio. En este capítulo se muestra, el contenido de la tesis en forma general, comenzando con una introducción a los sistemas de cómputo paralelo, haciendo énfasis en los sistemas de multicomputadoras. Se plantea el problema de la planificación y asignación de tareas a procesadores en una malla con arquitectura 2D, asi como los objetivos que se buscan minimizar o maximizar durante la fase de la planificación de tareas en la cola de espera y la fase de asignación de las tareas seleccionadas.

De igual forma, en este capítulo se fundamenta el entorno general del problema de investigación y su contexto, las áreas de investigación involucradas, la relevancia del proyecto y la justificación de uso de cada una de las técnicas involucradas. Finalmente se muestra una explicación amplia de los escenarios, tanto nacionales como internacionales en los cuales se presentó este trabajo, así como la finalidad de asistir a cada una de uno de ellos.

Capítulo 2. Marco teórico. Los fundamentos teóricos que son la base del presente trabajo de tesis se detallan en este capítulo. El contenido abarca temas relacionados con diferentes áreas del conocimiento, como son: las arquitecturas paralelas actuales, las técnicas de comunicación en sistemas paralelos, así como la optimización multiobjetivo basado en el frente de Pareto.

La fundamentación teórica presentada en este capítulo nos permite visualizar de manera general el contexto del problema de investigación propuesto en el capitulo anterior,

así como también la forma en que dichas tecnologías se apoyan para proponer una solución, tanto a la fase de planificación como en la fase de asignación de tareas en la malla de procesadores.

Capítulo 3. Estado del arte de la planificación y asignación de procesadores en una Malla 2D. La revisión al estado del arte de la planificación y asignación de tareas, llevada a cabo a partir de diferentes fuentes de información se resumen en este capítulo. Los métodos más significativos propuestos en esta área de investigación, las técnicas desarrolladas y los resultados obtenidos son explicados de forma resumida para dar una idea de las últimos avances en este tema.

Este capitulo hace énfasis en la bifurcación de las dos técnicas que existen, en el ámbito del reconocimiento de una malla de procesadores del sistema de multicomputadoras: reconocimiento parcial y reconocimiento total. Ambos reconocimientos desarrollados en diferentes investigaciones a través del tiempo han sido el parte aguas, para obtener mejores técnicas del manejo de recursos en la computación paralela. El conocimiento de estas técnicas previas, ha sido de suma importancia para el planteamiento que se propone en este trabajo, y que nos ha otorgado la visibilidad para plantear que no existe en la literatura un método que aborde este problema de investigación con las técnicas que hemos propuesto.

Capítulo 4. Método propuesto. La descripción del método que se propone en esta tesis, para llevar a cabo la planificación y asignación de las tareas en la malla de procesadores, se describe ampliamente en este capítulo. Todos los procedimientos involucrados y los algoritmos propuestos para cada una de las fases que integran el método, se detallan en las secciones de este capítulo.

Capítulo 5. Discusión de resultados. Esta sección describe la forma en que se realizaron los experimentos con los algoritmos de planificación y asignación de tareas. Un conjunto de gráficas comparativas explican de manera detallada los resultados obtenidos durante las distinta fases de experimentación.

Las experimentaciones se han dividido en dos fases: en la primera fase se aplicó el método inicial propuesto, que contempla la evaluación de cada uno de los cinco objetivos. Para cada uno de los objetivos, se crea la gráfica correspondiente de sus resultados, para permitir realizar una comparación del método propuesto, con el método de las curvas de Hilbert y el método de la asignación lineal.

Las experimentaciones en la segunda fase, se lleva a cabo con una extension al método propuesto, la cual se basa en la evaluación de cada una de las funciones, y en la evaluación de los valores obtenidos de las funciones para determinar y analizar el frente de Pareto. De esta manera, se busca encontrar los valores no dominados del conjunto de funciones objetivo. Los resultados obtenidos son mostrados a través de diferentes gráficas.

Conclusiones. En este capítulo se exponen las conclusiones de este trabajo de tesis, una vez que se finalizaron los desarrollos de los algoritmos y las fases de experimentación. También se incluyen los trabajos futuros, que se han planificado desarrollar posterior a la finalización de este proyecto.

Capítulo 2

MARCO TEÓRICO

Este capítulo contiene una revisión de los fundamentos teóricos que son la base del presente trabajo de tesis; se consideran los temas relacionados con la computación paralela, la inteligencia artificial y las matemáticas.

2.1. Descripción de teorías bases

Esta investigación se basa en el entendimiento de los procesos de planificación y de asignación de tareas y subtareas a una malla de procesadores en un sistema de multicomputadoras, ambos vistos como un problema multiobjetivo. Las disciplinas base o de referencia son: los sistemas de cómputo paralelo, los algoritmos evolutivos y la optimización multiobjetivo.

2.1.1. Disciplinas de referencia

El presente proyecto de investigación se fundamenta en un conjunto de disciplinas que pertenecen a diferentes áreas que interactúan de manera conjunta para conformar la estructura del proyecto mismo.

Las disciplinas tanto del área de la computación como de las heurísticas, sobre las que se fundamentan los conceptos sobre la planificación y asignación de tareas en un sistema de multicomputadoras, y que se abordan en la presente tesis son:

- Sistemas de cómputo paralelo
- Sistemas de multicomputadoras
- Complejidad computacional

- Optimización multiobjetivo

Cada una de las disciplinas mencionadas anteriormente, se explican de manera detallada en los siguientes capítulos de la presente tesis.

2.1.2. Sistemas de cómputo paralelo

La computación paralela y distribuida ha sido considerada como el futuro de la computación de alto desempeño [19]. Los diferentes sistemas de computación distribuida tales como los sistema multiprocesadores de memoria compartida, sistemas multicomputadora con transferencia de mensajes y sistema distribuidos de área amplia se han estudiado e investigado extensivamente para obtener más potencia computacional y permitir de esta manera que tareas paralelas se ejecuten en tiempos más cortos [27]. Ejemplos típicos de estas tareas son el procesamiento de imágenes, sistemas software para hospitales, industrias de manufactura y sistemas de transportación [61].

La computación paralela consiste de un conjunto de procesadores que cooperan para encontrar una solución a un problema dado [56]. La comunicación entre procesadores puede estar basada en un modelo de memoria compartida o en un modelo de memoria distribuida. En las arquitecturas de memoria compartida, también conocidas como de sistemas multiprocesadores, los procesadores se comunican mediante una memoria que comparten durante la ejecución de las tareas y en las computadoras paralelas con memoria distribuida, también conocidas como multicomputadoras, la comunicación entre procesadores se establece mediante el intercambio de mensajes a través de la red de comunicación [60].

Para solucionar un problema mediante computación paralela, éste debe ser dividido en subproblemas que serán asignados a un conjunto de procesadores. La dependencia de datos que existe entre los subproblemas puede dificultar el proceso de dividir el problema. La figura 2.1 ilustra el proceso que sigue la división de un problema hasta su fase de programación paralela [48]; a diferencia de la programación de sistemas secuenciales, la programación paralela debe dividir la aplicación en subtareas para permitir la distribución de la carga computacional entre los procesadores.

Cuando cada subproblema se asigna a cada procesador en el sistema paralelo, uno de los objetivos es asignar el conjunto de procesadores lo más cercanos posible para evitar la carga de comunicación en la red de comunicación. Los resultados de cada procesamiento deben ser eficientemente combinados para obtener el resultado final del problema. El punto importante aquí, es el tiempo que se usa para la comunicación entre

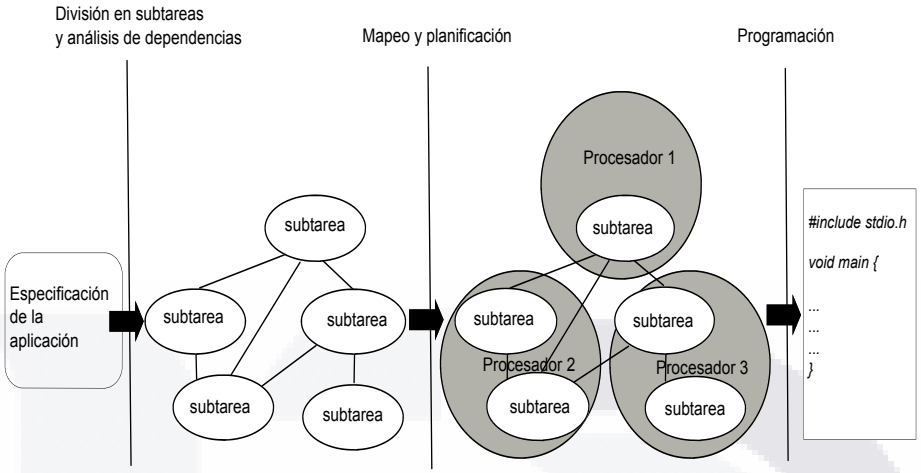

Figura 2.1: División de un problema durante su fase de programación paralela [48].

dos procesadores comparado con el tiempo de procesamiento. Ambos tiempos deben ser muy bien planeados, para obtener un buen algoritmo paralelo [56].

Existen varios métodos para implementar la computación paralela: a través de mallas [22, 51], cubos (grids) [3], y plataformas heterogéneas [27], usando ciertas métricas de desempeño, heurísticas tales como recocido simulado [19], el algoritmo voraz [3], y métodos alternativos de medición de cargas de trabajo, donde la carga no es generada por descubrimiento, sino generada dinámicamente por modelos de usuario que interactúan con el sistema y cuya conducta en una simulación es similar a la conducta de usuarios en la realidad [49].

La gran mayoría de las investigaciones convergen en que deben coexistir dos algoritmos para permitir la ejecución de las tareas en un sistema distribuido: el algoritmo de planificación de tareas y el algoritmo para la asignación de procesadores a las tareas [58].

El Algoritmo de planificación de tareas determina el orden de ingreso de las tareas a la malla de procesadores después que el algoritmo asignador informa de las submallas libres. Una vez que una tarea se selecciona por el algoritmo de planificación, se asigna dentro de la malla para iniciar su ejecución sin interrupción hasta que finaliza, momento en que es retirada de la malla de procesadores, entonces, la cola de tareas se contrae y la subsecuente tarea ingresa a la malla de procesadores. Por ejemplo, suponga una cola de espera de 6 tareas, en la que cada tarea consiste de un conjunto de subtareas que

van de 2 a 5 tareas, y una malla de 8×8 procesadores desde el procesador $< 0, 0 >$ hasta el procesador $< 7, 7 >$ como lo muestra la figura 2.2.

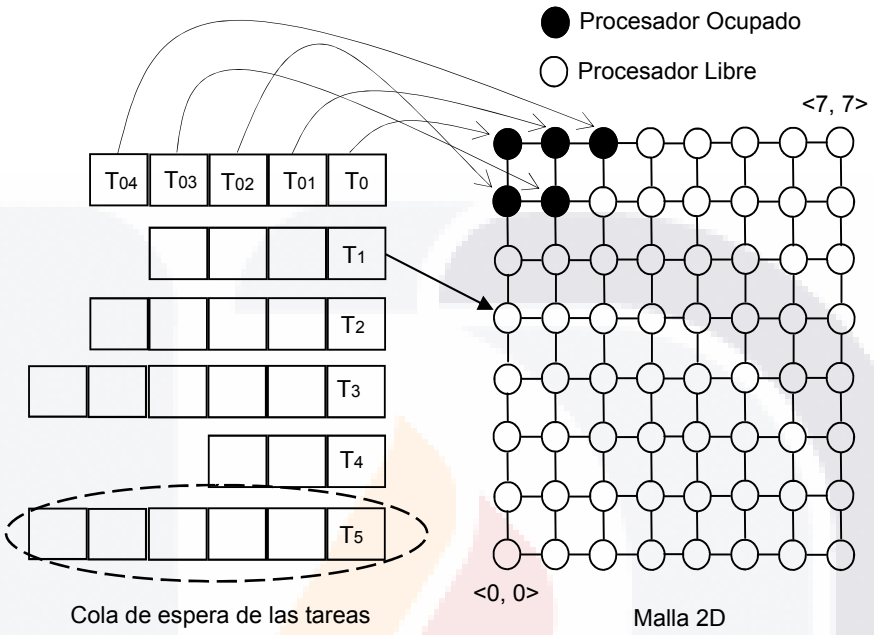

Figura 2.2: Planificación y asignación de tareas en una malla 2D.

En un tiempo t, el proceso de planificación determinará cuántas y cuáles tareas ingresarán a la malla, dependiendo del número de procesadores libres; para este caso suponga que la tarea 1, en la cabeza de la cola, ha sido seleccionada para ser asignada a la malla; después de su ingreso la cola de espera se contraerá, la tarea 2 pasa a ser la cabeza de la cola, y todas las tareas que permanezcan en la malla esperarán una siguiente planificación y asignación para su ejecución. La tarea 6 pasara a la posición 5 de la cola y un espacio estará disponible para el ingreso de una tarea nueva que podrá competir por su ingreso a la malla en la siguiente planificación.

Así como únicamente la tarea 1 fue planificada para su ejecución, se pueden planificar más tareas que ingresarán a la malla, dependiendo del número de procesadores desocupados que existan en la misma y del tipo de planificación que los algoritmos utilicen para colocar las tareas.

Los algoritmos de planificación de tareas pueden clasificarse como aquéllos que realizan una política libre de inanición, dado que tienen la propiedad de ser no discrimi-

nativos y aquellos cuya planificación se basa en un ordenamiento o búsqueda de tareas que puedan ser colocadas dentro de la malla; una vez que las tareas han terminado su ejecución, dejan submallas de procesadores libres dentro de la malla.

El algoritmo asignador de procesadores es responsable de informar el estado de la malla en un tiempo t, de asignar procesadores a las tareas que ingresan a la malla para su ejecución y desasignar procesadores una vez que la tarea termina, localizar las submallas libres y determinar los costos de comunicación entre tareas que son asignadas en submallas no adyascentes. Una tarea que ha sido asignada a la malla de procesadores para su ejecución se retira de la misma una vez que finalice y no se permite la migración de tareas o subtareas dentro de la misma.

En la computación paralela algunas de las funciones objetivo que son susceptibles de ser minimizadas o maximizadas para lograr un mejor desempeño son [49]:

1. Minimizar la inanición de tareas,
2. Minimizar la fragmentación interna,
3. Minimizar los costos de comunicación en la red de procesadores,
4. Minimizar el número de trabajos en la cola de espera,
5. Maximizar el número de tareas ejecutándose en paralelo dentro de la malla,
6. Minimizar el tiempo de respuesta a los usuarios para mantener su satisfacción y motivarlos para ejecutar más trabajos en el sistema.

Los problemas de optimización que modelan un sistema físico que involucran una función objetivo que ejecutan la tarea de encontrar una solución óptima son llamados de optimización de un solo objetivo, y cuando el problema de optimización involucra más de una función objetivo, la tarea de encontrar una o más soluciones óptimas se conoce como optimización multiobjetivo, también conocido como Realizar Decisiones Multicriterio (MCDM de sus siglas en inglés *Multiple Criterion Decision-Making*) [26].

En la computación evolutiva han surgido los algoritmos evolutivos multiobjetivo paralelos, cuyas aplicaciones se basan en la solución de problemas complejos que requieren excesivas cantidades de memoria, reducen la probabilidad de caer en óptimos locales, encuentran soluciones en dominios simultáneamente y trabajan con problemas multiobjetivo [15].

2.2. Arquitecturas actuales de la computación paralela

El procesamiento paralelo se define como el procesamiento de información acentuando la manipulación concurrente de elementos de información que pertenecen a uno o más procesos que resuelven un problema [44]. La computación paralela satisface las crecientes demandas de alto rendimiento de las aplicaciones científicas y de propósito general. Este tipo de computación requiere de varios procesadores trabajando en forma conjunta; actualmente se clasifican en dos grandes categorías [44, 55, 56]:

1. Una instrucción múltiples datos,
2. Múltiples instrucciones múltiples datos,

Ambos tipos de computación paralela se describen en los siguientes párrafos.

Una Instrucción, múltiples datos SIMD (de sus siglas en inglés, *Single Instruction stream, Multiple Data stream*). Esta categoría se define como una técnica empleada para conseguir paralelismo a nivel de datos y consiste en instrucciones que aplican una misma operación sobre un conjunto más o menos grande de datos. Es una organización que influye en muchas unidades de procesamiento bajo la supervisión de una unidad de control común, es decir, una única unidad de control despacha las instrucciones a diferentes unidades de procesamiento. Todos los procesadores reciben la misma instrucción de la unidad de control, pero operan sobre diferentes conjuntos de datos, es decir la misma instrucción se ejecuta de manera síncrona por todas las unidades de procesamiento.

Múltiples Instrucciones, Múltiples Datos MIMD (de sus siglas en inglés, *Multiple Instruction stream, Multiple Data stream*). Tienen un número de procesadores que funcionan independientemente. En cualquier momento, pueden ser diferentes procesadores y diferentes instrucciones sobre la ejecución de diferentes conjuntos de datos. En esta categoría cada procesador es capaz de ejecutar un programa distinto independientemente de los demás. Su clasificación con base a la organización del espacio de direcciones es:

- *Sistema Multiprocesadores de memoria compartida.* En estos sistemas 2, 1000 o más CPU se comunican por medio de una memoria compartida. Cada CPU tiene el mismo acceso a toda la memoria física y puede leer y escribir palabras individuales utilizando instrucciones LOAD y STORE. El acceso a una palabra de memoria por lo regular tarda de 10 a 50ns. La implementación de este modelo

tiene una alta dificultad inherente y por lo regular implica un tráfico considerable de mensajes. En la figura 2.3 se muestra este tipo de sistemas.

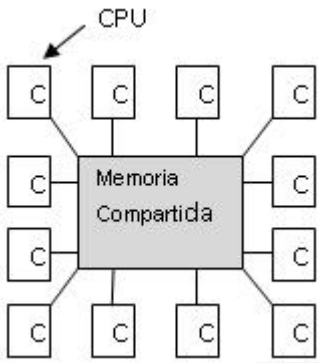

Figura 2.3: Multiprocesador de memoria compartida.

- *Sistema multicomputadoras con transferencia de mensajes.* Este tipo de sistemas están constituidos por varios pares de CPU-memoria conectados entre sí, mediante algún tipo de interconexión de alta velocidad. Cada memoria es local con respecto a una sola CPU y solo se tiene acceso a ella por medio de ésta. Las máquinas se comunican enviando mensajes de varias palabras por la interconexión. Con una buena interconexión un mensaje corto puede enviarse en 10-50μs, pero tarda mucho más que los multiprocesadores de memoria compartida. Estos sistemas son ejemplificados en la figura 2.4.

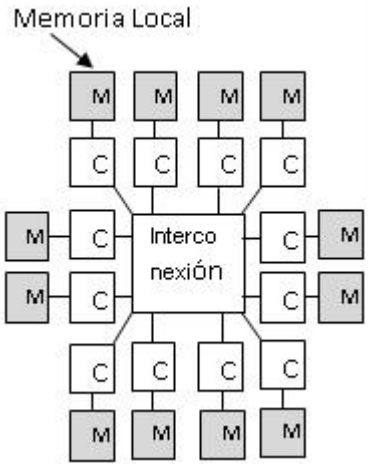

Figura 2.4: Multicomputadora con transferencia de mensajes.

- *Sistema distribuido de área extensa.* Son sistemas de cómputo completos por medio de una red extensa, como Internet, para formar un sistema distribuido. Cada computadora tiene su propia memoria y los sistemas se comunican con mensajes, los cuales suelen tardar 10-50ms. Estos sistemas son denominados también sistemas débilmente acoplados. Un esquema general de tal sistema se muestra en la figura 2.5.

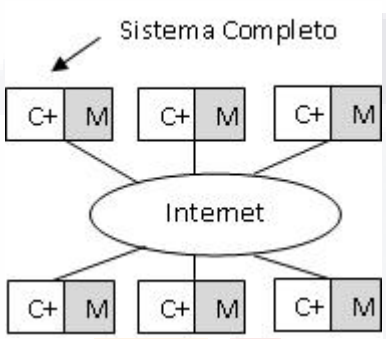

Figura 2.5: Sistema distribuido de área extensa.

Una de las principales diferencias de los tres sistemas es el retraso en la comunicación dado que difieren en casi tres órdenes de magnitud. Esa es la diferencia entre un día y tres años dado que toda comunicación entre componentes electrónicos u ópticos, se reduce en última instancia al envío de mensajes (cadenas de bits bien definidos) entre ellos [55]. La diferencia radica en la escala de tiempo, la escala de distancia y la organización lógica empleada para la transferencia de tareas entre computadoras, cuando se requiere que se comuniquen para trabajar en la resolución de un mismo problema. Los algoritmos de planificación de tareas consideran los tres objetivos anteriores para optimizar las respuestas a los usuarios.

Los sistemas multiprocesadores utilizan sistemas operativos normales: manejan llamadas al sistema, administran la memoria, proporcionan un sistema de archivos y administran dispositivos de Entrada/Salida (E/S). En ciertas áreas tienen características únicas: incluyen la sincronización de procesos, la administración de recursos y la calendarización (planificación). Utilizan características especiales en su hardware, por lo que construir multiprocesadores grandes es difícil y, por tanto, su costo es elevado.

Las multicomputadoras tienen varias CPU fuertemente acopladas, no comparten memoria RAM, tienen una tarjeta de interfaz, tal vez un disco duro para paginar y una red

de interconexión de alta velocidad cuyo uso se ha extendido a aplicaciones de ingeniería y aplicaciones científicas [51], están en un mismo recinto para poder comunicarse por medio de una red dedicada de alta velocidad para el envío de mensajes en una escala de tiempo de microsegundos, ejecutan el mismo sistema operativo, comparten un mismo sistema de archivos y están sometidos a una administración común. Un ejemplo de estos sistemas es un conjunto de nodos en un mismo recinto de una compañía o universidad trabajando en un modelado farmacéutico. Los nodos de un sistema multicomputadoras pueden verse como computadoras personales sin componentes superfluos. A partir del primer prototipo, el Cosmic Cube desarrollado en 1981 ha sufrido un desarrollo muy rápido. Su tardía aparición cabe atribuirla a tres factores fundamentales [44]: los requisitos de memoria, la comunicación entre procesadores y la dificultad de programación. De cierta manera, esta arquitectura puede considerarse como una extensión de las redes de estaciones de trabajo. Sin embargo, se diferencia por el grado de paralelismo, el orden de magnitud del tamaño de la red, y la capacidad de las memorias locales. La red de intercomunicación es mucho más densa que en las redes de estaciones de trabajo y las demoras producidas en la comunicación se miden en centenas de microsegundos.

En los sistemas distribuidos cada nodo es una computadora completa con un surtido completo de periféricos que pueden estar dispersos por todo el mundo, ejecutan sistemas operativos distintos, cada uno tiene su propio sistema de archivos y están bajo diferentes administraciones [55]. Un modelo representativo consiste en miles de máquinas que cooperan de manera informal a través de Internet.

Generalmente las redes de interconexión se pueden dividir en dos categorias: redes directas y redes indirectas [29, 36]. En las redes indirectas, se usan los Switches para interconenectar los nodos, ejemplos de redes indirectas son los travesaños (del termino en Inglés *Crossbar*) [4, 31], bus [29], y redes de interconexión multiestado. En las redes directas, cada nodo tiene una conexión punto a punto a uno o mas nodos (conocidos como sus vecinos), que permiten comunicación directa entre ellos. Ejemplos de redes directas son la malla (*mesh*), k-ary, n-cube e hipercubo [56]. Las redes directas han sido extensamente usadas en multicomputadoras de gran escala debido a su escalabilidad; pueden ser escaladas adicionando nodos y canales de comunicación basados en la estructura de red predefinida [42]. Las redes directas son capaces de explotar la comunicación local (la comunicación entre vecinos cercanos) que permiten ejecutar muchas aplicaciones del mundo real [10].

De las arquitecturas de multicomputadoras las que han recibido mucha atención debido a su simplicidad, regularidad estructural, facilidad de particionamiento y facilidad de implementación han sido las redes de malla [20, 22, 59, 61]. Las redes de malla son

apropiadas para una variedad de aplicaciones tales como cálculos matriciales, procesamiento de imágenes y problemas cuyas tareas gráficas pueden ser incrustadas dentro de las mallas [19, 27]. Las mallas han sido usadas como redes dentro de máquinas paralelas experimentales y prácticas [10] como por ejemplo la Intel Paragon [35], Delta Touchstone [34], Cray XT3 [38, 21], MIT J-machine [43] y la IBM BlueGene/L [13, 6].

2.2.1. Conceptos básicos

El sistema propuesto en este trabajo de tesis es un sistema de multicomputadoras conectada en malla, con una arquitectura 2D, con una cola de espera que contiene una lista de tareas que esperan por ejecutarse en dicho sistema mediante una asignación cuadrática dinámica. En esta sección se definen cada uno de los términos que se utilizarán en las siguientes secciones de esta tesis.

Definicion 1.1. *Una malla n-dimensional tiene* $k_0 \times k_1 \times ... \times k_{n-2} \times k_{n-1}$ *nodos, donde* k_i *es el número de nodos a lo largo de la i-ésima dimensión y* $k_i \geq 2$*.Cada nodo a esta identificado por n coordenadas,* $\rho_0(a)$*,* $\rho_1(a)$*, ... ,* $\rho_{n-2}(a)$*,* $\rho_{n-1}(a)$*, donde* $0 \leq \rho_i(a) < k_i$ *para* $0 \leq i < n$*. Dos nodos a y b son vecinos sí y sólo sí* $\rho_i(a) = \rho_i(b)$ *para todas las dimensiones, excepto para una dimensión j, donde* $\rho_j(b) = \rho_j(a) \pm 1$*. Cada nodo en una malla se refiere a un procesador y dos vecinos estan interconectados por un enlace de comunicación directo.*

Definicion 1.2. *Una malla 2D, que se referencia como* $M(W, L)$ *consiste de* $W \times L$ *procesadores, donde W es el ancho de la malla y L es la longitud. Cada procesador se denota por un par de coordenadas* (x, y)*, donde* $0 \leq x < W$ *y* $0 \leq y < L$*. Un procesador esta conectado por un enlace de comunicación bidireccional a cada uno de sus vecinos.*

Definicion 1.3. *En una malla 2D,* $M(W, L)$*, una submalla* $S(w, l)$ *es una submalla de dos dimensiones de nodos pertenecientes a* $M(W, L)$ *con ancho w y longitud l, donde* $0 < w \leq W$ *y* $0 < l \leq L$*.* $S(w, l)$ *esta representado por las coordenadas* (x, y, x', y')*, donde* (x, y) *es la esquina inferior izquierda de la submalla y* (x', y') *es la esquina superior derecha. El nodo de la esquina inferior izquierda se denomina el nodo base de la submalla y el nodo de la esquina superior derecha es el nodo final. En este caso* $w = x' - x + 1$ *y* $l = y' - y + 1$*. El tamaño de* $S(w, l)$ *es* $w \times l$ *procesadores.*

Definicion 1.4. *En una malla 2D* $M(W, L)$*, una submalla disponible* $S(w, l)$ *es una submalla libre que satisface las condiciones:* $w \geq \alpha$ *y* $w \geq \beta$ *suponiendo que la asignación de* $S(\alpha, \beta)$ *es solicitada, donde la asignación se refiere a seleccionar un conjunto de procesadores para una tarea de llegada.*

La figura 2.6 muestra un ejemplo de una malla 2D de tamaño 4×4, donde los procesadores asignados a alguna tarea se indican con círculos sombreados y los procesadores libres con círculos claros. Las redes en malla tienen la propiedad de ser particionables en submallas pequeñas [10, 20]. Por ejemplo, $(0, 0, 1, 1)$ representa la submalla S de tamaño 3×2 en la figura 2.6, donde $(0, 0)$ son las coordenadas de la base de la submalla y $(1, 1)$ son las coordenadas finales de ésta. Un sistema particionable tiene la ventaja de habilitar la asignación de múltiples tareas simultáneas, lo cual puede resultar en una buena utilización del procesador [20, 12]. Los tiempos de ejecución de un trabajo se pueden reducir asignando varios procesadores a las sub-tareas del mismo. Cuando se tienen muchos trabajos, la malla puede ser particionada en submallas, de esta manera cada trabajo puede ser asignado a una submalla [39]. Cuando un trabajo desocupa la malla, los procesadores que utilizaba se combinan con otros que estan desocupados (ociosos) para ser asignados a otras tareas y evitar que el sistema degrade su desempeño [11, 12]. Al proceso anterior se le denomina *Asignación de procesadores* .

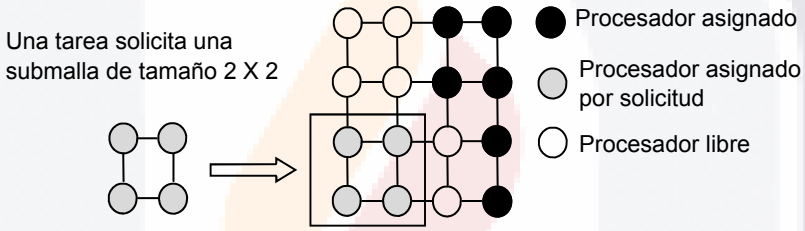

Figura 2.6: Ejemplo de una malla 2D de tamaño 4×4.

2.2.1.1. Asignación de procesadores.

La asignación eficiente de los procesadores y la planificación de tareas son dos procesos críticos si la potencia computacional de multicomputadoras de gran escala se desea aprovechar efectivamente [10, 61, 59]. El *Asignador de Procesadores* es el responsable de encontrar, seleccionar y asignar el conjunto de procesadores sobre el cual un trabajo paralelo se ejecutará, mientras que el *Planificador de Tareas* es responsable de determinar el orden en el cual los trabajos se seleccionan para su ejecución, usando una política de planificación [10, 61]. Si al momento de llegar un trabajo no puede ejecutarse inmediatamente dentro del sistema, debido a la falta de procesadores libres o a la existencia de otros trabajos que esperan por correr, se envía a la fila de espera. Una vez que los procesadores se asignan a una tarea, éstos permanecen asignados de forma exclusiva a la misma hasta que finaliza. Una vez que la tarea termina y sale del sistema, los procesadores son liberados y puestos a disposición del *Asignador de Procesadores*.

Una de las metas principales de la ejecución en paralelo es minimizar el tiempo que un trabajo espera para que le sea asignado un conjunto de procesadores libres en la malla, por lo que es importante desarrollar algoritmos con estrategias de asignación de procesadores eficientes, que minimicen los tiempos de espera de las tareas dentro del sistema de malla.

Dos han sido las estrategias de asignación de procesadores [10] que se han desarrollado, y su clasificación obedece al tipo de reconocimiento que se realiza en la malla de procesadores: las estrategias de asignación de procesadores contiguos y las estrategias de asignación de procesadores no contiguos, que se definen en las siguiente secciones.

Las estrategias de asignación de procesadores contiguos, aparece cuando se tiene un reconocimiento parcial del sistema y es posible asignar para la ejecución del trabajo, únicamente submallas contiguas de procesadores a los trabajos que lo solicitan. La figura 2.7 muestra una estrategia de asignación contigua de 4 procesadores en una malla de tamaño 4×4.

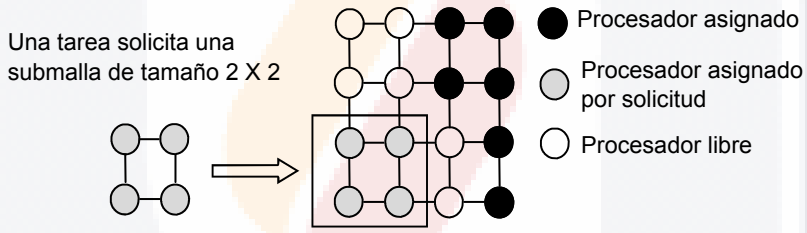

Figura 2.7: Asignación contigua de 4 procesadores en una malla 4×4.

Este tipo de estrategia implica que aún teniendo el número de procesadores libres en el sistema, no es posible asignarlos si las submallas que los contienen no están contiguas. Suponga que una tarea T solicita una submalla 2×2 como lo muestra la figura 2.8, aun existiendo el número de procesadores libres las submallas que los contienen no estan contiguas, por lo que la tarea debe ser puesta en la fila de espera del sistema, en donde permanece hasta que una submalla del tamaño solicitado se encuentre disponible. Esto provoca una fragmentación externa de 4 procesadores.

La fragmentación de procesadores puede ser de dos tipos: interna y externa [39]. *La fragmentación interna* ocurre cuando se asignan más procesadores a un trabajo de los que realmente requiere, mientras que la *fragmentación externa* ocurre cuando existen los suficientes procesadores libres para satisfacer solicitudes de asignación pendientes,

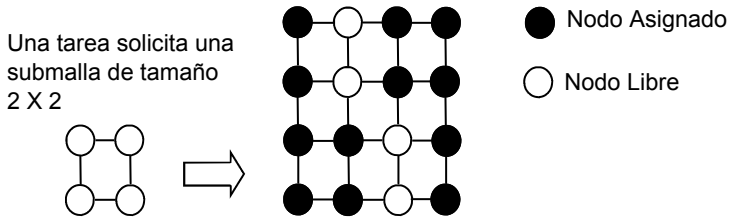

Figura 2.8: Fragmentación externa de 4 procesadores suponiendo que la estrategia de asignación es contigua.

pero no pueden ser asignados por no encontrarse contiguos; la figura 2.8 ejemplifica este tipo de fragmentación.

Las estrategias de asignación de procesadores no contiguos, surgen para eliminar las deficiencias presentadas por las asignaciones contiguas, e implican tener un algoritmo de reconocimiento total de la malla, de tal forma que si un trabajo de tamaño $n \times n$ no puede ser asignado a una submalla $m \times m$ libre, ocupe diferentes submallas libres en la malla completa, aún cuando éstas no se encuentren contiguas. La figura 2.9 muestra una asignación no contigua de procesadores en la malla.

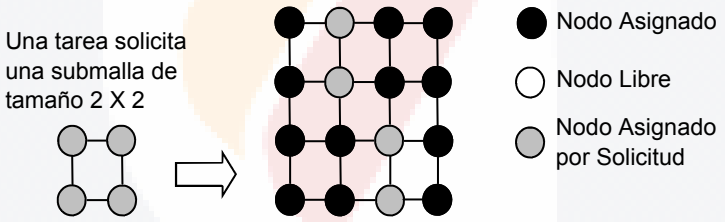

Figura 2.9: Asignación no contigua de 4 procesadores en una malla 4×4.

Para reducir la fragmentación del procesador que las asignaciones contiguas producen, se han propuesto asignaciones no contiguas [20, 41, 53, 8]. En las técnicas de asignaciones no contiguas un trabajo puede ejecutarse sobre múltiples submallas disjuntas, lo que evita una espera de una sola submalla del tamaño y forma solicitada. En la figura 2.9 si un trabajo solicita la asignación de una submalla de tamaño 2×2, la asignación contigua falla porque una submalla de esa cantidad de procesadores contigua no esta disponible. Sin embargo, los cuatro procesadores libres (dibujados con círculos en blancos) pueden ser asignados al trabajo cuando se adopta una asignación no contigua. Aunque la asignación no contigua puede incrementar la transferencia de mensajes en la red, se mejora la contigüidad para reducir la fragmentación externa del procesador e incrementar la utilidad del mismo.

Para mejorar el desempeño de un sistema de cómputo paralelo, el objetivo es buscar una estrategia que permita a los procesadores estar contiguos, y en caso de no encontrar submallas libres que satisfagan lo anterior, entonces se debe buscar que los procesadores estén lo mas cercanos posibles para evitar la sobrecarga de la comunicación en la malla; buscar submallas libres con pocos procesadores desagregados en la malla puede conducir a encontrar tareas con requerimientos de pocos procesadores en la cola de espera durante todas las planificaciones que se realicen, lo que puede provocar que tareas que requieran gran cantidad de procesadores sean relegadas hasta que existan submallas libres con un gran número de procesadores; debido a esto la estrategia debe evitar que tareas grandes caigan en inanición dentro del sistema.

Para ejemplificar lo anterior, considere la figura 2.10, dada la cola de espera con 6 tareas que esperan por ingresar, y las submallas libres desagregadas en la malla; aquí suponemos que las primeras 6 tareas T_0 a T_5 solicitan un reducido número de procesadores: 5, 4, 5, 6 y 3 respectivamente, y la tarea T_{14} solicita 29 procesadores; una vez que la planificación se realiza las primeras 5 tareas podrán ser objeto de asignación en la malla, pero dado el número de recursos de la tarea T_{14} que solicita 29 deberá esperar si las condiciones del algoritmo de planificación no establece una estrategia para atender tareas con grandes solicitudes de recursos y evitar la inanición de tareas. De no ser asignada la tarea deberá esperar a la siguiente planificación para competir por su asignación y bajo la condición de que no vuelvan a existir tareas pequeñas que compitan de manera directa con ella, porque entonces dicha tarea deberá esperar indefinidamente.

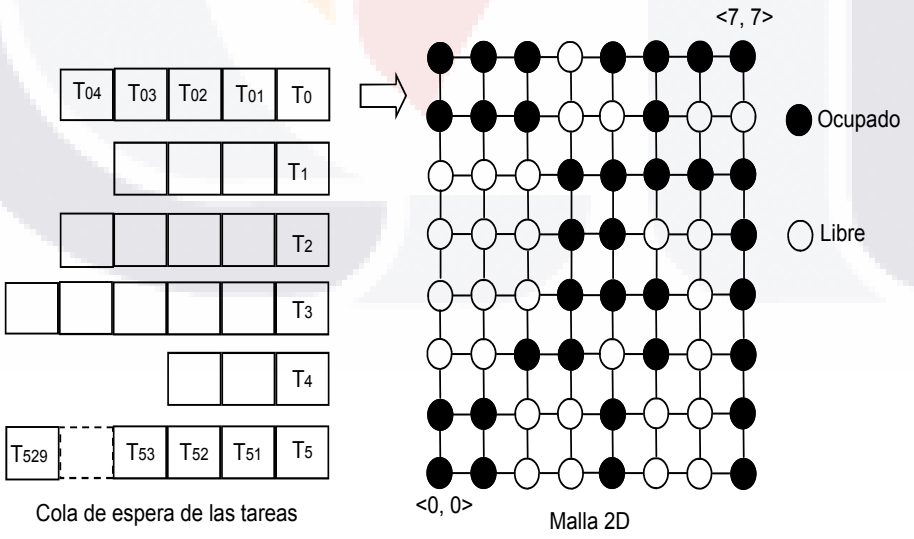

Figura 2.10: Una malla de procesadores de tamaño 8×8 con tareas asignadas y submallas libres.

En contraposición a las políticas de asignación libres de inanición, los algoritmos que planifican tareas permiten que éstas sean asignadas de una manera más rápida, debido a que en cada asignación podrán ser colocadas mas de una tarea a la vez, lo que produce que no exista una relación del número de tareas con número de asignaciones a la malla; por ejemplo, dada la cola de espera de las tareas de la figura 2.10 y considerando que ha existido planificación de tareas y el método de asignación es no contiguo, entonces en una sola asignación podrán ingresar de manera directa las primeras 5 tareas, y la tarea 14 podrá ir a la cabeza de la cola liberando de esta forma 5 espacios para que futuras tareas ingresen a la malla. Para elegir las tareas que ingresarán a la malla, es necesario un proceso de elección basado en dos criterios: ubicar la tarea considerando el número de procesadores que requiere buscando la mayor contigüidad posible y asignar el mayor número de tareas en los nodos libres.

Un método de planificación y asignación de tareas para un sistema paralelo que modele lo anterior puede disminuir el número de asignaciones que se realicen en la malla de procesadores, disminuír el tiempo de permanencia de las tareas en la cola de espera y maximizar el uso de los procesadores de la malla. De esta forma, minimizar y maximizar los objetivos en la asignación de tareas en una malla 2D, se visualiza como un problema multiobjetivo.

Una forma de abordar un problema multiobjetivo, es a través de los algoritmos metaheurísticos que establecen estrategias para recorrer un espacio de soluciones del problema, transformando de forma iterativa soluciones de partida. En un nuestra propuesta, una vez que se visualizan un conjunto de mallas libres y un conjunto de tareas posibles de asignar, se corre una meta-heurística de búsqueda para elegir iterativamente la mejor solución para llegar a una optimización en la asignación, es decir, encontrar la asignación más barata en cuanto a comunicación, contigüidad y número de procesadores que vayan a ser asignados. Las asignaciones necesarias que se deben realizar se consideran como asignaciones cuadráticas dinámicas en la malla, debido a que las combinaciones posibles se basan en el número de procesadores libres y en el número de tareas que esperan por entrar.

Ahora, para contrarrestar los riesgos que se corren al utilizar metaheurísticas de búsqueda en el problema multiobjetivo, hemos implementado una búsqueda parcial sistemática para evitar repeticiones manteniendo un alto grado de aleatoriedad. Esta búsqueda la implementamos mediante un recorrido exhaustivo e implementando un método de paro sin necesidad de llegar a completar todo el espacio de soluciones para evitar caer en óptimos locales.

La amplia adopción del encaminamiento segmentado con espera (*wormhole routing*) [20, 40], cuya característica principal es que la latencia de los mensajes sea menos sensitiva a la distancia y tengan la capacidad de moderar condiciones de tráfico pesado en sistemas prácticos, ha permitido considerar asignaciones no contiguas para multicomputadoras que usan largas distancias de comunicación.

El método que se usa para atender solicitudes de asignación particionadas tiene un impacto considerable en el desempeño de asignaciones no contiguas [20]. El proceso de particionamiento debe dirigirse para mantener un alto grado de contigüidad entre procesadores asignados a un trabajo paralelo, para que la sobrecarga de comunicación se reduzca sin afectar el desempeño total del sistema.

2.3. Cómputo evolutivo y optimización multi-objetivo

Los Algoritmos Evolutivos (EA) se han popularizado como métodos robustos y efectivos para la resolución de problemas de optimización. Tradicionalmente, los problemas abordados consideraban la optimización de una única función objetivo, pero en la última década se ha desarrollado una amplia gama de EAs para afrontar problemas con objetivos múltiples. Estos problemas cuentan con complejidades propias que los distinguen de los problemas de un solo objetivo, y por ello los EA para optimización multiobjetivo tienen características que los diferencian de los EA tradicionales.

Las técnicas de procesamiento paralelo y distribuido se aplican a los modelos clásicos de EA con el objetivo de obtener mejoras en la eficiencia computacional y perfeccionar la calidad del mecanismo evolutivo [16]. Desde la perspectiva de la eficiencia, paralelizar un EA permite afrontar la lentitud de convergencia para problemas cuya dimensión motiva el uso de poblaciones numerosas o múltiples evaluaciones de complejas funciones objetivo. Desde el punto de vista algorítmico, los modelos paralelo distribuidos de algoritmos evolutivos pueden explotar el paralelismo intrínseco del mecanismo evolutivo trabajando simultáneamente sobre varias poblaciones semiindependientes para resolver un problema.

En [37], la optimización multiobjetivo se presenta cuando un sujeto que debe decidir, ya sea una persona o una organización, tiene frente a sí un conjunto variado de objetivos, con un mayor o menor grado de compatibilidad o de contradicción entre ellos, y de los que, en cada caso, no desea ignorar un subconjunto relevante de los mismos. Es decir, ante cada situación concreta en la que haya que emitir una decisión, el sujeto o la organización contemplará ciertas consecuencias de la misma, las cuales quedarán

recogidas, al menos desde un punto de vista operativo, en una serie de funciones objetivo cuya optimización o satisfacción de modo conjunto no será, en general, posible de alcanzar.

La introducción de este esquema en el campo de la optimización matemática ha dado lugar a una rama de la misma que se denomina programación multicriterio o programación multiobjetivo.

En la programación multiobjetivo se establece que [37]:

1. La perfecta compatibilidad entre los distintos objetivos analizados, en cuyo caso el poder conseguir todos ellos, no exige plantearnos ninguna elección entre los mismos.
2. La existencia de una escala de prioridades claramente establecida para el conjunto de dichos objetivos, ya que entonces, aunque éstos no sean compatibles, bastará con la aplicación sucesiva de dicha escala hasta llegar al objetivo menos deseado que sea factible de conseguir.

En el mundo real aparecen problemas de optimización multiobjetivo (MOP de sus siglas en inglés *Multiobjective Optimization*) continuamente. Cuando nosotros queremos realizar la automatización de este tipo de problemas debemos modelarlos como MOP. Los algoritmos evolutivos multiobjetivo (*Multiobjective Evolutionary Algorithms* por sus siglas en inglés) son un campo de la computación evolutiva y una herramienta de diseño eficiente para solucionar los MOP. La popularidad de este campo de investigación es debida a que aún permanecen muchas preguntas sin resolver.

Un MOP puede ser formulado como:

Encontrar el vector $\overrightarrow{x}^* = [x_1^*, ..., x_n^*]^T$ el cual satisface las m restricciones de desigualdad:

$$g_i(\overrightarrow{x}) \geq 0, i = 1, ..., m$$

las p restricciones de igualdad

$$h_i(\overrightarrow{x}) = 0, i = 1, ..., p$$

y optimiza la función del vector

$$\overrightarrow{f}(\overrightarrow{x}) = [f_1(\overrightarrow{x}), ..., f_k(\overrightarrow{x})]^T$$

donde $\overrightarrow{x} = [x_1, ..., x_n]^T$ es el vector de las variables de decisión.

Esto es, se desea determinar de entre todos $\overrightarrow{x} = [x_1, ..., x_n]^T$ cuál satisface la desigualdad y las restricciones de igualdad anteriores, el particular $\overrightarrow{x} = [x_1, ..., x_n]^T$ el cual produce los valores óptimos de todas las k funciones objetivo del problema. Si Ω es el conjunto definido de todos los vectores $\overrightarrow{x} = [x_1, ..., x_n]^T$ que no violan las restricciones.

2.3.1. Algoritmos evolutivos para optimización multiobjetivo

La complejidad inherente a los problemas de optimización multiobjetivo plantea un difícil reto para su resolución mediante algoritmos exactos determinísticos a medida que crece la dimensión del espacio de soluciones. De este modo, las técnicas clásicas como los algoritmos enumerativos o los métodos exactos de búsqueda local, basados en gradientes o que utilizan las técnicas estándar de programación determinística, tales como los métodos ávidos (*greedy*), técnicas de ramificación y poda (*branch & bound*), etc., si bien pueden ser aplicables para problemas de complejidad reducida, exigen un costo computacional excesivo para la resolución de problemas multiobjetivo complejos con aplicación en el mundo real.

En este contexto, las técnicas heurísticas estocásticas han sido propuestas como alternativas para la resolución de problemas de optimización multiobjetivo, para alcanzar soluciones aproximadas de buena calidad en tiempos de cómputo razonables. Enmarcados en esta categoría, las técnicas de computación evolutiva se han manifestado como métodos robustos y efectivos para resolución de los problemas de optimización multiobjetivo y se han popularizado en la última década como consecuencia de su éxito.

Los Algoritmos evolutivos para optimización multiobjetivo (MOEA, por sus siglas en ingles *MultiObjective Evolutionary Algorithm*) surgen como una extensión de los EA para problemas de un solo objetivo, utilizando fundamentalmente varios conceptos relacionados con el tratamiento de funciones multimodales por parte de los EA monoobjetivo.

2.3.2. Algoritmos evolutivos

Los EA basan su funcionamiento en la simulación del proceso de evolución natural [32, 23]. Consisten en una técnica iterativa que aplica operadores estocásticos sobre un conjunto de individuos (la población) con el propósito de mejorar su aptitud (del término en ingles fitness), una medida relacionada con la función objetivo del problema en cuestión. Cada individuo de la población representa una solución potencial del problema, codificada de acuerdo a un esquema de representación, generalmente basado en números binarios o reales.

Inicialmente la población se genera de forma aleatoria, y luego evoluciona mediante la aplicación iterativa de interacciones denominadas operadores de reproducción, que incluyen recombinaciones de individuos (los cruzamientos) y modificaciones aleatorias (las mutaciones). Esta evolución es guiada por una estrategia de selección de los individuos más adaptados a la resolución del problema, de acuerdo a sus valores de aptitud.

La tabla 2.1, presenta un esquema genérico de un EA, donde puede identificarse el mecanismo evolutivo descrito y los operadores comentados.

Cuadro 2.1: Esquema de un algoritmo evolutivo.

```
Inicializar(P(0))
generacion = 0
Evaluar(P(0))
{
    mientras (no CriterioParada) hacer
        Padres = Seleccion(P(generacion))
        Hijos = Operadores de Reproduccion(Padres)
        NuevaPop = Reemplazar(Hijos,P(generacion))
        generacion ++
        P(generacion) = NuevaPop
    Finmientras

    retornar Mejor Solucion Hallada

}
```

2.3.3. Algoritmos evolutivos para optimización multiobjetivo

De acuerdo a [17], la capacidad de los EA para resolver problemas con múltiples objetivos fue sugerida en la década de 1960 por Rosenberg, pero hasta mediados de la década de 1980 no se presentó la implementación de un algoritmo evolutivo para optimización multiobjetivo [47]. A partir de la década de 1990 fueron realizadas una gran cantidad de propuestas de MOEA, formándose una comunidad de investigadores en el área que trabaja activamente en la actualidad.

Dado que trabajan en paralelo sobre un conjunto de soluciones, los EA tienen la potencialidad de tratar problemas con objetivos múltiples, hallando en cada ejecución un conjunto de soluciones aproximadas al frente de Pareto. Esto representa una importante ventaja respecto a los algoritmos tradicionales, que solamente generan una

solución por ejecución. Complementariamente, los EA tienen otras ventajas respecto a los algoritmos tradicionales, como ser menos sensibles a la forma o a la continuidad del frente de Pareto o permitir abordar problemas con espacio de soluciones de gran dimensión.

Un MOEA debe diseñarse para lograr dos propósitos en forma simultánea: lograr buenas aproximaciones al frente de Pareto y mantener la diversidad de las soluciones, para muestrear adecuadamente el espacio de soluciones y no converger a una solución única o a una sección acotada del frente. La figura 2.11 presenta gráficamente los propósitos de un MOEA, donde se ha demarcado con la región celeste (gris) el espacio de búsqueda de funciones objetivo de un problema hipotético, mientras que la línea roja (oscura) gruesa representa al frente de Pareto. El mecanismo evolutivo de los EA permite lograr el primer propósito, mientras que para preservar la diversidad los MOEA utilizan las técnicas de nichos, división (*sharing*), amontonamiento (*crowding*) o similares, utilizadas tradicionalmente por los EA en la optimización de funciones multimodales. La figura T dentro del eje X, Y presenta el esquema genérico de un MOEA, basado en [17]. Este esquema cubre la mayoría de los MOEA más comunes encontrados en la literatura, incluyendo el algoritmo NSGA-II que en los siguientes párrafos se describe en detalle.

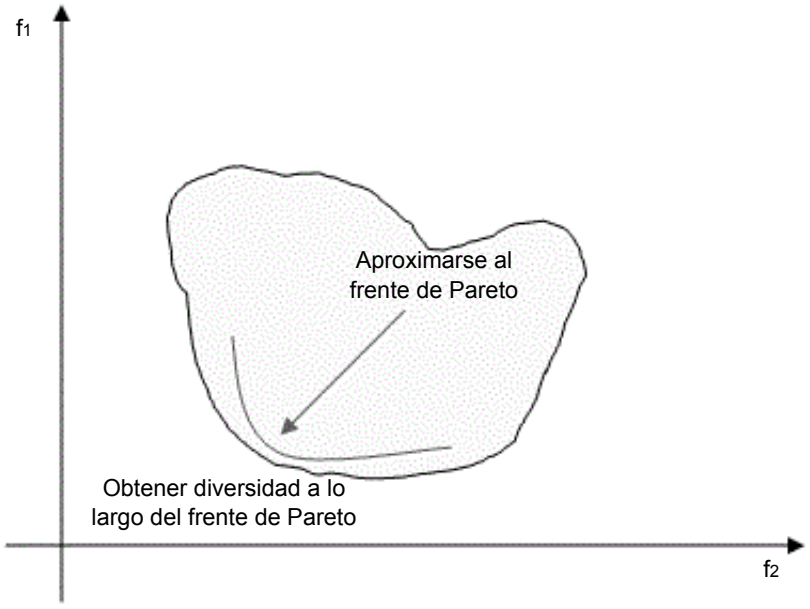

Figura 2.11: Propósitos de un algoritmo evolutivo para optimización multiobjetivo.

En la tabla 2.2, es posible observar dos operadores característicos de un MOEA, que no aparecen en la estructura genérica de un EA. El Operador de diversidad aplica la técnica utilizada para evitar la convergencia a un sector del frente de Pareto (nichos, intercambio de aptitud, amontonamiento, etc.). Asimismo, se incluye un procedimiento de asignación de aptitud, orientado a brindar mayor oportunidad de perpetuarse a aquellos individuos con mejores características, considerando los valores de las funciones objetivo y los resultados de la métrica utilizada para evaluar la diversidad.

Cuadro 2.2: Esquema de un algoritmo evolutivo para optimización multiobjetivo.

```
Inicializar(P(0))
generacion = 0
Evaluar(P(0))
{
    mientras (no CriterioParada) hacer
        Operador de diversidad(P(generación))
        Asignar aptitud(P(generación))
        Padres = Seleccion(P(generación))
        Hijos = Operadores de Reproducción(Padres)
        NuevaPop = Reemplazar(Hijos,P(generación))
        generación ++
        P(generación) = NuevaPop
        Evaluar(P(0))
    Finmientras

    retornar Mejor Solución Hallada

}
```

2.3.4. El algoritmo NSGA-II

El algoritmo NSGA-II (Non-dominated Sorting Genetic Algorithm, versión II) fue presentado por K. Deb y sus colegas del Laboratorio de Algoritmos Genéticos del Instituto Tecnológico Kanpur en India en el año 2000 [24]. Surgió como una versión mejorada del algoritmo NSGA [50], de quién heredó su estructura principal, pero incluyendo características distintivas para resolver tres aspectos fuertemente criticados en la comunidad de investigadores sobre el NSGA: el ordenamiento no−dominado, la ausencia de elitismo y la dependencia del parámetro σ para aplicar la técnica de compartición (sharing).

Las características principales del algoritmo NSGA-II abarcan:

1. El ordenamiento no-dominado elitista mediante una técnica de comparación que utiliza una subpoblación auxiliar, que le permite diminuir la complejidad de las verificaciones de dominancia de $O(MP^3)$ a $O(MP^2)$, siendo M el número de funciones objetivo y P el tamaño de la población utilizada.

2. La utilización de una técnica de amontonamiento (*crowding*) que no requiere especificar parámetros adicionales para la preservación de diversidad en la población, eliminando la dependencia de parámetros como el σ de compartición (sharing) utilizado por el NSGA original.

3. La asignación de valores de aptitud con base en los niveles o rangos de no dominancia, se hereda del NSGA-II original, aunque se considera en el procedimiento de asignación de los valores de distancia de crowding utilizados para evaluar la diversidad de las soluciones.

La tabla 2.3 presenta un esquema del algoritmo NSGA-II, basado en la descripción de [24]. Pueden apreciarse los operadores mencionados utilizados para el ordenamiento no dominado, la evaluación de la diversidad mediante la técnica de amontonamiento (*crowding*) y la asignación de aptitud.

Cuadro 2.3: Esquema del algoritmo NSGA-II.

```
Inicializar(P(0))
generacion = 0
Evaluar(P(0))
{
    mientras (no CriterioParada) hacer
        R = Padres ⋃ Hijos
        Frentes = SortingNoDominado(R)
        NuevaPop = φ
        i = 1
        mientras |NuevaPop| + |Frentes(i)| ≤ sizepop
            Calcular Distancia de Crowding (Frentes(i))
            NuevaPop = NuevaPop ⋃ Frentes(i)
            i++
        Finmientras
        Sorting por Distancia (Frentes(i))
        NuevaPop = NuevaPop ⋃ Frentes(i)[1 : (sizepop − |NuevaPop|)
        Hijos = SeleccionyReproduccion(NuevaPop)
        generacion ++
        P(generacion) = NuevaPop
    Finmientras

    retornar Mejor Solución Hallada

}
```

2.3.5. Algoritmos evolutivos paralelos

Las técnicas de procesamiento paralelo y distribuido se aplican a los modelos clásicos de EA con el propósito de obtener mejoras en la eficiencia computacional y proporcionar un mecanismo diferente de exploración del espacio de búsqueda [16]. Dividiendo la población en varios elementos de procesamiento, los Algoritmos Evolutivos Paralelos (pEA de sus siglas en ingles *Parallel Evolutionary Algorithm*) permiten afrontar la lentitud de convergencia para problemas complejos que motivan el uso de poblaciones numerosas o múltiples evaluaciones de funciones objetivo que exigen un costo computacional elevado. Conjuntamente, los pEA introducen un modelo de evolución diferente al modelo panmíctico secuencial, que posibilita explotar el trabajo simultáneo sobre poblaciones geográficamente distribuidas o localizadas de acuerdo a una estructura de organización espacial subyacente.

La organización de la población constituye el principal criterio utilizado por los investigadores para clasificar los modelos de pEA [2]. De este modo, se destacan tres grandes familias de pEA: los pEA basados en el modelo maestroesclavo, pEA de población distribuida y los pEA celulares. Las tres familias se describen a continuación.

Los pEA basados en el modelo maestro−esclavo, donde se distribuye la evaluación de la función de aptitud y se mantiene el mecanismo de evolución con interacción panmíctica que es característica de los modelos secuenciales. La figura 2.12 presenta gráficamente el modelo maestro-esclavo.

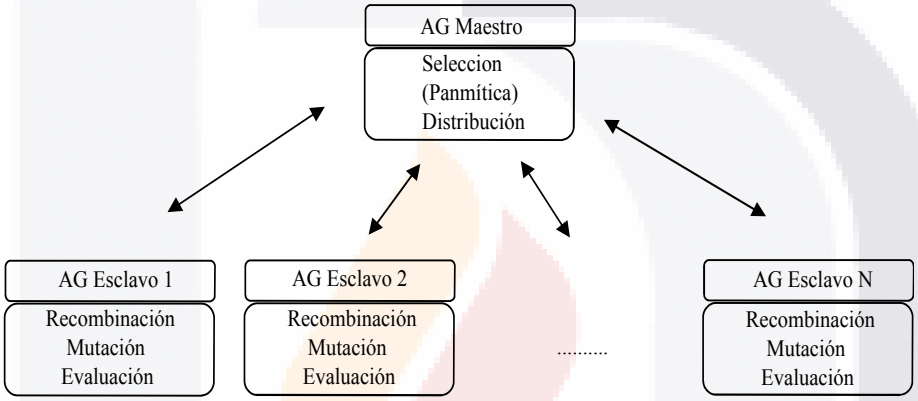

Figura 2.12: Algoritmo evolutivo paralelo modelo maestro−esclavo.

Los pEA de población distribuida, que trabajan con un conjunto de subpoblaciones independientes (islas) tienen como limitación el que las interacciones solamente son posibles entre individuos de la misma isla. Un operador adicional llamado migración posibilita intercambios ocasionales de individuos entre islas, introduciendo una nueva fuente de diversidad. La figura 2.13 presenta gráficamente el modelo de población distribuida.

Los pEA celulares, caracterizados por poseer una estructura espacial subyacente a la población y por su modelo especial de propagación de características de individuos, denominado difusión, que sigue las direcciones definidas por la topología de interconexión de los elementos de procesamiento. La figura 2.14 presenta gráficamente el modelo celular.

Estos modelos tienen diferentes variantes, y existen modelos híbridos que combinan las características de dos o más de los modelos de la categorización general presentada.

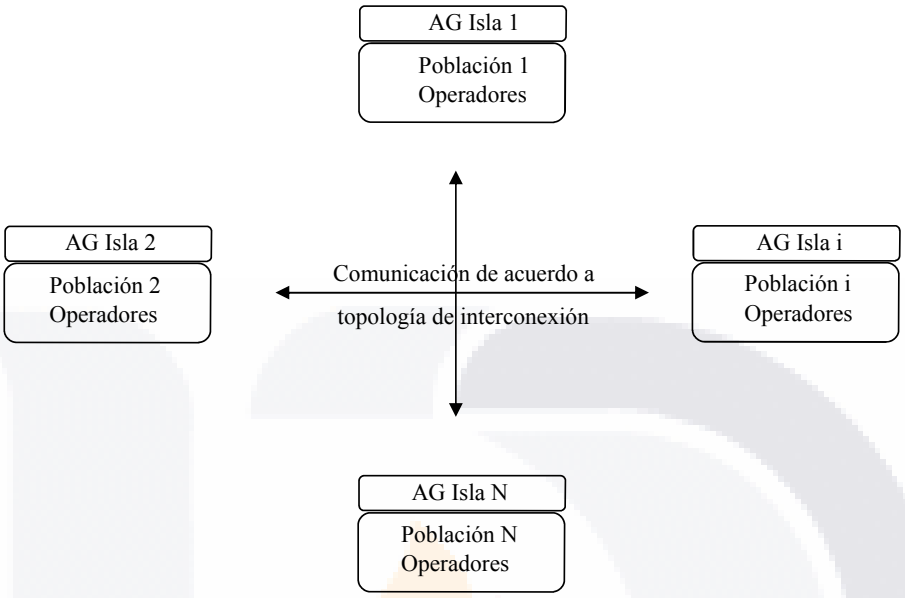

Figura 2.13: Algoritmo evolutivo paralelo, modelo de población distribuida.

En la última década, los modelos paralelos de EA se han popularizado por su eficiencia computacional y aplicabilidad para la resolución de variados problemas en diversas áreas como industria, economía, telecomunicaciones, bioinformática y otras [2].

2.3.6. Algoritmos evolutivos paralelos para optimización multiobjetivo

De acuerdo a [17], el número de propuestas de aplicación de paralelismo en el campo de MOEA es muy bajo, y el alcance de las propuestas es bastante restringido. En [57] se detalla el tema sobre paralelismo aplicado a MOEA y los autores exponen las principales consideraciones de diseño, implementación y testeo de algoritmos evolutivos paralelos para la optimización multiobjetivo.

En la sección del estado del arte se hace una referencia al uso de los algoritmos evolutivos paralelos para la optimización multiobjetivo, con la finalidad de mostrar el uso de este tipo de algoritmos, pero cabe señalar que en la literatura no se encontró una referencia a este tipo de algoritmos en la planificación y asignación de tareas en un sistema de multicomputadoras, sino en áreas totalmente diferentes.

Definición de la optimalidad de Pareto: Decimos que $\overrightarrow{x} = [x_1, ..., x_n]^T \in \Omega\Omega, \Omega\Omega \subseteq \mathbb{R}^n, f_i : \mathbb{R}^n \to \mathbb{R}$ es Pareto óptima si para cada $\overrightarrow{x} = [x_1, ..., x_n]^T$, y $I = 1, ..., k$ ambos,

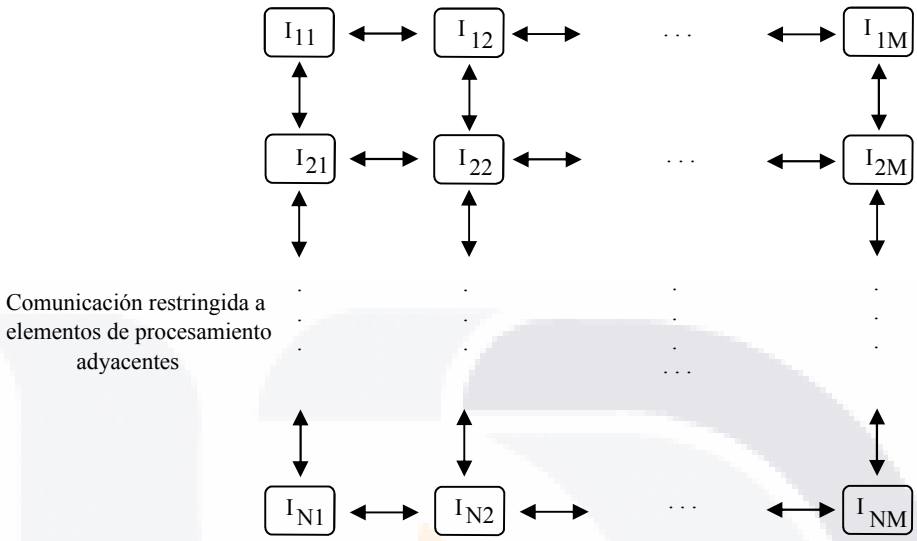

Figura 2.14: Algoritmo evolutivo paralelo, modelo celular.

$$\bigwedge_{i \in l}(f_i(\overrightarrow{x}) = f_i(\overrightarrow{x}^*))$$

2.3.7. Planteamiento de los objetivos de planificación y asignación de tareas

En esta sección se plantean los objetivos de la planificación y asignación de tareas en una malla 2D. De esta forma y tomando en consideración los objetivos propuestos en [58], se enumeran como:

1. Disminuir el número de asignaciones a la malla de procesadores que realiza el algoritmo de asignación de tareas.

2. Disminuir el tiempo de permanencia de las tareas en la cola de espera.

3. Maximizar el uso de procesadores de la malla, es decir, disminuir el porcentaje de procesadores que permanecen libres después que el algoritmo de asignación coloca una o más tareas en la malla de procesadores (fragmentación externa) [33].

4. Disminuir la inanición de tareas, es decir evitar una discriminación en la asignación de las tareas que requieren una gran cantidad de procesadores (tareas grandes), provocada porque las tareas que requieren una reducida cantidad de procesadores (tareas pequeñas) están siendo asignadas continuamente.

5. Maximizar la contigüidad entre procesadores (asignar el conjunto de procesadores libres lo mas cercanos posible) para minimizar la distancia en la ruta de comunicación y evitar la interferencia entre los mismos [56]; esto con el fin de obtener un buen algoritmo paralelo que disminuya el tiempo de comunicación y maximice el tiempo de procesamiento [54], [56].

2.3.8. Minimizar los tiempos de espera de las tareas

Minimizar los tiempos de espera de las tareas en la cola, está dado por:

$$t_{sj} = \sum_{i=1}^{s_j} t_{ij}$$

Cuya función objetivo es:

$$minTQ = \left(\sum_{i=1}^{s_j} t_{i1} + \sum_{i=1}^{s_j} t_{i2} + ... + \sum_{i=1}^{s_j} t_{ij} \right)$$

Sujeto a:
$t_i \in \mathbb{R}\ 1 \leqslant j < |J|$

2.3.9. Minimizar la inanición de las tareas grandes

Cuando las tareas son colocadas en la cola de espera de la malla el proceso gobernador que elige la tarea a ser ejecutada debe, en primera instancia, verificar si existen en la cola tareas que han permanecido durante más de 4 ciclos de asignación en espera de ser asignados a la malla de procesadores; en caso de ocurrir lo anterior, el proceso de asignación se detiene hasta que a dicha tarea le son asignados el conjunto de procesadores de la malla que solicita.

La política de asignación que no permita la inanición de tareas grandes al favorecer el ingreso de tareas pequeñas, está dada por la siguiente definición.

El algoritmo asignador dispone de un conjunto finito de procesadores libres que constituyen un grafo, en los que debe asignar el mayor número de tareas a ejecutarse, igualando el número de m^2 lugares que se encuentren libres. Donde m^2 forma un subgrafo, de modo que se cumpla que los subgrafos asignados sean conexos y consideren el número de subtareas que constituyen a una tarea dada.

Así, dado $G = (V, E)$ y dado j_k constituida por S_i, que solicita un $H = (V', A')$, en un tiempo $t + 1$ tal que:
$|V| \leqslant |V|$

2.3.10. Maximizar el uso de los procesadores en la malla

El objetivo de construír algoritmos de planificación y algoritmos de asignación de tareas en sistemas de cómputo paralelo, es mantener la máxima utilización de los procesadores de la malla por ser los recursos que facilitan la ejecución de los trabajos de los usuarios en el sistema. En un entorno paralelo se presentan: la fragmentación interna y la fragmentación externa. El objetivo de los sistemas de planificación es cuidar que el número de procesadores asignados a un trabajo para su ejecución no exceda la cantidad solicitada, para evitar que un cierto número de procesadores se encuentren libres y sin poder ser asignados a otra tarea; de igual forma evitar que existan suficientes procesadores que puedan atender a una tarea para su ejecución estén en la submalla libre, pero debido al metodo de planificación estos no puedan ser asignados.

Así, para evitar ambos tipos de fragmentación, el sistema propuesto considera el objetivo de maximizar el uso de los procesadores en la malla proponiendo un novedoso esquema de planificación.

2.3.11. Llevar a cabo una asignación cuadrática de tareas en la malla

Matemáticamente podemos formular el **Problema de asignación cuadrática (Quadratic Assignment Problem por sus siglas en Inglés QAP)** definiendo dos matrices de tamaño $n \times n$: una matrix de flujo F cuyos $(i, j)-$ésimos elementos representan los flujos entre tareas i y j y un arreglo de distancias D cuyos $(i, j)-$ésimos elementos representan la distancia entre sitios i y j. Una asignación se representa por el vector p, el cual es una permutación de los números *1, 2, ... , n* . $p(j)$ es el lugar donde la tarea j es asignada. Con esta definición, el problema de asignación cuadrática puede ser escrito como:

$$min_{p\in} \ \sum_{i=1}^{n} \ \sum_{i=1}^{n} \ f_{ij} dp(i)p(j)$$

2.3.12. Esquema de planificación de tareas y asignación de procesadores en la malla

Los esquemas de planificación y asignación de tareas que proponemos en este trabajo están basados en un planteamiento multiobjetivo. Planificar las tareas de la cola de

espera es visto como un problema de asignación cuadrática para el cual se desarrolla un algoritmo que permita una búsqueda de tareas que puedan caber dentro de la submalla libre en un tiempo t, dado por la función de minimización:

$$min\ e(\pi^t) = (d_{ij})(m_{ij}) + ... + (d_{ik,jk})(m_{ik,jk})$$

que busca por el valor mas pequeño en el conjunto de todas las posibles asignaciones en un tiempo t cuyo dominio son las posibles permutaciones de las tareas dentro de una submalla.

El primer proceso que realiza el algoritmo es identificar las tareas que quepan dentro de la submalla para obtener una poblacion incial de individuos candidatos.

En un tiempo t_0, permanecen un conjunto de tareas que esperan por entrar a la malla de procesadores; el asignador de tareas utilizando la asignación lineal permitirá asignar las primeras n tareas que quepan en la malla; cuando la tarea *n+1* no quepa en los restantes procesadores el asignador de tareas inicia el algoritmo de búsqueda para determinar la mejor disposición de las tareas seleccionadas.

2.4. Complejidad computacional

La teoría de la *complejidad computacional* proporciona herramientas para medir la dificultad de problemas abstractos, a la vez, en términos absolutos (*complejidad intrínseca* de un problema) y en términos comparativos con otros problemas (*clases de complejidad*) [45].

El objetivo fundamental de dicha teoría es la clasificación de problemas en función de la *resolubilidad algorítmica práctica* de los mismos. Para ello, se define un concepto de eficiencia o resolubilidad a través de computadoras que existen actualmente. Un *algoritmo* se dirá *eficiente* si la cantidad de recursos necesarios para su ejecución, en el caso peor, está acotada por un polinomio en el tamaño del dato de entrada. De esta manera se fija una frontera entre la resolubilidad algorítmica práctica (tratabilidad) y la no resolubilidad algorítmica práctica (intratabilidad).

¿Por qué se consideran las funciones polinómicas para establecer la línea que distingue la tratabilidad de la intratabilidad? En primer lugar porque esa clase de funciones es estable (cerrada) por ciertas operaciones importantes (suma, producto y composición de funciones).

Los problemas se clasifican en *tratables e intratables*, según sean o no resolubles de forma eficiente. De acuerdo con lo expresado anteriormente, la clase de complejidad de los problemas tratables es, precisamente la clase **P**. Los problemas computacionalmente intratables serán aquellos que no se pueden resolver algorítmicamente en tiempo polinomial; es decir, que no pueden ser resueltos por máquinas reales para instancias de tamaño razonablemente grandes.

Es fácil justificar por qué a este tipo de problemas se les denomina intratables desde el punto de vista computacional. En efecto, sea P_1 un problema tratable y P_2 un problema intratable. Sean A_1 y A_1 algoritmos óptimos que resuelven los problemas P_1 y P_2, respectivamente. Supongamos que el tiempo de ejecución de A_1 es cuadrático (es decir, $t_{A1}(n) = n^2$) y el de A_2 es exponencial (por ejemplo, $t_{A2}(n) = 2^n$). Supongamos que ejecutamos ambos algoritmos sobre una máquina que es capaz de realizar un millón de operaciones por segundo. Entonces para una entrada de tamaño 20, el algoritmo A_1 tardaría cuatro diezmilésimas de segundo, mientras que el algoritmo A_2 tardaría aproximadamente un segundo. En cambio, para una entrada de tamaño 100, la ejecución del algoritmo A_1 duraría una centésima de segundo mientras que la ejecución del algoritmo A_2 duraría aproximadamente trescientos noventa mil millones de siglos (para hacernos una idea de la cifra tan enorme que representa dicho número, basta decir que el Big Bang se estima que sucedió entre doce y quince millones de siglos).

A partir de lo anterior, el problema P_2 es intratable, inmanejable desde el punto de vista computacional.

Ahora bien, conviene advertir que se le podría poner algunos reparos a la identificación de la noción de tratabilidad (en tiempo polinomial) con el concepto de resolubilidad mecánica de forma práctica. Imaginemos que un cierto algoritmo A_3 resuelve un cierto problema en tiempo de ejecución $t_{A3}(n) = n^{10}$. Entonces dicho algoritmo se comporta peor que el algoritmo A_2 (de costo exponencial) para entradas de tamaño menor que 990. Más aún, A_3 no podría ser ejecutado sobre una máquina real para ejemplares de tamaño 10. Para tranquilizarnos un poco convendría decir que casi todos los problemas conocidos que son resolubles en tiempo polinomial tienen un algoritmo cuyo tiempo de ejecución es del orden $O(n^3)^7$, si bien la constante multiplicativa que aparece en el orden asintótico es muy grande, en algunos casos. Esto justifica la elección de la clase de funciones polinómicas como clase distinguida para definir el concepto de tratabilidad.

Ahora bien ¿qué motiva el hecho de que algunos problemas sean computacionalmente difíciles y otros sean fáciles? No siempre es sencillo decidir qué problemas son tratables

y cuáles no lo son. Más aún, existe una clase amplia de problemas de los que no sabemos si son tratables o no. El problema del camino hamiltoniano es uno de ellos.

La complejidad computacional considera globalmente todos los posibles algoritmos para resolver un problema dado. En la complejidad computacional estamos interesados en la distinción que existe entre los problemas que pueden ser resueltos por un algoritmo en tiempo polinómico y los problemas para los cuales no conocemos ningún algoritmo polinómico.

La teoría de los NP completos no proporciona un método para obtener algoritmos de tiempo polinómico, ni dice que que estos algoritmos no existan. Lo que muestra es que muchos de los problemas para los cuales no conocemos algoritmos polinómicos están computacionalmente relacionados.

2.4.1. Tiempo polinomial

Un algoritmo puede resolverse en tiempo polinomial si el total de pasos necesarios para terminar el algoritmo para una determinada entrada es $O(n^k)$ para algún entero no negativo k, donde n, es la complejidad de la entrada.

Por ejemplo, se puede utilizar una fórmula polinómica para determinar si el camino más óptimo que debe seguir un autobús que pasa por n ciudades necesita $17n2 + n$ horas, entonces el problema se puede resolver en un tiempo polinómico.

2.4.1.1. Problemas P

Los problemas tratables son aquellos para los que existe un algoritmo que resuelve el problema en tiempo polinomial. Los problemas de decisión que pueden ser resueltos en tiempo polinomial calculado a partir de la entrada por una máquina de Turing determinista son conocidos como problemas P [46].

2.4.1.2. Problemas NP

Los problemas intratables son aquellos problemas polinomiales no deterministas. Los problemas NP con tiempo polinomial no determinista (*NP nondeterministic polynomial time*) son todos los problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turnig no determinista. Cuando se trata de una máquina de Turing no determinista la clase es llamada NP [46].

Se puede extender el concepto de procedimiento mecánico admitiendo la posibilidad de que su ejecución sea no determinista en el siguiente sentido: en cada instante de una computación existe un conjunto de instrucciones que son ejecutables de tal manera que el propio procedimiento puede seleccionar cualquiera de ellas para proseguir su ejecución. Asi pues, en un cierto paso computacional no determinista puede suceder que una configuración posea más de una configuración sucesora y, en consecuencia, a partir de un cierto dato de entrada de un problema, un procedimiento mecánico no determinista puede proporcionar, de manera independiente, muchas computaciones distintas. Esto hace necesario definir lo que entendemos por resolver un problema a través de un algoritmo no determinista, así como fijar cuál es el costo en tiempo de dichos procedimientos.

La clase NP esta formada por todos aquellos problemas que se pueden resolver por algoritmos no deterministas cuyo tiempo de ejecución está acotado por un polinomio; es decir, problemas cuyas posibles soluciones pueden ser verificadas en tiempo polinomial a fin de decidir si realmente son o no soluciones correctas. Determinar que un problema pertenece a la clase NP suele ser tarea fácil: basta considerar un algoritmo ordinario de búsqueda exhaustiva que lo resuelva, y transformarlo en un algoritmo no determinista a través de una instrucciones de elección adecuadas.

Así pues, los problemas de la clase NP serían resolubles en tiempo polinomial mediante máquinas que tuvieran la capacidad de realizar en paralelo y de manera independiente, un número no acotado de cálculos. Por tanto, se verifica la inclusión $P \subseteq NP$. Ahora bien, desde el punto de vista de la eficiencia computacional ¿añade algo realmente nuevo el modo no determinista respecto del modo ordinario, que podríamos denominar determinista? es decir,

¿Es estricta la inclusión $P \subseteq NP$?

La respuesta no se conoce hoy día y, sin lugar a dudas, es una de las cuestiones abiertas más importantes a las que se enfrenta la ciencia del siglo XXI.

2.4.1.3. Problemas NP completos

Dentro de la clase NP podemos destacar una subclase de problemas que tiene especial interés: los problemas que son los mas difíciles de la clase, en el sentido de que cualquier otro problema de la clase NP, puede ser resuelto a través de él, con un costo en tiempo adicional de tipo polinomial (mediante una reducción en tiempo polinomial). Es la clase de los problemas denominados NP completos.

Capítulo 3

ESTADO DEL ARTE DE LA PLANIFICACIÓN Y ASIGNACIÓN DE PROCESADORES EN UNA MALLA 2D

3.1. Introducción

El espacio compartido puede ser usado en conjunto con el tiempo compartido en computadoras paralelas debido a la presencia de múltiples procesadores [25]. En el espacio compartido un trabajo se asigna a distintos subconjuntos de procesadores; esto es, los procesadores no son asignados concurrentemente a mas de un trabajo. En el tiempo compartido un procesador gasta un intervalo de tiempo ejecutando un trabajo y cambia a la ejecución de otro. La sobrecarga que resulta desde el contexto de intercambio en el tiempo compartido degrada el desempeño del sistema y como resultado viene a ser menos popular en sistemas prácticos [25].

Muchas estrategias de asignación emplean espacio compartido y pueden ser categorizadas como contiguas y no contiguas. En asignaciones contiguas los procesadores estan físicamente contiguos y tienen la misma topología que la red de multicomputadoras (malla) para mantener mínima la sobrecarga de comunicación entre procesadores asignados. Una consecuencia directa de la asignación contigua es que una utilización buena del sistema no puede ser llevada a cabo debido al problema de fragmentación que la asignación contigua presenta. El problema de la fragmentacion es de dos tipos: interna

y externa. La fragmentación interna ocurre cuando alguno de los procesadores asignados a un trabajo no se usa, mientras que la fragmentación externa ocurre cuando un número suficiente de procesadores libres estan disponibles para satisfacer la solicitud de un trabajo pero no pueden ser asignados debido a que no estan contiguos.

El objetivo principal de este capítulo es describir algunas de las estrategias de asignación contigua y no contigua que han sido propuestas en la literatura para multicomputadoras conectadas en malla.

3.2. Estrategias de asignación contigua para mallas 2D y 3D

La asignación contigua ha sido extensamente estudiada para multicomputadoras conectadas en malla. Muchos de los estudios previos se han enfocado en reducir la alta fragmentación de procesadores causada por la asignación contigua. A continuación se describen algunas de las estrategias más comunes que utilizan este tipo de asignación.

Sistema amigo de dos dimensiones (2DBS de sus siglas en Ingles: Two Dimensional Buddy System). La asignación 2DBS se aplica a sistemas de mallas cuadradas con una potencia de 2 en la longitud de los lados. Los procesadores asignados a las tareas forman tambien submallas cuadradas con una potencia de dos en los lados. Si una submalla de tamaño $\alpha \times \beta$ tal que $\alpha \leq \beta$, esta técnica 2DBS asigna una submalla de tamaño $s \times s$ donde $s = 2^{\lceil log_2(max(\alpha \times \beta)) \rceil}$. Por ejemplo si una tarea solicita 2 procesadores ésta se asigna a una submalla cuadrada de procesadores con una longitud igual a 2 en sus lados, lo que produce que dos procesadores esten ociosos y exista una fragmentacion interna del 50 % en la submalla asignada. Esta técnica no puede ser usada para mallas que su tamaño no corresponda a una potencia de 2.

Corrimiento del cuadro (FS por sus siglas en inglés Frame Sliding). La estrategia de cuadro corredizo se aplica a una malla de cualquier tamaño y forma, realiza una búsqueda para una asignación apropiada usando un conjunto de cuadros no solapados. FS asume que una tarea que llega solicita una submalla de procesadores de forma rectangular. Los cuadros de procesadores de la misma longitud en sus lados, se buscan de derecha a izquierda y de abajo hacia arriba. Los saltos a cuadros sucesivos son con base en el ancho y largo de la submalla solicitada. El objetivo de la búsqueda es encontrar un cuadro apropiado para la asignación; esta estrategia establece que todos los procesadores deben estar libres para la asignación y la malla debe ser lo suficientemente grande para cualquier requerimiento. Este proceso termina cuando se encuentra una

asignación apropiada o cuando todos los cuadros son recorridos y no fué encontrado un espacio apropiado de procesadores. La figura 3.1 muestra una malla de tamaño 6×5 en donde el algoritmo realiza una solicitud de una submalla 3×2. En este método todos los procesos de asignación inician con el primer procesador libre encontrado en la esquina inferior izquierda de la submalla. El ejemplo de la figura 3.1 muestra que el primer cuadro no puede ser considerado para la asignación porque existe un procesador asignado dentro del cuadro; la solicitud se recorre horizontalmente sobre la base del ancho de la submalla solicitada, lo cual provoca que el cuadro salga de la malla. El siguiente recorrido es en forma vertical desde la parte mas alta de la malla según el largo de la solicitud; en este ejemplo el cuadro de procesadores no es utilizado debido a que contiene procesadores asignados. El proceso de búsqueda continúa de forma iterativa, que para el caso de ejemplo termina sin encontrar un cuadro para la asignación. Un problema con esta estrategia es que no puede reconocer submallas libres debido a que los saltos son de acuerdo al ancho y largo del trabajo solicitado.

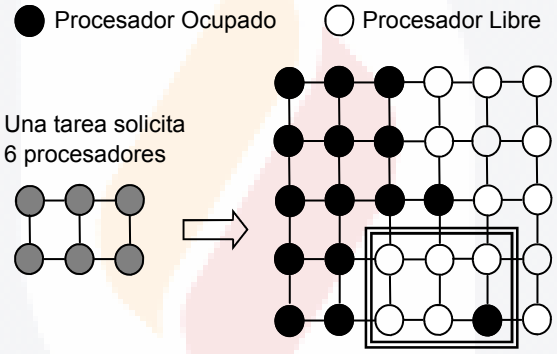

Figura 3.1: Asignación de procesadores que utiliza la estrategia cuadro corredizo.

Búsqueda adaptativa (AS por sus siglas en Inglés Adaptive Scan). Esta estrategia es un mejoramiento de FS que utiliza un procedimiento de búsqueda en lugar de una operación de corrimiento. Esto es, mueve un cuadro verticalmente con una distancia de un procesador y horizontalmente según las submallas asignadas; este método soporta la reorientación (rotación) de las solicitudes cuando la asignacion falla para la orientacion de la peticion. Una tarea que solicita una submalla $\alpha \times \beta$ puede ser asignada a una submalla $\beta \times \alpha$. Sin embargo, debido a que la distancia del movimiento del cuadro es muy corta, se incrementa significativamente el tiempo de búsqueda y asignación, lo que provoca que AS no sea recomendable para grandes sistemas paralelos.

Primer ajuste (FF por sus siglas en Inglés First Fit) y Mejor ajuste (BF por sus siglas en Inglés Best Fit) para mallas 2D. Las estrategias FF y BF solu-

cionan el problema de no encontrar una posible asignación al usar una de las estrategias anteriores. Los procesadores que pueden servir como nodos base para las submallas libres y que pueden acomodar la solicitud de trabajo actual, estan representadas por un arreglo de tamaño N, donde N es el número de procesadores en el sistema de la malla. En FF, el primer nodo base se escoge como la asignación base. En BF, una base que tiene el número mas grande de vecinos ocupados y el área libre mas pequeña se selecciona como la asignación base. Así, dada una solicitud para una submalla 2×2 y la malla mostrada en la figura 3.2 FF y BF asignarían las submallas S_1 y S_2, respectivamente. Las estrategias FF y BF pueden detectar todas las submallas libres lo suficientemente largas, pero carecen de la habilidad del reconocimiento de submallas completas, al no considerar el cambio de orientación de las solicitudes. Los algoritmos de asignación y desasignación para la estrategia FF se presentan en la tabla 3.1 y la tabla 3.2.

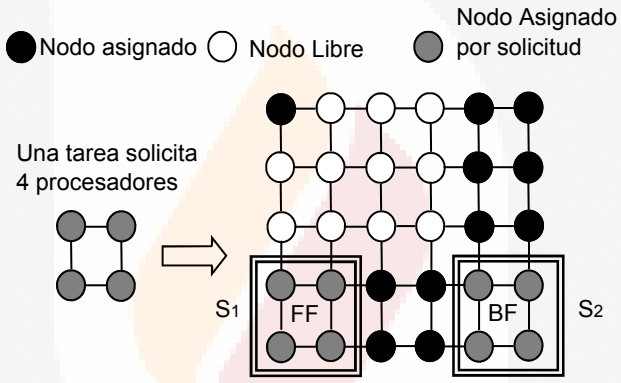

Figura 3.2: Asignación de procesadores que utiliza las estrategias primer ajuste y mejor ajuste.

Cuadro 3.1: Pseudocódigo de la estrategia de asignación contigua FF.

```
Procedure FF_Allocate (α, β, γ)
{
    W = MeshWidth; D = MeshDepth; H = MeshHeight
    MeshSize = W × D × H
    JobSize = α × β × γ
    int w_i, d_j, h_k, w_x, d_y, h_z
    int Avail; // To determine the number of processors
                  for an incoming job.
    if (JobSize > freeprocessors) return failure
    for each w_i from 0 to W − 1
        for each d_j from 0 to D − 1
            for each h_k from 0 to H − 1
                if the node (w_i, d_j, h_k) is free then
                    Avail = 0
                    for each w_x from w_i to w_i + α − 1
                        provided that w_x < W
                        for each d_y from d_j to d_j + β − 1
                            provided that d_y < D
                            for each h_z from h_k to h_k + γ − 1
                                provided that h_z < H
                                if the node (w_x, d_y, h_z)
                                                is free then Avail + +;

                    if (Avail == JobSize) {
                        for each w_x from w_i to w_i + α − 1
                            for each d_y from d_j to d_j + β − 1
                                for each h_z from h_k to h_k + γ − 1

                                    allocate the node(w_x, d_y, h_z)
                                        to the current job by
                                        setting node's ID to job ID
                        return success.

                    }
                }
                return failure

}
```

Cuadro 3.2: Pseudocódigo de la estrategia de desasignación contigua FF.

```
Procedure FF_De_Allocation ():
{
    jid = id of the departing job;
    For all nodes in the mesh system
    if (nodes'id == jid)
        de_allocate it.

}
```

Primer ajuste (FF por sus siglas en Inglés First Fit) y mejor ajuste (BF por sus siglas en Inglés Best Fit) para mallas 3D. En estas dos estrategias, se buscan las submallas libres y FF asigna la primera submalla que sea lo suficientemente extensa para contener la tarea, mientras que BF asigna la submalla apropiada mas pequeña. Los resultados de las simulaciones muestran que estas dos estrategias tienen un desempeño comparable en términos del tiempo de respuesta y efectividad de la planificación de tareas. El desempeño de FF es muy cercano al de BF. Tanto FF como BF no tienen reconocimiento completo; una solicitud de asignacion es acomodada solamente si existe una submalla lo suficientemente extensa con la misma orientacion de la solicitud, por lo que se produce una alta fragmentación externa de procesadores. Esta técnica utiliza los arreglos de bits, para la busqueda de procesadores libres en la malla.

Girando el primer ajuste (TFF por sus siglas en Inglés Turning First Fit) y girando el mejor ajuste (TBF por sus siglas en Inglés Turning Best Fit). El problema de la falta de una posible asignación existente al utilizar FF y BF se soluciona al utilizar TFF y TBF. Las estrategias TFF y TBF soportan la rotación de las solicitudes de llegada, dado que consideran todas las orientaciones de la solicitud cuando es necesario. Sean (α, β, γ) el ancho, la profundidad y el largo de una solicitud de asignación de submalla. Las seis permutaciones (α, β, γ), (α, γ, β), (β, α, γ), (γ, α, β) y (γ, β, α) se consideran para la asignación. Si la asignación es exitosa, para cualquiera de estas permutaciones el proceso se detiene. Por ejemplo, suponga una malla (3, 3, 2) y las solicitudes de tareas (2, 3, 2) y (3, 2, 1) en ese orden. El segundo trabajo no se puede acomodar hasta que es rotado a (1, 3, 2) como se muestra en la figura 3.3. Los resultados de simulaciones con esta estrategia han mostrado que TFF puede mejorar el desempeño en términos del tiempo de respuesta tras la llegada de las tareas a la cola de espera. Al cambiar la orientación de las solicitudes puede disminuír la fragmentación externa y se ha demostrado que el desempeño de TFF es casi idéntico al de TBF. En este mismo trabajo se estudian otras estrategias tales como el primero en llegar primero en salir (FIFO por sus siglas en Inglés First In, First Out) y Fuera de Servicio (OO

por sus siglas en Inglés *Out-of-Order*) para mejorar el desempeño de las técnicas TFF y TBF, pues el objetivo es evitar bloquear la cabeza de la cola de espera producida por tareas que no pueden ser servidas de inmediato.

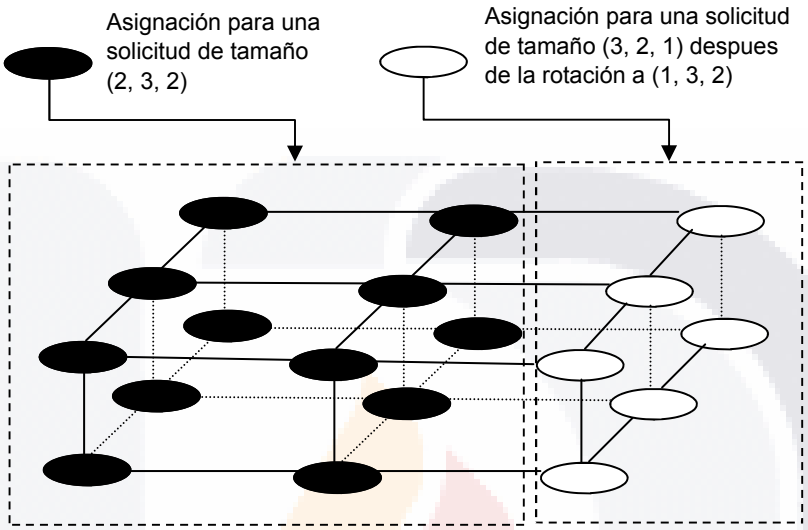

Figura 3.3: Asignación con rotación para una solicitud de tamaño (2, 3, 2) seguida por una solicitud (3, 2, 1).

Algunos métodos recientes de asignación de tareas, muestran que las técnicas propuestas inicialmente en esta área de investigación se continúan utilizando. Por ejemplo, en [7], el método FF es utilizado en conjunto con el metodo BF, de la forma siguiente: si una tarea solicita una submalla de tamaño 4×4 y la solicitud no puede ser otorgada, el tamaño de la solicitud es reducida a un múltiplo de 2, para este caso se solicita una malla de tamaño 2×2, y así sucesivamente hasta que el requerimiento es de la cantidad mínima de procesadores, para este caso 1×1. Cuando la primera técnica falla, se activa una segunda técnica que es la BF, a través de esta técnica, se realiza una búsqueda en las submallas libres pero con el mejor ajuste, es decir, con el número exacto de procesadores que la tarea requiere. Con el hecho de aplicar 2 técnicas alternas dentro del método de asignación, se busca mejorar la condición de contigüidad mediante el mantenimiento de un buen nivel de cercanía entre procesadores, que ejecutan una misma tarea, y reducir la latencia de comunicación que es causada por la no contigüidad entre procesadores.

3.3. Estrategias de asignación no contigua para mallas 2D

Los avances en las tecnicas de ruteo tales como el ruteo Wormhole, han hecho que la latencia de comunicación sea menos sensitiva a la distancia cuando existe comunicación entre los procesadores. Esto ha hecho que sea factible asignar tareas en procesadores no contiguos dentro de la malla de procesadores. La asignación no contigua permite a los trabajos ser ejecutados cuando el número de procesadores libres es suficiente para atender una solicitud aun estando éstos disagregados en la malla. A continuación se describen algunas de las estrategias de asignación no contiguas que se han desarrollado.

Aleatoria (*Random*). La asignación aleatoria es una estrategia directa, en la cual una solicitud para un número de procesadores se satisface con un número de procesadores seleccionados de manera aleatoria. Tanto la fragmentación interna como la externa se eliminan, debido a que todos los trabajos son asignados exactamente al número de procesadores solicitados, si éstos se encuentran disponibles. Debido a que ningún tipo de contigüidad existe en esta estrategia, existe una alta interferencia en la comunicación entre tareas.

Paginación (*Paging*) [39]. En la estrategia de Paginación la malla completa 2D se divide entre páginas que son submallas con una longitud en sus lados de $2^{size-index}$, donde size-index es un entero positivo. Una página es la unidad de asignación. Las páginas son indexadas de acuerdo a varios esquemas de indexación: importancia de filas (row-major), movimiento de la fila de mayor importancia (*shuffled row-mayor*), como serpiente (*snake-like*) y mezclar la indexación con forma de serpiente (*shuffled snake-like indexing*), como se muestra en la figura 3.4. Se usa una lista ordenada para mantener el contacto con todas las páginas no asignadas. Estas páginas son almacenadas en un orden incremental de sus índices, asignados por el esquema de indexación. Cada entrada en la lista contiene los correspondientes índices de renglones y columnas de las páginas y el orden de indexado de las páginas. El número de páginas que un trabajo solicita se calcula como:

$$P_{request} = [(\alpha \times \beta)/Psize]$$

donde P_{size} es el tamaño de la página y, α, β son las longitudes de los lados de la submalla solicitada. Si el número de páginas libres es mayor o igual a $P_{request}$ se quitan de la lista libre y son asignadas a la tarea solicitante. Cuando un trabajo es desasignado, las páginas desocupadas por este, se mezclan en la lista de páginas libres. Una estrategia

de paginación se identifica como $paging(size\_index)$. Por ejemplo, $paging(2)$ significa que las páginas son submallas de tamaño 4×4.

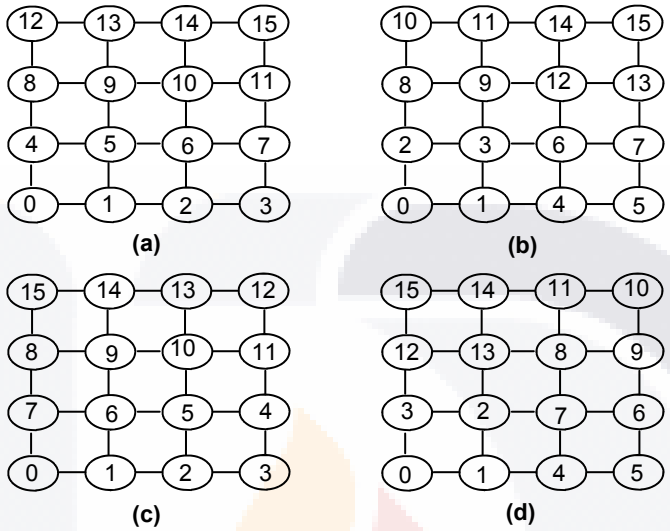

Figura 3.4: Asignación con paginación usando diferentes esquemas de indexación: (a) importancia de filas, (b) fila de mayor importancia, (c) como serpiente, y (d) indexación con forma de serpiente.

Esta estrategia produce fragmentación interna cuando $size - index > 0$. La forma de calcular la fragmentación interna de los procesadores en la malla cuando un conjunto de tareas se está ejecutando esta dada por:

$$Internal\_Fragmentation = \frac{\sum_{jobs} Lost\_Processors}{\sum_{jobs} Allocated\_Processors}$$

Donde $Lost\_Processors$ es para un trabajo paralelo que solicita Job_Size procesadores, pero es asignado a $Number\_of\_Allocated\_Pages$, que se calcula usando la fórmula:

$$Lost\_Processors = Number\_of\_Allocated \times Psize - Job_Size$$

Para ejemplificar lo anterior consideramos una estrategia de paginación con $size\_index = 1$, y una solicitud de una tarea de tamaño 3×3. Cuando se lleva a cabo la asignación se otorgan 3 páginas (12 procesadores) a la tarea en lugar de 9, de esta forma 3 procesadores quedan ociosos lo que produce una fragmentación interna del 25 %. Los algoritmos del método de paginación para la asignación y desasignación de tareas en la malla de procesadores se muestran en las tablas Tabla 3.3 y Tabla 3.4.

Cuadro 3.3: Pseudocódigo del método de asignación por paginación.

// Page_Side=2^{size_index}; Psize= $Page\_Side \times Page\_Side$
// The parameter jid is the id of the job that is being considered
// for allocation
// α and β are the side lengths of the job's allocation request
Procedure Paging_Allocation jid, α, β

Begin {
Job_Size = $\alpha \times \beta$
$P_{request} = [Job\_Size/Psize]$
Allocation:
Step 1. if (number of free pages < $P_{request}$ return failure else
go to step 2

Step 2. allocate the first $P_{request}$ pages from the list of
unallocated pages to the job,
setting the IDs of these pages to jid, and return success.
} End

Cuadro 3.4: Pseudocódigo del método de desasignación por Paginación.

// jid; id of departing job;
Procedure Paging_Deallocation (jid):
Begin {
for all allocated pages
if (page's $id == jid$)
de_allocate the page and add it to the list of unallocated pages.

} End

Estrategia amigable múltiple (MBS por sus siglas en Inglés Multiple Buddy Strategy) [39]. En MBS la malla se divide en submallas cuadradas, con lados iguales y con un potencia de 2 durante el proceso de inicialización. MBS mantiene registros de bloques libres (FBR por sus siglas en Inglés Free Block Records) para todos los cuadros de procesadores del mismo tamaño.

La entrada $FBR[i]$, contiene el número de cuadros disponibles de tamaño $2^i \times 2^1$, y una lista ordenada de las localidades en la malla de cada cuadro. El número de procesadores p , solicitados por una tarea de llegada esta representado como un número base 4 de la forma:

$$p = d_i \times 2^i \times 2^i + d_{i-1} \times 2^i \times 2^i + ... + d_0 \times 2^0 \times 2^0$$

donde los factores $d_0...d_i \in 0, 1, 2, 3$. Esta estrategia intenta satisfacer cada término i en la solicitud con d_i bloques de procesadores libres de igual tamaño: $2^i \times 2^i$ procesadores usando FBR. Si un bloque solicitado no está disponible, MBS busca por un bloque largo en FBR y repetidamente lo divide en 4 bloques adyascentes hasta que produce bloques del tamaño deseado. Los 4 bloques producidos de un bloque de tamaño $2^i \times 2^i$ son bloques $2^{j-1} \times 2^{j-1}$; si el procedimiento anterior falla, MBS divide la solicitud en un bloque de tamaño $2^i \times 2^i$ en 4 solicitudes pequeñas para bloques $2^{i-1} \times 2^{i-1}$ y repite el proceso de asignación. En este algoritmo la asignación siempre es exitosa cuando el número de procesadores libres en la malla es suficiente. Esto es porque la solicitud, o parte de ésta puede ser particionada en solicitudes de bloque 1×1. La estrategia MBS esta compuesta de cinco partes: inicialización del sistema, algoritmo de factorización de solicitudes, algoritmo generador amigable, algoritmo de asignación y el algoritmo de desasignación.

Asignación No Contigua Adaptativa (ANCA por sus siglas en Inglés Adaptive Non-contiguous Allocation) [20]. Este método primero intenta asignar un trabajo contiguamente; cuando la asignación contigua falla, la divide en dos subcuadros de igual tamaño (subsolicitudes). Por ejemplo, una tarea que solicita una submalla de tamaño 8×3 es particionada en dos subcuadros de tamaño 4×3. Estos subcuadros son asignados a submallas disponibles, en caso de ser posible, de no ser posible entonces cada uno de esos subcuadros es dividido en dos subcuadros de igual tamaño, y ANCA trata de asignar todos los subcuadros a las localidades disponibles sucesivamente. Este proceso termina cuando la asignación es exitosa para todos lo subcuadros, o el proceso se repite un número específico de veces. Más aún, la asignación falla si la longitud de un lado de la submalla alcanza 1, lo cual puede causar fragmentación externa.

Búsqueda Adaptativa y Amigable Múltiple (AS&MB por sus siglas en Inglés Adaptive Scan and Multiple Buddy). AS&MB es una estrategia híbrida. Primero intenta asignar un trabajo de forma contigua utilizando la estrategia de búsqueda adaptativa. Cuando la estrategia de búsqueda adaptativa falla, al asignar la solicitud de una tarea, emplea la estrategia de asignación no contigua MBS para la asignación. Los resultados de simulaciones muestran que el desempeño de AS&MB es casi

idéntico al de MBS en términos de los promedios de tiempo de respuesta y de servicio (el tiempo promedio que el trabajo utiliza para ejecutarse una vez que se asignan los procesadores del sistema). Sin embargo la distancia de los saltos que se realizan en AS, incrementa el tiempo de asignación por lo que AS&MB son métodos de asignación no factibles para mallas grandes.

Variantes de Paginación (PV por sus siglas en Inglés *Paging Variants*). Además de los cuatro esquemas de indexación considerados en [85], las curvas de Hilbert y el llenado de espacios (*H−Indexing Hilbert's Curves and H−indexing space−filling*) se han propuesto para el ordenamiento de procesadores en mallas 2D. En PV se han usado diferentes heurísticas. Dada una solicitud para asignar p procesadores, primero se hace un intento para encontrar un conjunto de al menos p procesadores consecutivos libres. Si esto falla, el conjunto de p procesadores con el intervalo más pequeño se asigna a la solicitud. El algoritmo del primer ajuste (FF) es quien busca los procesadores consecutivos libres, de tal forma que si encuentra el primer conjunto con el largo suficiente y además es el mejor ajuste para la solicitud esa submalla es asignada a la tarea. Los tipos de ordenamiento de procesadores que utiliza este método son: como serpiente (*snake−like*), Curvas de Hilbert y la indexación de llenado de espacio (*H−indexing space−filling*).

En las estrategias de asignación no contiguas anteriores, la estrategia aleatoria (*Random*) ignora la contigüidad de procesadores asignados a un trabajo, lo que conduce a un incremento en los tiempos de comunicación. En el método de paginación existe cierto grado de contigüidad debido a los esquemas de indexación utilizados. La contigüidad puede ser incrementada si se incrementa el parámetro $size\_index$, sin embargo se presenta la fragmentación interna de procesadores cuando $size\_index \geq 1$. MBS falla al asignar mallas contiguas aún cuando exista alguna disponible en el sistema, por ejemplo si un trabajo solicita la asignación de 16 procesadores en la malla como se muestra en la figura 3.5, la solicitud se factoriza como un número 4×4 pero debido a que no existe una submalla libre de tamaño 4×4 la solicitud se particiona en 4 solicitudes de bloques 2×2. Los 4 bloques no contiguos mostrados en la figura pueden ser asignados a la solicitud, aunque una submalla libre contigua de tamaño 2×8 mostrada en la figura, esté disponible. Podemos notar que la comunicación entre procesadores pertenecen a bloques asignados al trabajo que puede interferir en la comunicación con otros trabajos. En MBS la asignación contigua se busca explícitamente para solicitudes de tamaño 2^{2n}, donde n es un entero positivo. ANCA puede dispersar las submallas más de lo necesario y requiere que la asignación para todas las submallas ocurra en la misma iteración de la partición y la asignación, sin la posibilidad de asignar submallas grandes en una iteración previa; ANCA detiene el proceso de búsqueda y particionamiento cuando la

longitud de un lado alcanza el valor de 1, lo cual puede causar fragmentación externa. En PV se permite crear asignación con la más mínima granularidad cuando se usa la variable $size_index = 0$, que indica una unidad de asignación de un solo procesador, pero llegar a este nivel requiere que la tarea gaste un largo tiempo de espera en la cola.

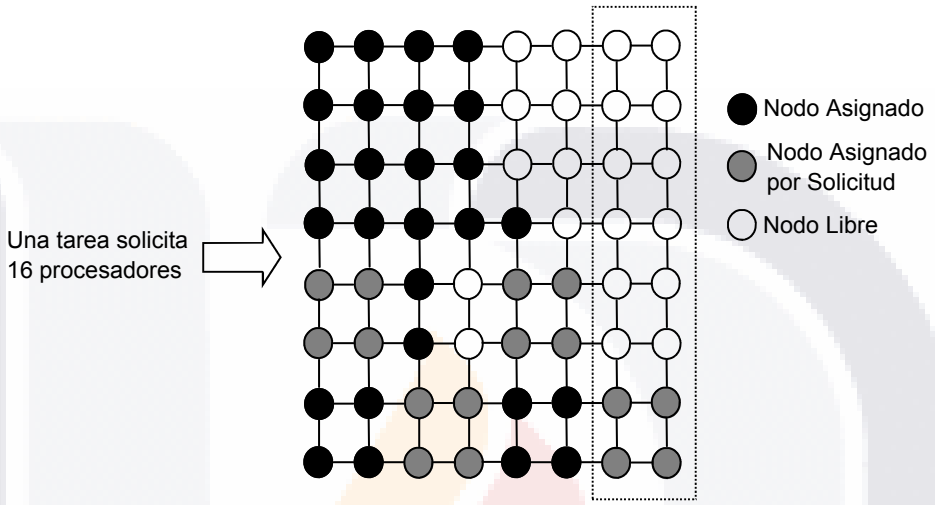

Figura 3.5: Una malla de tamaño 8×8 recibe una solicitud de asignación para 16 procesadores usando la estrategia MBS.

Capítulo 4

MÉTODO PROPUESTO

4.1. Descripción del método propuesto

El planteamiento del método propuesto, parte del supuesto que se tiene un conocimiento a priori de la medida del grado de comunicación entre la tarea principal y las subtareas que la constituyen, de todas las tareas que se encuentran en la cola de espera, así como la relación entre las mismas subtareas. Para ejemplificar la relación entre la tarea principal y sus subtareas, considere que se tiene una tarea T_1, con tres subtareas T_{11}, T_{12} y T_{13} la interacción que se puede dar entre ellas se ejemplifica en la figura 4.1 a través de líneas que muestran la transferencia de mensajes, de esta forma la tarea T_1, puede enviar y recibir mensajes de sus sub-tareas, y a su vez las subtareas podrán hacer lo mismo con la tarea principal y entre ellas mismas.

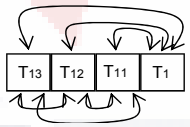

Figura 4.1: Paso de mensajes entre tareas, para una tarea con 3 subtareas.

Se tiene también una matriz de distancias simétrica entre procesadores, que especifica los saltos que un mensaje debe realizar entre un procesador y otro. Dado lo anterior el proceso de asignar los procesadores se basa en llevar a cabo un cálculo de asignación de procesadores en base a la disponibilidad de los mismos en la malla y de las tareas que permanecen en la cola de espera. Esta asignación cuadrática permite determinar a través de los cálculos y de la frecuencia de aparición de cada tarea en cada procesador, cual es la tarea o tareas que producen la mejor asignación que representa la solución

más factible. El seudocódigo para la búsqueda de tareas en la cola de espera se muestra en la tabla 4.1.

Cuadro 4.1: Pseudocódigo para la búsqueda de tareas en la cola de espera.

El algoritmo de asignación informa de los procesadores libres en la malla
Poblacion Inicial = Lista de tareas en la malla

Repite ($(Procesadores\_libres\_en\_la\_malla > 0)$ or
(Total de tareas en la malla = Tareas Verificadas))
Selecciona de manera aleatoria una tarea de la cola.
Verifica si la tarea no ha sido seleccionada al menos una vez.
Si (el número de procesadores que requiere la tarea <=
procesadores libres en la malla)
Si (Primer tarea seleccionada)
Crea la lista de tareas seleccionadas.
Si no
Adiciona la tarea a la lista de las tareas seleccionadas.
Decrementa el número de procesadores libres en la malla.
Si no
Selecciona otra tarea.
Suma 1 a la variable Tareas Verificadas.
Fin Repite

Informa la lista de tareas seleccionadas al algoritmo asignador.

Población Inicial. Para arrancar el algoritmo, la población inicial es la lista de tareas que están en la cola de espera; este conjunto de tareas se genera de manera aleatoria utilizando dos vertientes:

- Si se usan cargas pesadas para ingresar al sistema de multicomputadoras.
- Si se usan cargas livianas para ingresar al sistema de multicomputadoras.

Una carga pesada se establece cuando el número de tareas esta por encima de un promedio de trabajos previamente establecido en el Sistemas Operativo. Para nuestras pruebas en el sistema objetivo, se ha establecido una carga pesada cuando el número de tareas es mayor o igual a 5000 y cuando el número de subtareas de las tareas que se

generan excede la mitad del número de procesadores que el sistema paralelo contiene. Por ejemplo, si:

$$\text{Number\_Tasks} = 5600;$$
$$\text{Number\_of\_Subtasks} = 64;$$

habremos establecido que el sistema procesará como máximo 5600 tareas y el número de subtareas por tarea podrá ser como máximo 64, lo que representa la totalidad de procesadores del Sistema Paralelo en cuestión. En sistemas reales las cargas pesadas se procesan en horarios de 10:00 PM a 5:00 AM, que son los periodos de tiempo que establecen los administradores de los sistemas para evitar que durante el día, los trabajos con mucha carga computacional interfieran en trabajos con menos demanda y que requieren respuestas rápidas por parte de los usuarios.

Selecciona de manera aleatoria una tarea de la cola. Del número de tareas que se encuentran en la cola, se elige al azar una tarea n independientemente del número de subtareas que contenga; dicha tarea será verificada en un paso posterior para detectar si cabe en alguna submalla, submallas o en la totalidad de los procesadores libres.

Verifica si la tarea no ha sido seleccionada al menos una vez. La forma para evitar que una tarea sea seleccionada dos veces en el paso anterior, es verificar si la tarea en cuestión ya está registrada en la lista de tareas seleccionadas; la forma de llevar a cabo lo anterior es recorrer la lista de tareas verificando el número seleccionado, con el número de tarea de cada nodo de la lista que tiene registrado. En caso de ser encontrada, dicha tarea se desecha y se elige al azar otra tarea. En el caso de que el número de tareas seleccionadas corresponda con el número de tareas en la cola de espera, se detiene el proceso y continua el siguiente paso del algoritmo; En caso de que el número de procesadores libres se haya agotado por solicitud, tambien se detiene la selección al azar.

Comparación entre el número de procesadores que requiere la tarea y los procesadores libres en la malla. Este punto de verificacion permite al algoritmo verificar si la tarea puede ser asignada (cabe) a la malla de procesadores; si el número de procesadores libres restantes en la malla, independientemente de su ubicación, es suficiente para que la tarea pueda ser ingresada, entonces el número de procesadores se disminuye en la misma cantidad que el número de subtareas mas uno y la tarea es adicionada en la lista de tareas seleccionadas. En caso de no ser suficientes, entonces se elige una nueva tarea. Este proceso se repite hasta que se hayan agotado los procesadores libres o todas las tareas hayan sido seleccionadas al menos una vez.

Después que el algoritmo planificador construye la lista de tareas seleccionadas, el algoritmo asignador recibe la lista de tareas que serán puestas en la malla. El algoritmo de asignación se muestra en la tabla 4.2 y se explica en cada una de las siguientes secciones.

Cuadro 4.2: Seudocódigo del algoritmo de asignación.

```
Recibe la lista de tareas seleccionadas

Repite hasta encontrar un costo total mínimo de asignación
    Generar la asignación
    Evaluar la población
    Estimar el modelo probabilístico
    Salvar el mejor individuo
Fin repite

Ordena las soluciones
Informa la mejor solución y asigna las tareas en la malla
Verifica los procesadores libres en la malla
Informa de los procesadores libres en la malla al algoritmo de planificación
```

En los siguientes párrafos se describe cada una de las partes del algoritmo, y en la siguiente seccion se muestra un ejemplo de la funcionalidad del método propuesto con los algoritmos.

Recibe la lista de tareas seleccionadas. El algoritmo asignador recibe la lista de tareas selecionadas para iniciar el proceso de ubicar cada tarea en la malla. La lista de tareas esta compuesta por el número de la tarea, las subtareas que contiene la tarea, el tiempo total de ejecución de la tarea compuesto por el tiempo de cada una de las subtareas y el de la tarea principal, así como también el número de procesadores que solicita.

El ciclo de repetir hasta encontrar un costo total mínimo de asignación, es un proceso gobernador para los cuatro siguientes pasos principales del algoritmo y, finaliza cuando el modelo probabilístico encuentra un costo menor de asignación.

Generar la asignación. Una vez que se recibe la lista de tareas candidatas a ser asignadas, se ejecuta el procedimiento para generar las asignaciones en la malla de procesadores, el cual permite ubicar las tareas en diferentes submallas de procesadores.

Evaluar la población. Al realizar la asignación de un conjunto de tareas en la malla, se realiza el cálculo del costo que implica asignar de manera definitiva esas tareas para su ejecución en dichos procesadores; una ejecución del algoritmo para calcular la asignación cuadrática se realiza en esta parte del algoritmo.

Estimar el modelo probabilístico. Una vezque se han ralizado las evaluaciones de las poblaciones generadas para la asignación de las tareas en los procesadores libres, el algoritmo procede a estimar el modelo probabilístico. En esta parte se utiliza el modelo probabilístico más simple en el que todas las variables que describen el problema son independientes; dado lo anterior el algoritmo procede a calcular las frecuencias de aparición de una tarea en cada una de las celdas vacías de la malla en un tiempo t, de una parte de la población creada y que se considera como los mejores individuos

Ordenar las soluciones. Este proceso permite determinar, cual es la mejor solución,, que para realizar nuestros experimentos, es el valor menor de todos los obtenidos en cada uno de los objetivos evaluados.

Informa la mejor solución y asigna las tareas en la malla. Una vez obtenida la solución más barata en el paso previo, el algoritmo asignador coloca las tareas en la mallaen donde permanecen hasta su finalización.

Informa de los procesadores libres en la malla al algoritmo de planificación. Para llevar a cabo este paso, el algoritmo asignador realiza un censo de los procesadores libres, en caso de encontrarlos desocupados, el número de procesador y su ubicación son almacenados en el nodo de la lista que se crea para mantener un informe de todos los nodos libres. Este proceso por cuestiones de tiempo no se ejecuta continuamente, esperará ciertos ciclos de reloj para volver a detectar los procesadores libres en la malla. El número de ciclos que el algoritmo esperará depende del tipo de cargas que el sistema esté ejecutando.

4.1.1. Ejemplificación del método propuesto

En el siguiente ejemplo se muestra el funcionamiento del método. En un tiempo t se tiene una malla de procesadores de tamaño 4×4 cuya matriz de estado se muestra en la figura 4.2 donde 1 representa los procesadores ocupados que fueron asignado a una tarea en un tiempo $t-1$ y un 0 representando los procesadores libres que no han sido asignados a alguna tarea o sub-tarea, las distancias simétricas entre procesadores están dadas en la Tabla 4.3, la tabla 4.4 representa la cola de espera conteniendo 4 tareas pendientes de ejecución, dichas tareas esperan por ejecutarse en la malla y dos matrices de costos de comunicación representadas en la tabla 4.5 y la tabla 4.6 de las tareas T_1, T_2, T_3 y T_4 con sus subtareas y entre las subtareas mismas.

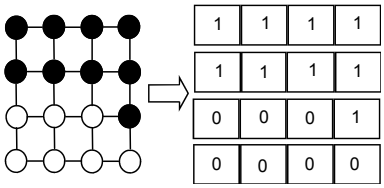

Figura 4.2: Representación en una matriz de una malla de procesadores de tamaño 4×4.

Cuadro 4.3: Distancia simétrica entre procesadores de la malla de tamaño 4×4 de la figura 4.2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	1	2	3	1	2	3	4	2	3	4	5	3	4	5	6
2	1	0	1	2	2	1	2	3	3	2	3	4	4	3	4	5
3	2	1	0	1	3	2	1	2	4	3	2	3	5	4	3	4
4	3	2	1	0	4	3	2	1	5	4	3	2	6	5	4	3
5	1	2	3	4	0	1	2	3	1	2	3	4	2	3	4	5
6	2	1	2	3	1	0	1	2	2	1	2	3	3	2	3	4
7	3	2	1	2	2	1	0	2	3	2	1	2	4	3	2	3
8	4	3	2	1	3	2	1	0	4	3	2	1	5	4	3	2
9	2	3	4	5	1	2	3	4	0	1	2	3	1	2	3	4
10	3	2	3	4	2	1	2	3	1	0	1	2	2	1	2	3
11	4	3	2	3	3	2	1	2	2	1	0	1	3	2	1	2
12	5	4	3	2	4	3	2	1	3	2	1	0	4	3	2	1
13	3	4	5	6	2	3	4	5	1	2	3	4	0	1	2	3
14	4	3	4	5	3	2	3	4	2	1	2	3	1	0	1	2
15	5	4	3	4	4	3	2	3	3	2	1	2	2	1	0	1
16	6	5	4	3	5	4	3	2	4	3	2	1	3	2	1	0

Cuadro 4.4: Cola de espera con 4 tareas pendientes de ejecución en un tiempo t.

T_1	T_{11}	T_{12}	T_{13}
T_2	T_{21}	T_{22}	
T_3	T_{31}	T_{32}	T_{33}
T_4	T_{41}		

Cuadro 4.5: Matriz de costos de comunicación para las tareas T_1, T_2.

	T_1	T_{11}	T_{12}	T_{13}	T_2	T_{21}	T_{22}
T_1	0	3	0	3	0	0	0
T_{11}	2	0	1	4	0	0	0
T_{12}	0	1	0	2	0	0	0
T_{13}	3	5	3	0	0	0	0
T_2	0	0	0	0	0	1	3
T_{21}	0	0	0	0	2	0	4
T_{22}	0	0	0	0	4	3	0

Cuadro 4.6: Matriz de costos de comunicación para las tareas T_3, T_4.

	T_3	T_{31}	T_{32}	T_{33}	T_4	T_{41}
T_3	0	1	3	2	0	0
T_{31}	1	0	1	2	0	0
T_{32}	4	5	0	1	0	0
T_{33}	2	5	2	0	0	0
T_4	0	0	0	0	0	3
T_{41}	0	0	0	0	2	0

4.2. Contraposición de los objetivos durante la planificación y asignación de tareas

El problema principal para la utilización eficiente de procesadores en los sistemas multiusuario en malla dinámicas, es la planificación de recursos computacionales [1,

22, 30]. Tanto el planificador como el asignador, buscan hacer un uso eficiente de los recursos computacionales en un sistema de multicomputadoras durante el proceso de planificación y asignación de tareas a la malla, lo cual involucra un conjunto de objetivos contrapuestos que se buscan maximizar o minimizar [33]. El problema de la asignación de tareas en un sistema de multicomputadoras se puede abordar en dos niveles: a nivel de tarea o a nivel de programa [30, 33]. En este trabajo consideramos la asignación a nivel de tarea (asignar un procesador a una tarea o una subtarea) y utilizamos una estrategia de asignación no contigua para detectar los objetivos tanto en la planificación como en la asignación de tareas. Una vez detectados los objetivos, el siguiente paso es detectar cuales se contraponen entre sí cuando se pretende encontrar la mejor solución al problema. En los siguientes párrafos se describe la forma en que los objetivos se contraponen.

La Figura 4.3 ejemplifica un conjunto de 6 tareas en la cola, que esperan por ingresar a una malla de procesadores de tamaño $< 8 \times 8 >$, los procesadores ocupados se muestran en círculos rellenos y los procesadores libres con círculos sin relleno. Para ejemplificar la contraposición de los objetivos 1 y 2: disminuir el número de asignaciones a la malla de procesadores que realiza el algoritmo de asignación de tareas y disminuir el tiempo de permanencia de las tareas en la cola de espera, consideramos que en un tiempo t el asignador informa de 29 procesadores libres, con este dato el planificador determina que: el conjunto de las 5 tareas T_0, T_1, T_2, T_3 y T_4 son candidatas para ocupar 21 procesadores en la malla, o asignar la tarea T_5 que requiere 26 procesadores y la tarea T_4 que solicita 3. Asignar el conjunto de 5 tareas libera el mismo número de posiciones en la cola para el ingreso de nuevas tareas, disminuyendo así el número de accesos a la cola para buscar tareas, permite que un mayor número de usuarios y trabajos sean atendidos y, se disminuye el tiempo de espera de las tareas en la cabeza de la cola; pero de manera contrapuesta, se genera una fragmentación externa de 8 procesadores y se incrementa la inanición de tareas en un ciclo, al no ser atendida una tarea que requiere un gran número de procesadores. Ahora, si se asigna las tareas T_4 y T_5 no se produce inanición ni fragmentación externa, pero menos tareas nuevas pueden ser aceptadas en la cola.

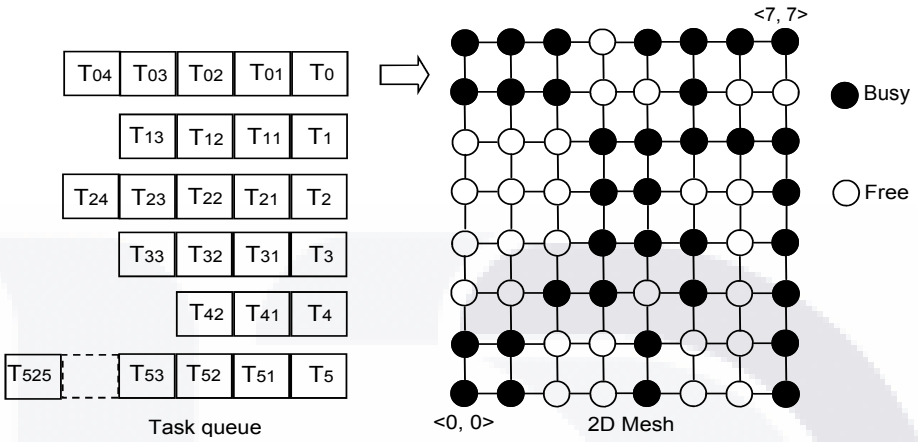

Figura 4.3: Estructura del sistema para la ejecución de tareas en un sistema de multicomputadoras en malla 2D.

Para ejemplificar la contraposición entre los objetivos 3 y 5, consideramos el mismo ejemplo descrito arriba. Así, utilizando el metodo propuesto en [10] y deduciendo el comportamiento del algoritmo, despues de realizar la busqueda del costo mas bajo de comunicación y la asignación cuadrática, el conjunto de las 5 tareas se asigna en procesadores contiguos, de la siguiente forma: la tarea T_0 se asigna en la submalla $< 4,0 >< 5,2 >$ sin considerar el procesador en la posicion $< 4,2 >$, la tarea T_1 se asigna en la submalla $< 2,0 >< 3,1 >$, la tarea T_2 se asigna en la submalla $< 0,5 >< 2,6 >$ sin considerar el procesador en la posición $< 2,5 >$, la tarea T_3 se asigna en la submalla $< 0,2 >< 1,3 >$, y la tarea T_4 se asigna en la submalla $< 6,3 >< 7,4 >$ sin considerar el procesador en la posicion $< 7,4 >$. Si el metodo propuesto detecta el incremento de inanición en el sistema, la tarea T_5 será asignada a la malla junto con la tarea T_1 o la tarea T_3 que se selecciona para ocupar la totalidad de los procesadores, después de una búsqueda en la cola y evitar la fragmentación externa; de esta forma los 29 procesadores libres de la malla permaneceran ocupados. Si se contraponen los objetivos 3 y 5 podemos deducir que al asignar la tarea T_1 o T_3 y la tarea T_5 se maximiza el uso de procesadores en la malla, pero se minimiza la adyacencia entre los procesadores y en contraposición si se asigna el conjunto de las 5 tareas se maximiza la adyacencia entre procesadores, pero se produce una fragmentación externa de 8 procesadores.

Mantener un control estricto de tareas que quepan en la malla para cumplir los objetivos propuestos, produce búsquedas exhaustivas en la cola y cálculos para ubicar

en la mejor posición las tareas en la malla [9]. Más que buscar las mejores posiciones de las tareas en la malla, se debe realizar un análisis de los objetivos que se buscan optimizar porque al tratar de localizar submallas del tamaño que las tareas requieren, con el único objetivo de estar contiguos sin considerar una evaluación de otros objetivos, puede conducir a resultados pobres en los tiempos de respuesta y en el desempeño del sistema.

En este trabajo presentamos un análisis del frente de Pareto de los diferentes objetivos propuestos y con los resultados obtenidos en [10], en donde se presentan los resultados de la evaluación de cada uno de los objetivos con diferentes cargas de trabajo en una malla de tamaño $< 16 \times 16 >$; para realizar dicha evaluación hemos establecido una escala de prioridades para el conjunto de los objetivos propuestos, de tal forma que si estos no son compatibles basta con la aplicación sucesiva de dicha escala, hasta llegar al objetivo menos deseado que sea factible de conseguir.

La escala de prioridades la establecemos de acuerdo a la revisión de los trabajos propuestos con anterioridad, clasificándolos por el tipo de asignación o planificación que el método realiza.

4.2.1. Generación de una primera asignación

Para generar la primera asignación se toma la matriz de estado de la malla haciendo una elección aleatoria de las tareas tomadas de la cola de espera y cuyo número de procesadores que solicitan están disponibles en las submallas libres. Para realizar la elección aleatoria se genera un número aleatorio tomando como base el número de tareas en la cola; si la tarea elegida cabe en la submalla o submallas libres, entonces se considera para calcular su asignación y costo; en caso contrario se elige otra tarea.

Cada individuo de la población está representado por una asignación de las tareas y sus subtareas en los espacios libres como se muestran aglutinados en la tabla 4.7. Esta representación se denomina matriz de asignación de tareas o Matriz de Estado en el tiempo t; de esta manera generaríamos una población inicial aleatoria de tamaño 2, constituida por las tareas T_1 y T_2.

Cuadro 4.7: Matriz de asignación de tareas según la matriz de estado de la malla en el tiempo t, que representa una primera solución del problema de asignación cuadrática dinámica.

1	1	1	1
1	1	1	1
T_{11}	T_{12}	T_{21}	1
T_1	T_{13}	T_2	T_{22}

Para la obtención del primer valor de la función objetivo, se realiza el cálculo del costo de la asignación para cada tarea sobre la base de los costos de comunicación entre tareas y las distancias entre procesadores, considerando el paso de mensajes de un procesador a otro y viceversa. Al considerar el paso de mensajes entre procesadores se debe calcular el costo de transferencia de los mismos, del origen al destino y viceversa, es decir, para el caso de ejemplificación la tasa de transferencia de la tarea T_1 a la subtarea T_{11} es diferente de la subtarea T_{11} a la tarea T_1, aunque puede ocurrir que ambos pesos sean iguales; los valores de las distancias entre procesadores permanecerán iguales. Así los valores a calcular están dados en las operaciones mostradas en la tabla 4.8 para la tarea T_1, y en la tabla 4.9 para la tarea T_2. Los totales de los respectivos individuos se suman para obtener el costo total de la solución, un total de 35 para la tarea T_1 que aparece en tabla 4.8 y 17 para la tarea T_2 que aparece en la tabla 4.9.

La representación de los cálculos anteriores está representado por:

C_{ij} es el costo de comunicación de i y j.

Donde:
i es la tarea o la subtarea de la tarea.
j es la tarea o la subtarea de la tarea.
C_{ij} es la distancia entre procesadores donde es asignada la tarea con sus subtareas.

Cuadro 4.8: Cálculo del costo de transferencia de mensajes para la tarea T_1.

$T_1 \rightarrow T_{11}$	$T_{11} \rightarrow T_1$	$(3+2)*1$	5
$T_1 \rightarrow T_{12}$	$T_{12} \rightarrow T_1$	$(0+0)*2$	0
$T_1 \rightarrow T_{13}$	$T_{13} \rightarrow T_1$	$(3+3)*1$	6
$T_{11} \rightarrow T_{12}$	$T_{12} \rightarrow T_{11}$	$(1+1)*1$	2
$T_{11} \rightarrow T_{13}$	$T_{13} \rightarrow T_{11}$	$(4+5)*2$	18
$T_{12} \rightarrow T_{13}$	$T_{13} \rightarrow T_{12}$	$(2+3)*1$	5
		TOTAL	35

Cuadro 4.9: Cálculo del costo de transferencia de mensajes para la tarea T_2.

$T_2 \rightarrow T_{21}$	$T_{21} \rightarrow T_2$	$(1+2)*1$	3
$T_2 \rightarrow T_{22}$	$T_{22} \rightarrow T_2$	$(3+4)*1$	7
$T_{21} \rightarrow T_{22}$	$T_{22} \rightarrow T_{21}$	$(4+3)*1$	7
		TOTAL	17

4.2.2. Generación de la segunda asignación

La segunda asignación se genera al producir una nueva asignación en la submalla libre y que corresponde al asignar las tareas T_3 y T_4 como lo muestra la tabla 4.10. Se realiza el cálculo del segundo valor de la función objetivo de la misma forma que el anterior; los valores están dados en la tabla 4.11 para la tarea T_3, y en la tabla 4.12 para la tarea T_4.

Cuadro 4.10: Matriz de asignación de tareas según la matriz de estado de la malla en el tiempo t, que representa una segunda solución del problema de asignación cuadrática dinámica.

1	1	1	1
1	1	1	1
T_{31}	T_{33}	0	1
T_3	T_{32}	T_4	T_{41}

Cuadro 4.11: Cálculo del costo de transferencia de mensajes para la tarea T_3.

$T_3 \rightarrow T_{31}$	$T_{31} \rightarrow T_3$	$(1+1)*1$	2
$T_3 \rightarrow T_{32}$	$T_{32} \rightarrow T_3$	$(3+4)*1$	7
$T_3 \rightarrow T_{33}$	$T_{33} \rightarrow T_3$	$(2+2)*2$	8
$T_{31} \rightarrow T_{32}$	$T_{32} \rightarrow T_{31}$	$(1+5)*2$	12
$T_{31} \rightarrow T_{33}$	$T_{33} \rightarrow T_{31}$	$(2+5)*1$	7
$T_{32} \rightarrow T_{33}$	$T_{33} \rightarrow T_{32}$	$(1+2)*1$	3
		TOTAL	39

Cuadro 4.12: Cálculo del costo de transferencia de mensajes para la tarea T_4.

$T_4 \rightarrow T_{41}$	$T_{41} \rightarrow T_4$	$(3+2)*1$	5
		TOTAL	5

4.2.3. Generación de la tercera y cuarta asignación

La generación de la tercera y cuarta asignación mostradas en la tabla 4.13 se produce al asignar a la malla las tareas T_2 y T_4 cuyos valores obtenidos a través de los cálculos en la primera y segunda generación son aplicados en ésta, para obtener el tercer valor de la función objetivo. En la cuarta generación se asignan las tareas T_2 y $T-3$ a la malla, para obtener un cuarto valor de la función objetivo. La asignación producida se muestra en la tabla 4.14.

Cuadro 4.13: Matriz de asignación de tareas según la matriz de estado de la malla en el tiempo t, que representa una tercera solución del problema de asignación cuadrática dinámica.

1	1	1	1
1	1	1	1
0	0	T_{21}	1
T_4	T_{41}	T_2	T_{22}

Cuadro 4.14: Matriz de asignación de tareas según la matriz de estado de la malla en el tiempo t, que representa una cuarta solución del problema de asignación cuadrática dinámica.

1	1	1	1
1	1	1	1
T_{31}	T_{33}	T_{21}	1
T_3	T_{32}	T_2	T_{22}

En nuestro ejemplo hemos mostrado un caso en el que todos los procesadores aparecen totalmente adyacentes, pero en caso de encontrarse disjuntos, el proceso para calcular las distancias es el mismo.

4.2.4. Evaluación de la población.

Una vez que se han calculado los costos de las asignaciones en la tabla 4.8, tabla 4.9, tabla 4.11 y tabla 4.12. Los totales obtenidos se suman para hacer la evaluación de un individuo mediante el valor obtenido en la función objetivo. El pseudocódigo para la evaluación de la población se muestra en la tabla 4.15.

Cuadro 4.15: Seudocódigo para la evaluación de la población.

Evaluación de la población

Repite desde $Tasks = 1$ hasta M Tareas que caben en la malla
    Realiza la suma de los totales obtenidos en cada iteración
        de los costos de transferir mensajes entre la tarea y sus subtareas
Fin repite

Reporta el valor total de la función objetivo

4.2.5. Estimar el modelo probabilístico

Como se explicó en la sección anterior, en esta parte se utiliza el modelo probabilístico más simple, en el que todas las variables que describen el problema son independientes, por tanto, calculamos las frecuencias de aparición de cada una de las tarea en cada celda vacía de la malla en el tiempo t de una parte de la población que son los mejores individuos, mediante una selección por truncación y el porciento de la truncación. Para este caso las frecuencias de aparición pueden mostrarse en la tabla 4.16, en la fila uno

se muestran las posiciones de cada uno de los procesadores libres en la malla, en tanto que, en la columna uno se muestra la lista de las tareas con sus respectivas subtareas, en las intersecciones $(fila, columna)$ se muestra la fecuencia de aparición de la tarea con el procesador. Por ejemplo:

$$[T_{22}, P(0,3)] = 3$$

nos muestra que la tarea T_{22} durante la fase de asignación aparece asignada al procesador $P(0,3)$ en tres ocasiones. La última fila de la tabla muestra la sumatoria de las veces que aparecen las tareas en los procesadores.

El Seudocódigo para estimar el modelo probabilístico se muestra en la tabla 4.17.

Cuadro 4.16: Frecuencias de aparición de cada tarea en cada celda.

	T_3	T_{31}	T_{32}	T_{33}	T_4	T_{41}
T_3	0	1	3	2	0	0
T_{31}	1	0	1	2	0	0
T_{32}	4	5	0	1	0	0
T_{33}	2	5	2	0	0	0
T_4	0	0	0	0	0	3
T_{41}	0	0	0	0	2	0

Cuadro 4.17: Seudocódigo para estimar el modelo probabilístico.

Estimar el modelo probabilístico

Repetir para asignar tareas $= 1, 2, ..., M$ hasta el final de la tabla que contiene las tareas que pueden ser asignadas a las submallas libres
Verificar que la siguiente posición sea asignada a cada tarea,
contabilizar en 1 cada asignación y almacenarlo en la Matriz de la estimación del modelo probabilístico
Fin repetir

Reportar los valores obtenidos en las frecuencias de asignación

4.2.6. Guardar el mejor individuo

Este proceso es llevado a cabo al tomar de cada población generada el mejor individuo en el momento en que se ordena la población actual. El proceso de ordenamiento de menor a mayor se genera para obtener la minimización de la asignación.

Una vez que la población ha sido evaluada y se determina el mejor individuo, el algoritmo de asignación coloca el conjunto de tareas en la malla de procesadores, en donde permanecen hasta su terminación.

El siguiente paso, es determinar el porcentaje de fragmentación externa producida por la la asignación; para llevar a cabo el cálculo se realiza una verificación en la lista de procesadores libres y se utiliza la fórmula:

$$FE = \frac{Numero\_de\_procesadores\_libres}{Total\_de\_procesadores\_en\_la\_malla} \times 100$$

El resultado se utiliza para las estadísticas de las ejecuciones realizadas.

Un siguiente paso es realizar un tiempo de espera (Dummy_Loop) en el sistema para permitir que las tareas prosigan su ejecución (las tareas que permanecían en la malla), y la asignadas recientemente inicien su ejecución. Una vez finalizado el Dummy_Loop, el algoritmo asignador verifica la lista de tareas en ejecución para detectar las tareas que han finalizado su tiempo de ejecución.

El algoritmo para la verificación de los tiempos de terminación de las tareas se muestra en la tabla 4.18 y cada uno de los procedimientos se describen a continuación.

Cuadro 4.18: Pseudocódigo para la verificación de los tiempos de terminación.

```
Verifica tiempos terminación

Si (Lista_tareas_en_ejecucion! = NIL)
Repetir hasta final de la lista de tareas en ejecucion
    N = Tarea_de_la_lista
    Si (Tiempo_transcurrido_del_sistema ≥ Tiempo_ejecucion_tarea_N)
        Asigna la tarea en la lista de estado terminado
    Fin_si
Fin repetir
```

Verificar los tiempos de terminación de las tareas en ejecución. Este procedimiento lleva un control del tiempo transcurrido en el sistema y el tiempo de cada tarea en ejecución, ambos tiempos deben ser verificados durante un periodo de tiempo para determinar si alguna de las tareas ha finalizado su ejecució. Este procedimiento accede a la estructura de datos de cada tarea que se crea cuando inicia la ejecución; la conformación de la estructura de datos que contiene toda la infromación se muestra en la tabla 4.19, en donde se enlistan los identificadores y una descripción detallada de éstos.

Cuadro 4.19: Estructura de datos que identifica a una tarea en ejecución dentro del sistema.

Identificador	Descripción
Name	Es el identificador general de la tarea denominado como: TAREA o SubTarea
Task	Es un identificador numérico único asignado a cada tarea cuando ingresa a la cola de espera del sistema
Subtask	Es un identificador numérico único asignado a cada subtarea cuando ingresa a la cola de espera del sistema
Processor	Es el número de procesador que se le asigna a la tarea o subtarea dentro del sistema, este campo permanece con un valor de 0, hasta que el sistema determina cual número de procesador contendrá
NumSubtasks	Es el número de subtareas que contiene la tarea principal
time	Es el tiempo que la tarea requerirá para su ejecución dentro del sistema, en caso de ser proporcionado; si el sistema no es proporcionado por el usuario, entonces el sistema va incrementando este tiempo hasta que la tarea indique su finalización mediante un mensaje al algoritmo de verificación de los tiempos de terminación
TimeOut	Almacena el tiempo de salida de la tarea del sistema y es registrado una vez que la tarea anuncia su salida o el sistema la retira cuando se ha cumplido el tiempo de terminación
Wait_Loops	Es un indicador que verifica el sistema durante cada asignación para todas las tareas de la cola de espera; cuando el sistema realiza una asignación de un conjunto de tareas, este indicador se incrementa en uno, si durante cierto número de asignaciones el sistema detecta que una o mas tareas no han sido asignadas, detiene el proceso de planificación y asigna a aquellos trabajos que empiezan a caer en inanición.
nodo_subtask $*sgte$	Es el indicador ancla que permite moverse de un nodo a otro dentro de la lista de tareas que permanecen en la cola de espera

Ciclo para la finalización y extracción de tareas del sistema, dentro de este ciclo se considera la primera tarea para ser retirada de la malla, se verifica el tiempo transcurrido del sistema contra el tiempo de ejecución de la tarea, en caso de que el tiempo del sistema ya sea mayor o igual al tiempo que la tarea ha estado en ejecución, se inserta la tarea en la lista de tareas de estado terminado. Toda la información relacionada con la ejecución de la tarea es extraída de la estructura de datos propia de la tarea y que fue creada cuando la tarea ingresó al sistema; dicha información es almacenada para llevar un control de la lista de tareas que finalizaron su ejecución dentro del sistema.

Una vez que se ha creado la lista de tareas en estado terminado, se procede a la liberación de los procesadores de la malla, y se procede a informar al algoritmo planificador de todos los procesadores asi como también de la ubicación de los mismos. El seudocódigo para estos procesos se muestra en la tabla 4.20.

Cuadro 4.20: Pseudocódigo para la liberación de los procesadores en la malla.

```
Verifica la lista de estado terminado

Si (Lista_estado_terminado_tareas! = NIL)
Repetir hasta final de la lista de tareas en estado terminado
    N = Tarea_de_la_lista
    Verifica procesadores asignados a la tarea N
    Procesadores_libres++
Fin repetir
Informa el número de procesadores libres al algoritmo de planificación
```

El algoritmo se estructura de la siguiente manera:

Verificar la lista de estado de terminado, este procedimiento se ejecuta para detectar cuales tareas fueron detectadas como finalizadas y que fueron puestas en el estado de terminado, una vez que pasaron por el estado de listo y el estado de ejecución en el sistema; en caso de no encontrar tareas en la lista, el algoritmo finaliza este proceso y prosigue al siguiente paso, en caso contrario inicia el recorrido nodo a nodo de la lista.

El ciclo para recorrer la lista de tareas en estado de terminado, se inicia cuando el algoritmo asignador recibe el indicativo por parte del algoritmo planificador, de que al menos una tarea ha finalizado su tiempo de ejecución en el sistema. De cada tareas que permanecen en esta lista se verifican los procesadores que le fueron asignados en la malla, se libera cada uno de ellos y se incrementa el contador de los procesadores libres. Finaliza este procedimiento informando del número de procesadores libre al algoritmo de planificación.

Cabe hacer notar que antes de iniciar el recorrido de la lista de las tareas en ejecución para detectar las tareas que han finalizado su tiempo de ejecución, el algoritmo asignador verifica si alguna tarea ha informado de su terminación, en caso de ser así, se incluye en la lista de tareas con estado de terminado.

4.3. Segundo planteamiento del método propuesto

En esta sección se describe un segundo planteamiento, con el cual se logró un mejoramiento de los tiempos de respuesta del sistema (en relación con los resultados obtenidos en las primeras experimentaciones), a las tareas que solicitan su ingreso a la malla de procesadores, así como también obtener el frente de Pareto de los objetivos involucrados.

La primera fase de experimentación consistió en la evaluación de cada uno de los objetivos propuestos, y la comparativa con dos sistemas de asignación de tareas: curvas de Hilbert y la asignación lineal; en esta experimentación los resultados obtenidos fueron mejores en 4 de los objetivos propuestos; pero no así en el objetivo de inanición de tareas. La segunda fase de experimentación propuesta, fue desarrollada para visualizar un comparativo de los objetivos contrapuestos y así, determinar las relaciones existentes al momento de la ejecución de las tareas en el sistema. En esta experimentación durante la fase de evaluación de cada uno de los objetivos, se crea el frente de Pareto para encontrar la mejor asignación posible. Los resultados obtenidos de las experimentaciones con el segundo planteamiento del método propuesto se describen en la sección 5.

4.3.1. Algoritmos del segundo planteamiento del método propuesto

Esta sección presenta, y describe los 5 algoritmos propuestos para el segundo planteamiento del método propuesto:

- Algoritmo de inicio del sistema de planificación y asignación de tareas.
- Algoritmo de identificación de submalla libre.
- Algoritmo de preselección.
- Algoritmo de evolutivo UMDA.
- Algoritmo para generar la matriz de distribuciones.

Cada uno de los algoritmos propuestos se explica en las siguientes secciones.

4.3.2. Algoritmo de inicio del sistema de planificación y asignación de tareas

La figura 4.4 muestra el algoritmo principal para el funcionamiento del segundo planteamiento del método propuesto. Este algoritmo divide su funcionamiento en 7 secciones principales:

- Inicio del sistema.
- Algoritmo de identificación de submalla libre.
- Algoritmo de preselección.
- Algoritmo Evolutivo UMDA.
- Evaluación de los objetivos.
- Evaluación del frente de Pareto.
- Algoritmo de asignación

Este algoritmo se constituye por dos ciclos principales de ejecución: uno interno y otro externo. A través del ciclo interno, se permite a las generaciones estar en constante evolución para determinar la mejor asignación a la malla de procesadores. El ciclo externo permite procesar la totalidad de las tareas que se encuentran en la cola, definido en los parámetros como: el número total de tareas en la carga.

A continuación se describen cada una de las 7 secciones que lo conforman.

Inicio del sistema.

Esta sección, establece los parámetros globales de ejecución, tales como:

- El número total de tareas en la carga. Que se define como el número completo de tareas que el sistema procesará.
- El número de procesadores en la malla. La totalidad de los procesadores para la ejecución de las tareas.
- El número de tareas en la cola.
- El número de subtareas por tarea.
- La cota de tiempo de espera para una tarea.

Estos párametros permiten al usuario arrancar el sistema.

¿Existen tareas en la cola de espera?

En esta sección se establece una estructura condicional para la verificación de la existencia de tareas en el sistema, y permite finalizar, en caso de que todas las tareas hayan sido procesadas, o continuar la ejecución con las tareas sin procesar.

Llama al algoritmo de identificación de submalla libre.

Esta sección del algoritmo, recibe como entrada la malla de procesadores en la cual realiza un censo, procesador por procesador para verificar si se encuentra ocupado o libre, en caso de encontrarlo libre, la posición y el número de procesador es almacenado en la submalla de procesadores libres. El proceso de verificación y almacenamiento se ejecuta hasta que la totalidad de la malla es verificada. El diagrama de flujo para este proceso se muestra en la figura 4.5.

Algoritmo de preselección.

El algoritmo de preselección permite crear una sublista de tareas formada por una preselección aleatoria de tareas de la carga del sistema. Una vez que la tarea se preselecciona, se verifica que quepa en la submalla libre; en caso positivo se coloca en la sublista de tareas, y en caso contrario, la tarea se rechaza.

Los criterios de paro para este algoritmo son:

- Cuando la totalidad de tareas ha sido seleccionada de la cola de espera, y éstas ya han sido puestas en la sublista de tareas.
- Cuando la totalidad de los procesadores libres ya ha sido preasignada y no pueden ser preseleccionadas más tareas.

El algoritmo de preselección de tareas puede ser visto en la figura 4.6.

Algoritmo evolutivo UMDA para evolucionar asignaciones dada una sublista.

Entrada: Sublista de tareas. Salida: Mejor preasignación de la sublista.

Algoritmo evolutivo UMDA.

El algoritmo evolutivo UMDA que muestra en la figura 4.7, en su primera fase genera una población inicial de diferentes preasignaciones de la sublista de tareas en la submalla de procesadores libres.

La segunda fase evalúa la población de preasignaciones, según la función cuadrática dada en la definicion 8 y almacena los costos en un arreglo multidimensional, para enseguida ordenar la población en un orden ascendente y poder seleccionar la mejor preasignación hasta ése momento del algoritmo.

De la población obtenida, se selecciona un porcentaje de la población para que en el siguiente paso se realice el cálculo de la matriz de distribuciones; el algoritmo para el cálculo de la matriz de distribuciones se muestra en la figura 4.8

En la fase final de este algoritmo se muestrea la nueva población usando la matriz de distribuciones. El criterio de paro para este algoritmo se establece cuando la generación de las poblaciones a concluído.

La salida del algoritmo es la mejor preasignación para la sublista dada.

Evaluar los 4 objetivos restantes.

Esta parte del algoritmo obtiene los valores para los objetivos de:

- las tareas que fueron postergadas para su ejecución: mide la inanición de tareas
- el porcentaje de procesadores que no fueron utilizados en la asignación: la fragmentación interna,
- el tiempo promedio de espera de las tareas que permanecen en la malla,
- el número de asignaciones realizadas hasta el momento.

Todos los valores son almacenados en un tabla para posteriormente realizar la comparación de los valores obtenidos.

Verifica la mejor preasignación.

Este proceso compara el valor obtenido de la preasignación, con los valores obtenidos con anterioridad, (excepto en la primera asignación por no contar con los valores previos), en caso de obtener una solución cuyo valor se considera no dominado, se incluye en el frente aproximado de Pareto.

Cumplimiento de la condición de criterio de paro.

Esta condicional permite detener la ejecución del algoritmo, que indica haber obtenido el criterio de paro: el porcentaje de evaluaciones solicitadas se ha realizado con éxito.

Algoritmo de asignación.

Es el proceso responsable de asignar las tareas en la malla de procesadores, una vez que se ha obtenido el frente aproximado de Pareto.

4.3.3. Algoritmo de identificación de submalla libre

Este algoritmo recibe como entrada el estado de la malla en un tiempo t, su núcleo principal recorre la malla de procesadores para detectar las posiciones de los procesadores que se han utilizado en las asignaciones posteriores.

Las posiciones libres que han sido localizadas, se almacenan de manera temporal en una estructura de datos, que es accesada posteriormente en la fase de asignación de las tareas.

El algoritmo finaliza al entregar la submalla libre de la malla. El diagrama de flujo para este algoritmo se muestra en la figura 4.5

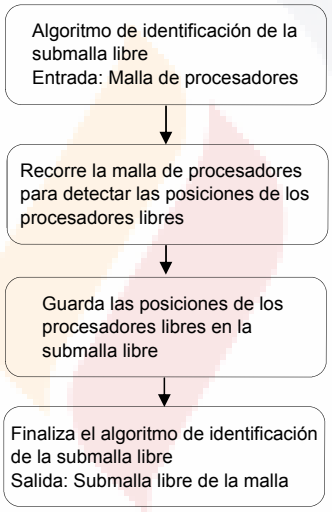

Figura 4.5: Algoritmo de identificación de submalla libre.

4.3.4. Algoritmo de preselección

Una vez que se recibe como entrada el número de procesadores libres en la malla, inicia un ciclo de búsqueda en la cola de espera de la forma: se preselecciona una tarea en la cola de espera en forma aleatoria y realiza una comparación del número de subtareas que la tarea seleccionada tiene; una condicional permite realizar la verificación del número de procesadores requeridos, contra el número de procesadores libres en la malla. La bifurcación para éste, establece que si es positivo se decrementa el número de procesadores libres en la malla, y la tarea es preseleccioada junto con sus subtareas. En caso de ser negativo el proceso de preselección se obtiene una nueva tarea.

Cuando una tarea es preseleccionada, se verifica si es la primera tarea; en caso de ser así, se crea la sublista de tareas preseleccionadas; en caso de no serlo únicamente se incrementa en 1 el número de tareas preseleccionadas.

Las condiciones de paro, para éste algoritmo son cuando el número de procesadores libres en la malla es cero, ó cuando el total de tareas en la cola de espera fueron preselecciondas. Se alcanza la finalización del algoritmo cuando se obtiene la sublista de tareas preseleccionadas.

El diagrama de flujo para el algoritmo de preselección se muestra en figura 4.6.

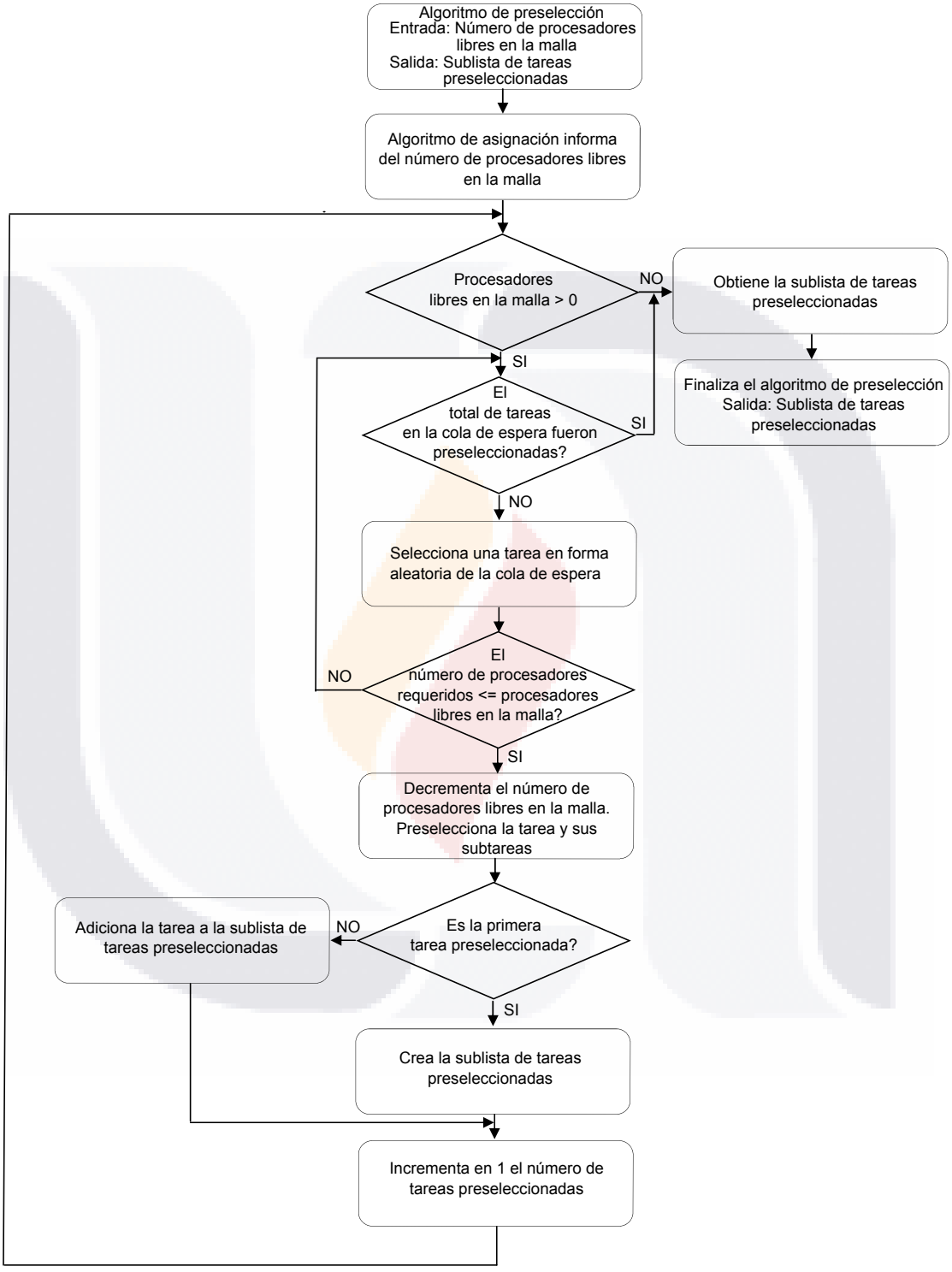

Figura 4.6: Algoritmo de preselección.

4.3.5. Algoritmo evolutivo UMDA para evolucionar las asignaciones.

Es la parte fundamental para determinar la mejor sublista preseleccionada de tareas, que será asignada a la malla de procesadores. Una vez que se obtiene una sublista de tareas, ésta es considerada como sublista candidata a ser asignada y colocada dentro de la población inicial de preasignaciones, para de esta manera obtener la población de preasignaciones.

Dada la población de preasignaciones se ejecuta un ciclo para realizar la evaluación utilizando la función cuadrática planteada en la sección 2.3.5.

La siguiente fase del algoritmo plantea un ordenamiento de la población evaluada de mayor a menor según su valor obtenido en la función cuadrática. Esto permite seleccionar un porcentaje de la población evaluada que se considerará posteriormente. Con base en el porcentaje seleccionado, procedemos a realizar el cálculo de la matriz de distribuciones, la cual nos permite almacenar, de cada generación, la distribución de probabilidad conjunta a partir de los individuos seleccionados.

El siguiente paso, es obtener una nueva población, la cual se muestra usando la matriz de distribuciones. Todos los pasos anteriores son repetidos, hasta que el criterio de paro se cumple: el número de generaciones otorgado al algoritmo se alcanza.

Este algoritmo produce como salida la mejor preasignación para la sublista de tareas dada. El diagrama de flujo del algoritmo evolutivo UMDA para evolucionar las asignaciones se muestra en la figura 4.7.

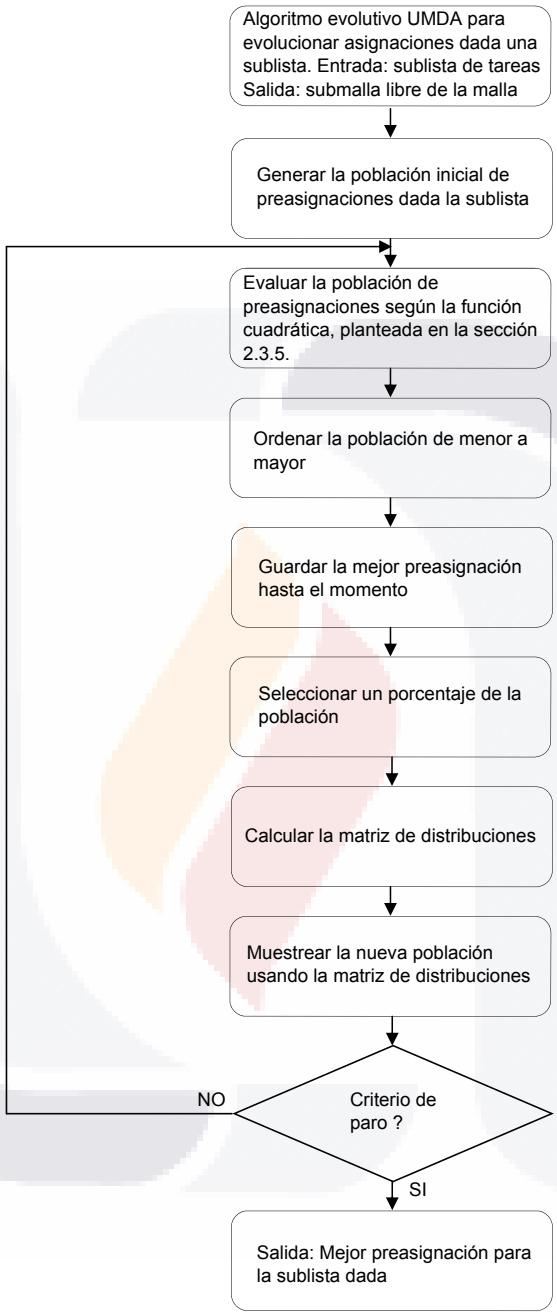

Figura 4.7: Algoritmo evolutivo UMDA.

4.3.6. Algoritmo para generar la matriz de distribuciones para las mejores preasignaciones de una sublista.

Una vez que el algoritmo UMDA determina los mejores k individuos, se inicia el proceso de la generación de la matriz de distribuciones para las mejores pre asignaciones de una sublista.

A través del condicional de la verificación de inexistencia de preasignaciones a la matriz de distribuciones, se produce un ciclo para:

- recorrer la malla de procesadores con la preasignación a realizar, lo que permitirá generar una solución.
- contabilizar las asignaciones de las tareas o subtareas en las posiciones de la malla.
- asignar cada valor en la matriz de distribuciones de las tareas a los procesadores y,
- adicionar en uno las preasignaciones evaluadas.

Este proceso iterativo se realiza hasta que ya no existen preasignaciones que deban ser adicionadas a la matriz de distribuciones; cuando esto ocurre, se genera la matriz de distribuciones por renglón y se muestrea la nueva población a partir de la matriz de distribuciones por renglón.

El condicional para verificar la existencia de las posiciones en la matriz de distribución sin evaluar, permite generar un ciclo iterativo que divide cada asignación de tarea o subtarea entre el total de las pre asignaciones evaluadas. Finalmente, cuando se ha realizado la totalidad de evaluaciones se realiza una llamada al algoritmo UMDA para enviar la matriz de asignaciones para su evaluación.

El diagrama de flujo para generar la matriz de distribuciones para las mejores pre asignaciones de una sublista, se muestra en la figura 4.8.

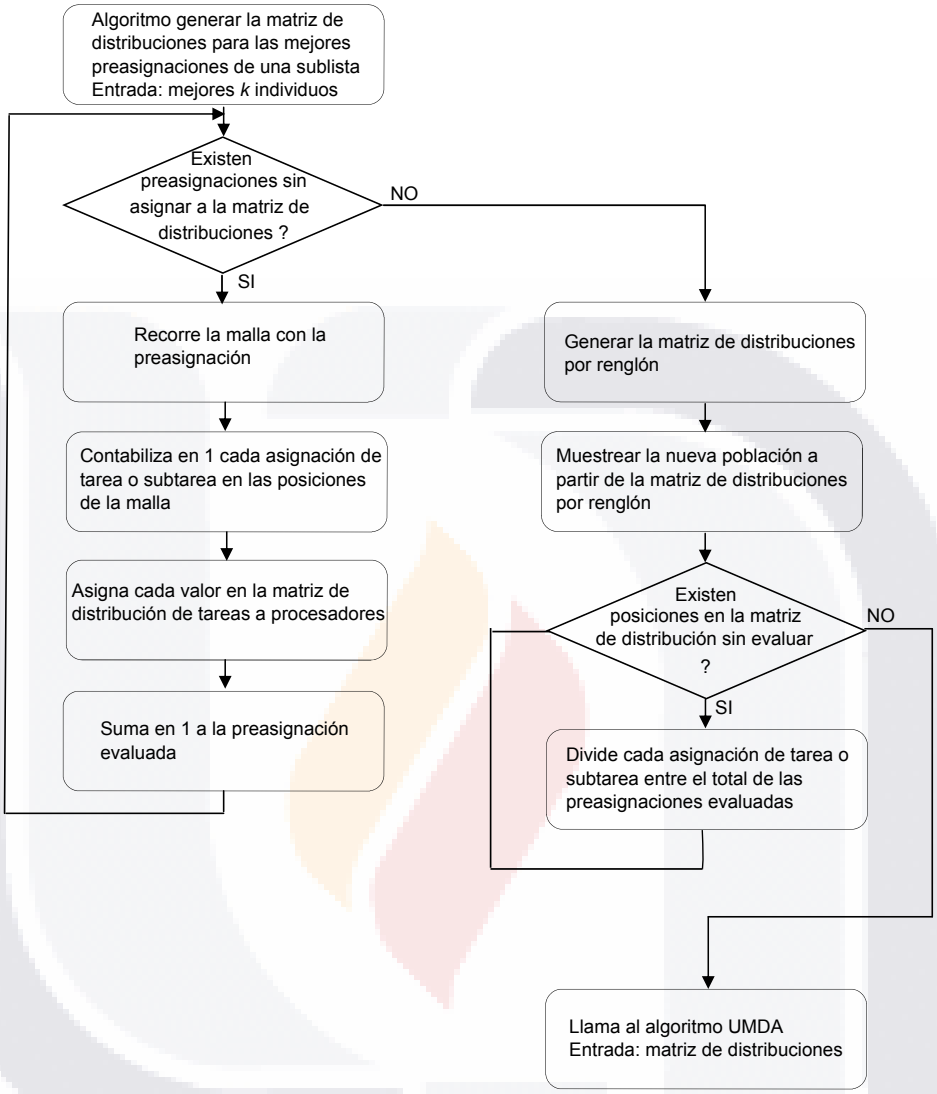

Figura 4.8: Algoritmo para generar la matriz de distribuciones.

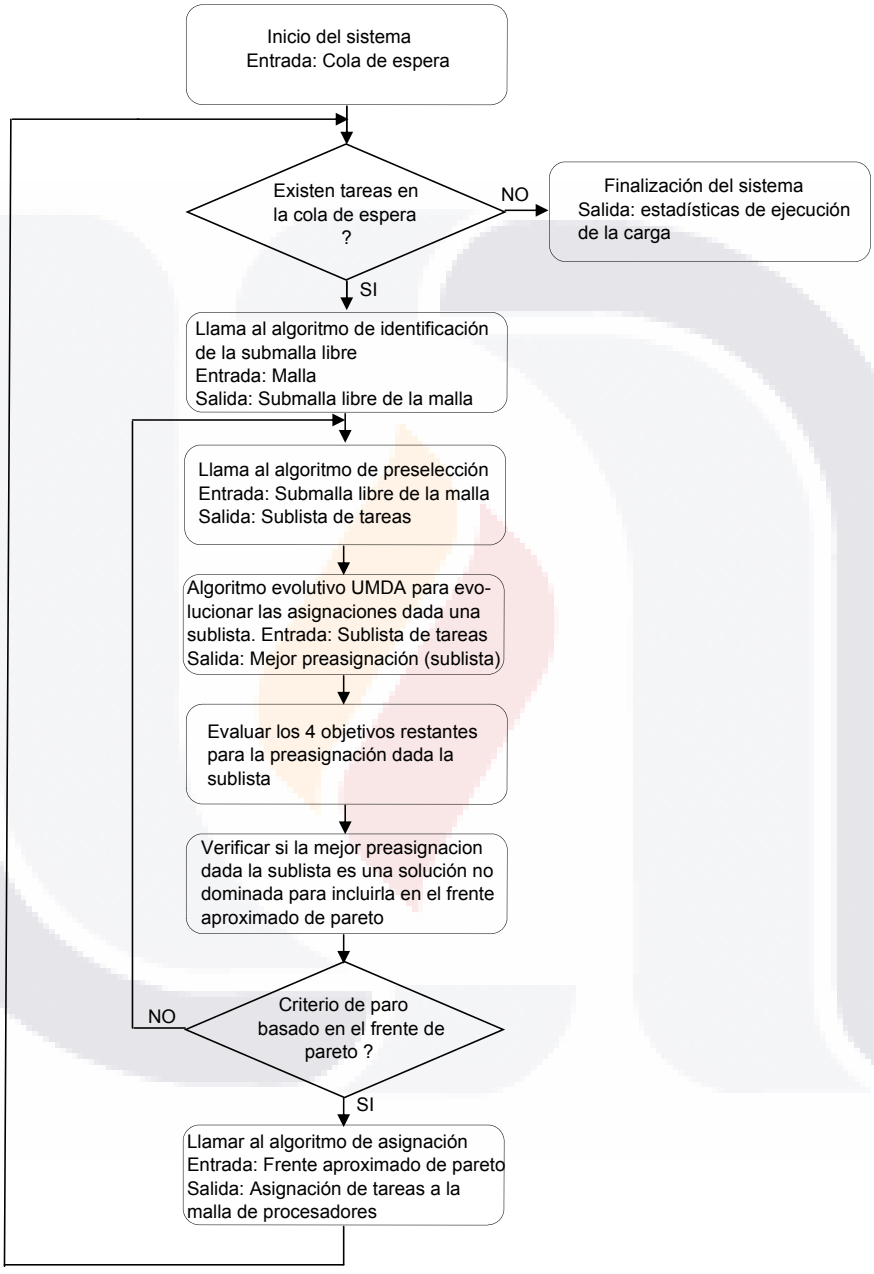

Figura 4.4: Algoritmo de inicio del sistema de planificación y asignación de tareas.

Capítulo 5

DISCUSIÓN DE RESULTADOS

5.1. Introducción

En esta sección se describen los resultados obtenidos a través de los experimentos realizados con los métodos propuestos. Los experimentos están basados sobre una comparación del método propuesto con dos métodos de asignación de tareas: la asignación lineal y el método de las curvas de Hilbert.

El método de asignación lineal es el método más usado en la asignación de procesadores, su facilidad de implementación permite que las tareas sean rápidamente colocadas dentro de las submallas libres, pero presenta dificultades cuando se empiezan a liberar procesadores. La naturaleza del método permite realizar asignaciones de la parte inferior izquierda a la parte superior derecha. El reconocimiento que el algoritmo hace esta basado en una ruta lineal de la malla. El problema principal con este método es que no puede asignar submallas libres en un orden diferente al método lineal establecido, lo que produce una alta segmentación de las tareas e incrementa la transferencia de mensajes en la malla de procesadores.

El método de las curvas de Hilbert. Este método establece un llenado de espacios en forma de curvas, que visita cada punto en una malla cuadrada de tamaño $2 \times 2, 4 \times 4, 8 \times 8, 16 \times 16$, o en cualquier orden de potencia 2. Es un método descrito en 1892 por David Hilbert. Las aplicaciones del método han sido en procesamiento de imágenes, especialmente en la compresión y dilatación de imágenes.

5.2. Parámetros utilizados en los experimentos

Antes de describir cada uno de los experimentos realizados, en esta sección se define cada uno de los parámetros involucrados en los experimentos, así como los posibles valores que pueden tomar. En esta sección las referencias a términos específicos, como por ejemplo, *malla libre* se realizan de manera directa sin explicar su significado, pues se asume que el lector tiene conocimiento de los capítulos anteriores.

1. *Dimensión de la malla.* Atendiendo la definición 2 la cual referencia la dimensión como $M(W, L)$ que consiste de $W \times L$ procesadores, donde W es el ancho de la malla y L es la altura de la malla.

2. *Número de tareas al inicio de la prueba.* Se considera como un conjunto inicial de tareas para ejecutar la primera asignación directa en la malla de procesadores, una vez que las tareas se empiezan a retirar de la malla el método de asignación que utiliza la meta-heurísticas es activado.

3. *Número de subtareas por tarea.* Es un parámetro variable definido por el usuario el cual indica al sistema el número máximo de subtareas que una tarea puede tener y que en términos de un problema computable y paralelizable es el número de sub-problemas en que podrá ser dividido.

4. *Tiempo promedio de espera de las tareas.* Es el cálculo de la sumatoria de los tiempos de espera de las tareas entre el número total de las tareas.

5. *Número total de tareas.* Es el número total de las tareas que el sistema atendió.

6. *Tiempo de verificación de terminación de las tareas.* Es un periodo de tiempo establecido por el sistema para la verificación de los tiempos de terminación de las tareas; cuando el sistema verifica la terminación de una tarea, puede ocurrir:

   a) Que la tarea haya finalizado su tiempo.

   b) Que la tarea aún no finalice su tiempo y únicamente lo disminuya en relación al tiempo transcurrido en el sistema.

7. *Tiempo total de ejecución.* Es el tiempo total que el sistema tarda en ejecutar n tareas.

8. *Número de asignaciones realizadas.* Se establece como el número de asignaciones que el algoritmo realiza para procesar la totalidad de las tareas.

9. *Promedio de subtareas que tienen las tareas.* Se obtiene al sumar la totalidad de subtareas de cada tarea entre el número total de tareas.

5.3. Experimentos realizados usando el método propuesto

Esta sección muestra los experimentos realizados aplicando cada uno de los objetivos que se contraponen; los experimentos de cada objetivo se realizan de manera aislada cada uno del resto, los resultados obtenidos que se muestran en cada una de las gráficas son también relacionadas a cada objetivo.

Los experimentos realizados son:

1. Número de asignaciones vs número de tareas.
2. Tiempo promedio de espera de las tareas (expresado en segundos).
3. Fragmentación externa. Porcentaje de ocupación de procesadores por cada asignación realizada.
4. Porcentaje de inanición de tareas.

Los resultados obtenidos en cada experimentación se muestran en una gráfica comparativa, que muestra el comportamiento de cada uno de los métodos con las diferentes cargas, tamaño de la cola, tiempos de respuesta y los porcentajes de inanición obtenidos.

5.3.1. Propuesta de solución considerando el costo de asignación de un conjunto de tareas en una submalla libre

El primer experimento realizado con el método propuesto, establece usar el costo de asignar un conjunto de tareas con sus respectivas subtareas, a una o varias submallas desocupadas. El procedimiento realizado es el siguiente:

1. El algoritmo de asignación de procesadores informa al planificador de tareas el número y la posición de los n procesadores libres en la malla.
2. El planificador de tareas activa el método para generar la asignación cuadrática con n combinaciones posibles.
3. Por cada combinación generada se realiza el cálculo de los costos de comunicación para cada solución propuesta.
4. Se activa el algoritmo de la burbuja para ordenar las soluciones propuestas, en el orden de mayor a menor.

5. La mejor solución (de menor costo) se considera para ser asignada en la malla de procesadores, a través del algoritmo de asignación de tareas.

Número de asignaciones vs número de tareas.

La asignación de una o varias tarea a un conjunto de procesadores se realiza una vez que el algoritmo de asignación a informado al algoritmo de planificación (en caso de existir) el número de procesadores libres en la malla; con la cantidad de procesadores libres el algoritmo planificador determina cual tarea o tareas se asignarán.

Este experimento compara los dos métodos que no realizan una planificación previa al ingreso de las tareas a la malla, contra el método propuesto que si realiza la planificación. El objetivo es minimizar el tiempo que consume el sistema en asignar las tareas para su procesamiento mediante la minimización del número de asignaciones realizadas.

El número de asignaciones se obtienen al sumar cada asignación realizada durante el procesamiento de las n tareas de cada experimento. Los resultados obtenidos se muestran en la figura 5.1, al usar 360, 819, 988, 1338, 2657, 5120, 5444, 7230 y 10115 tareas en el sistema; en estos resultados hemos encontrado que el método propuesto al realizar la planificación, disminuye el número de asignaciones a la malla en menos del 50 % en relación con los otros dos métodos, los cuales al no realizar planificación el número de asignaciones corresponde con el número de tareas que se procesan.

Lo anterior demuestra que minimizar el número de asignaciones a la malla, mejora significativamente los tiempos de procesamiento del sistema de multicomputadoras.

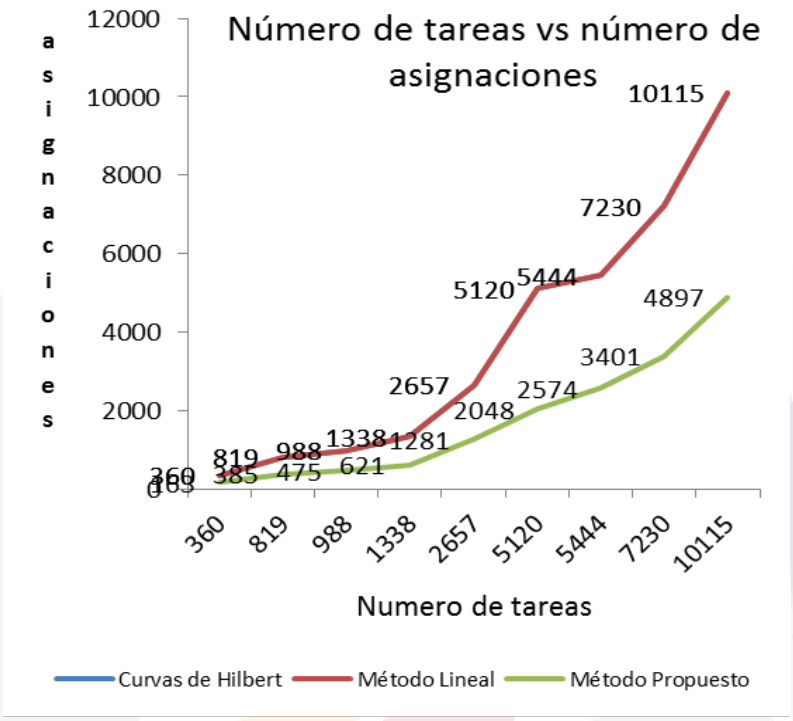

Figura 5.1: Número de tareas contra el número de asignaciones en la malla que los tres métodos realizan durante su ejecución.

5.3.2. Tiempo promedio de espera de las tareas (expresado en segundos)

El tiempo promedio de espera de una tarea, es el tiempo que una tarea espera en la cola para que le sea asignado un conjunto de procesadores libres de la malla. Uno de los objetivos de la computación paralela es minimizar el tiempo que una tarea permanece en la cola, aún si se tratara de una tarea que requiere una gran cantidad de procesadores.

El objetivo de este experimento es determinar el tiempo promedio de espera de las tareas en la cola cuando existe una carga de 360, 819, 988, 1338, 2657, 5120, 5444, 7230 y 10115 tareas en el sistema. La figura 5.2 muestra que el método propuesto produce un tiempo menor de espera, aún con cargas pesadas en el sistema (más de 10000 tareas), en relación al método lineal y el método de curvas de Hilbert. El tiempo promedio de espera de las tareas se expresa en segundos.

Lo anterior demuestra que la planificación de tareas previa a la asignación, minimiza los tiempos de espera de las tareas.

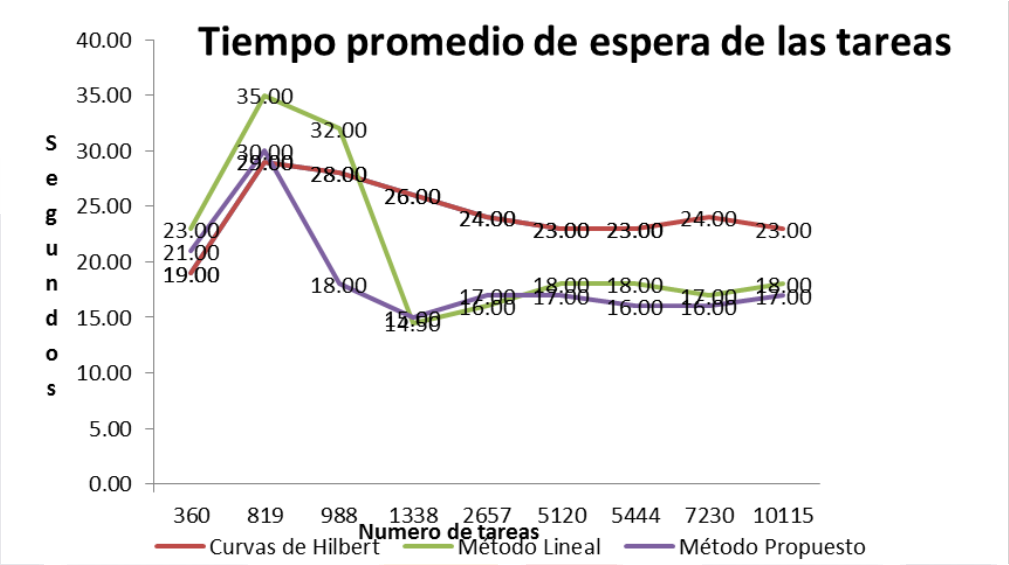

Figura 5.2: Tiempo promedio expresado en segundos que las tareas esperan para ingresar a la malla de procesadores.

5.3.3. Fragmentación externa. Porcentaje de ocupación de procesadores por cada asignación realizada

La fragmentación externa ocurre cuando existen procesadores desocupados dentro de la malla y, no pueden ser asignados a alguna tarea que los requiere, debido a la estrategia de asignación utilizada. Este experimento calcula la fragmentación externa que produce cada método por cada asignación de tareas a la malla. En la figura 5.3 se observa que el método propuesto presenta mejores resultados que los otros dos métodos en este experimento al minimizar el porcentaje de procesadores desocupados en la malla después de cada asignación. Las asignaciones que realiza el método propuesto están basadas en el cálculo de las mejores asignaciones producidas por el algoritmo metaheurístico, calculadas en base a los costos de comunicación y la mejor ubicación de las tareas (calculando las frecuencias de aparición de las tareas en las celdas) en la malla; cabe mencionar que la totalidad de los procesadores no se utiliza en cada asignación, pero el porcentaje de procesadores desocupados durante los experimentos permanece en un promedio de 0.08 % que es menor al compararse con los otros 2 métodos.

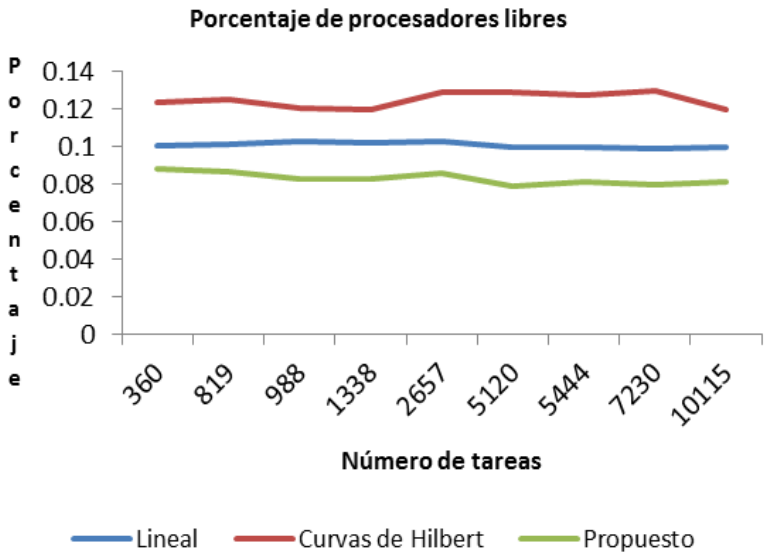

Figura 5.3: Fragmentación externa. Porcentaje de procesadores libres en cada experimento realizado.

5.3.4. Porcentaje de inanición de tareas

En el método propuesto para detectar una tarea en inanición, primero se identifican las tareas con solicitudes grandes de procesadores, y se monitorean durante las fases de asignación; si han transcurrido mas de 5 asignaciones y las tareas grandes no han sido objeto de asignación, entonces se detiene el proceso de aceptación de tareas en la cola de espera, hasta que a la tarea o tareas en inanición le son asignados el número de procesadores que solicitan para su ejecución. En nuestras pruebas, la inanición se presenta cuando el número de tareas en el sistema crece significativamente (más de 2500), como lo muestra la figura 5.4. A diferencia de los métodos con los que hemos realizado la comparación, presentan un porcentaje nulo de inanición, debido a que no realizan una planificación previa en la cola de espera.

Es importante comentar que el sistema propuesto tiene la ventaja de presentar un porcentaje mínimo de inanición, debido a que, si una tarea grande compite con tareas pequeñas el costo de asignación puede ser mínimo, porque los procesadores que se asignarán permanecen contiguos.

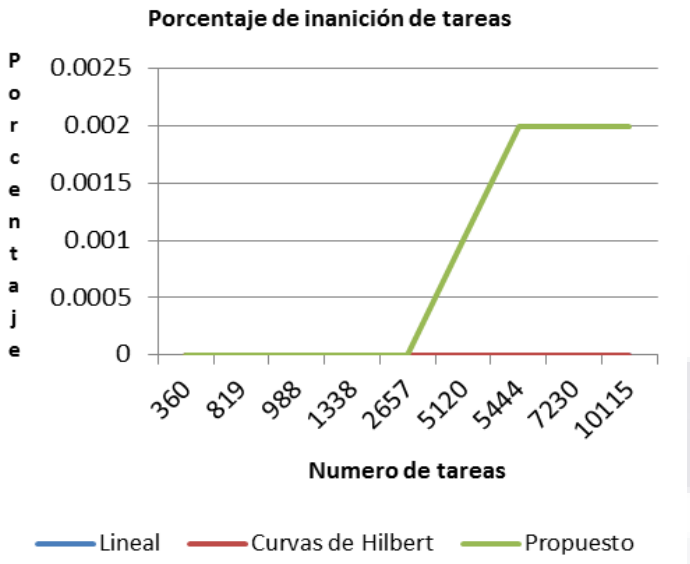

Figura 5.4: Porcentaje de inanición de tareas durante los experimentos realizados.

5.3.5. Conclusiones de la aplicación del método propuesto

Una vez finalizadas las pruebas utilizando el método propuesto, de manera resumida podemos decir que la evaluación de los objetivos de forma separada, permite observar el comportamiento de las funciones objetivo con diferentes cargas del sistema. La evaluación separada de los objetivos:

1. Número de asignaciones.
2. Tiempo promedio de espera de las tareas.
3. Fragmentación Externa.
4. Inanición de tareas.

Permitió visualizar el comportamiento del sistema para cada objetivo y determinar la forma en que se contraponen uno a otro. Al realizar una planificación de las tareas que esperan en la cola, y evitando utilizar una política de asignación FIFO, se logra disminuir el número de veces que el algoritmo debe extraer de la cola, tareas para su asignación, lo que produce una mejora en los tiempos que los trabajos deben esperar por ser atendidos. En entornos paralelos reales, donde la cantidad de tareas, se incrementa

considerablemente, realizar planificaciones previas a la asignación son de gran utilidad para permitir una selección mas dinámica y enviar al algoritmo de asignación un mayor número de trabajos para ser asignados, produciendo que la fragmentación externa de procesadores disminuya considerablemente.

La inanición de tareas, es sin duda alguna un problema que se presenta de forma inherente en la planificación de tareas, porque si el sistema acepta tareas que requieren un gran número de procesadores (mas del 50 % de la cantidad de procesadores disponibles en el sistema paralelo), se generará un cuello de botella al intentar realizar la asignación de este tipo de trabajos, generando esperas con tiempos muy altos; para nuestro caso, una forma de paliar este problema, es logrando que el algoritmo detecte la espera postergada de alguna tarea, y evitar que nuevas tareas se sigan generando, hasta que se atienda el trabajo que este en inanición. Para nuestro caso, aunque se ha detectado tareas con este problema, el algoritmo ha funcionado correctamente logrando que las cifras reportadas sean mínimas.

Esta primera fase de experimentación nos permitió plantear una segunda fase de experimentación, la cual se explica en la siguiente sección, la cual consistió en la observación de los resultados hasta aquí obtenidos y plantear la contraposición de cada uno de los objetivos y definir la incidencia que tiene cada uno de ellos, tanto en la fase de planificación como en la fase de asignación.

5.4. Planteamiento de la extension al método propuesto

Una vez realizados los experimentos anteriores, se propuso una extension al método propuesto para determinar el frente de Pareto considerando los cinco objetivos propuestos, y la forma en que los mismos se contraponen. El esquema de trabajo para la realización para esta experimentación se detalla a continuación.

El trabajo realizado consistió en la determinación de los valores de las funciones objetivo para cada uno de los objetivos evaluados con el método propuesto, en cada una de las experimentaciones realizadas.

De esta forma, los objetivos a evaluar se identifican de la siguiente forma:

1. la función 1, minimizar el número de trabajos en la cola de espera, a través de este objetivo es posible que el sistema pueda realizar la asignación de mas de una

tarea a la vez, para permitir disminuir el número de trabajos en la cola de espera e ingresar un número mas grande de tareas a la cola; el valor de esta función representa el porcentaje en relación con el total de tareas que se ejecutaron,

2. la función 2, maximizar el número de tareas ejecutándose en paralelo dentro de la malla, lo que permite disminuir el tiempo promedio de espera de las tareas,

3. la función 3, minimizar la fragmentación externa, el porcentaje de procesadores libres una vez que se realiza la mejor asignación a la malla de procesadores.

4. la función 4, corresponde a minimizar la inanición de tareas, el porcentaje de tareas grandes que no son asignadas a la malla de procesadores, es decir, permanecen en la cola de espera, una vez que se realiza la mejor asignación de las tareas a la malla de procesadores,

5. la función 5, minimizar los costos de comunicación entre la tarea y sus subtareas al ser asignadas a la malla de procesadores, es decir asignar los procesadores de la manera mas cercana posible entre ellos.

A modo de colofón y con el objetivo de resaltar la parte central de esta segunda experimentación, se quiere comentar que muchos de los problemas de optimización reales son dinámicos, dado que se producen cambios en las condiciones de las que dependen las funciones objetivo, en las restricciones que deben satisfacer las soluciones, etcétera. Por ejemplo en un problema de planificación pueden variar las características de los recursos y el volumen de tareas a asignar [14].

En los problemas de optimización cuyas soluciones deben optimizar a la vez varios objetivos, a menudo contrapuestos la noción de optimo toma otro significado, dado que en lugar de proporcionar una única solución, los procedimientos que se aplican a estos problemas de optimización multiobjetivo, proporcionan un conjunto de soluciones de compromiso, generalmente conocidas como soluciones de Pareto óptimas [18]. En los problemas de optimización multiobjetivo dinámicos, las funciones objetivo y el conjunto de variables que definen el espacio de soluciones pueden cambiar en el tiempo, y la rapidez de reacción ante los cambios es un aspecto importante en el contexto de la optimización dinámica, y por tanto, el uso de plataformas de altas prestaciones puede resultar muy adecuado a este tipo de problemas.

Los objetivos están normalmente en conflicto entre sí, de forma que optimizar uno de ellos se debe hacer a costa de empeorar los valores de otro, debiendo por tanto establecerse un compromiso aplicando el concepto de Pareto optimalidad, produciendo

vectores que son óptimos en el sentido de Pareto, y a cuyas soluciones se les llama no dominadas. Al conjunto de todas las soluciones no dominadas, cuando se obtiene el espacio de decisión, determinan el frente de Pareto en el espacio de objetivos.

5.4.1. Contraposición entre objetivos

En esta sección se muestra una secuencia de gráficas que permiten observar el comportamiento del algoritmo de evaluación de las funciones objetivo. En ellas podemos observar como se crea el espacio de soluciones para diferentes cargas de trabajo del sistema.

La primer gráfica que aparece en la figura 5.5 muestra las cargas de trabajo y los resultados obtenidos durante las fases de evaluación. La búsqueda de las mejores soluciones se dan conforme se crea el frente Pareto generado, y que se muestra en la parte central de la gráfica. Para aquellos objetivos de minimización las tendencias de los valores son hacia la parte interna de la gráfica y para los valores de maximización de las funciones los valores son los que se obtienen en la parte alta de la gráfica.

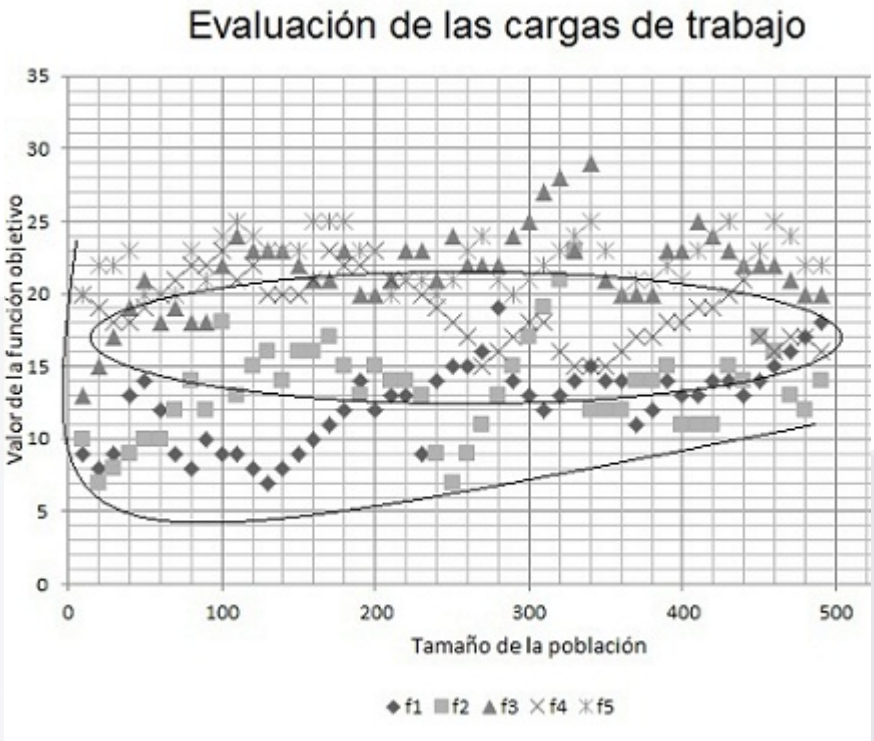

Figura 5.5: Evaluación de las diferentes cargas de trabajo para obtener el frente de Pareto aproximado.

Conforme la carga de trabajo en el sistema se hace mas denso, la búsqueda de las mejores soluciones se realiza en una área mas densa, que aglutina los resultados obtenidos.

Las siguientes gráficas, que aparecen en la figura 5.6 y en la la figura 5.7, muestra una carga mas pesada en el sistema, los resultados obtenidos para este tipo de cargas permite al algoritmo obtener una muestra mas amplia de soluciones para que en el ciclo de búsqueda la población sea mas extensa. El tiempo de ejecución para este carga, se estableció en cuatro horas y 3 minutos. La gráfica muestra a los individuos que obtienen los valores mas significativos para las soluciones.

La tendencia de búsqueda de las mejores soluciones, se basa en obtener las mejores soluciones utilizando como pivote la mejor solución, para el objetivo 5. A partir de los valores de esta función objetivo, determinamos los siguientes valores para las funciones, un segundo valor a considerar es el de la función 3, después el de la función 4, la

función 2 y como ultimo valor el de la función 1. El escalamiento de estos valores, esta determinado para la selección de la mejor asignación en la malla de procesadores.

Las tendencias de las cargas en el sistema se determinan, con el número de tareas que seran procesadas, asi como tambien el largo de la cola para cada ejecución. El desempeño de los algoritmos propuestos, tiende a ser exponencial, esto es, según la carga del sistema es el tiempo que se tarda en procesar todas las tareas y subtareas. Las siguientes gráficas muestran los valores obtenidos en cuanto al valor de las funciones objetivo y el tamaño de la población.

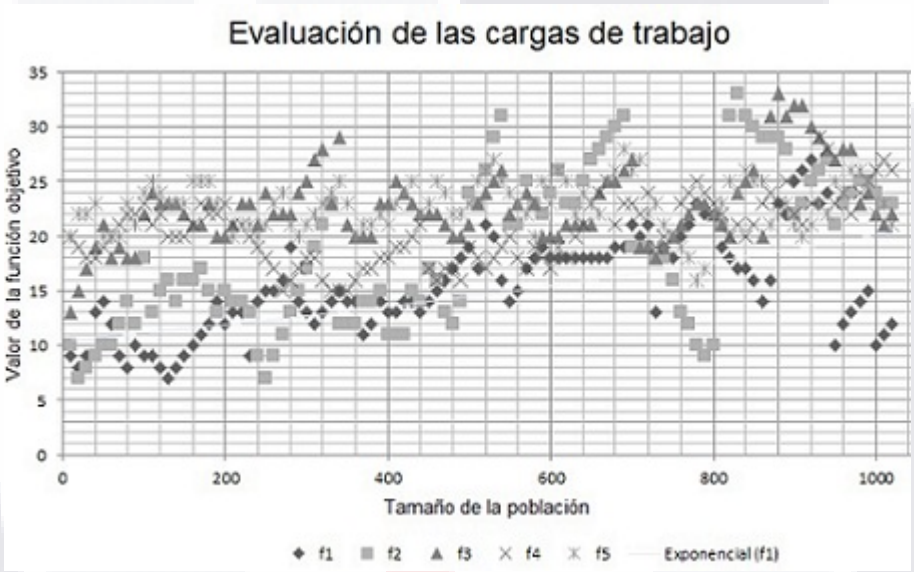

Figura 5.6: Evaluación de las diferentes cargas de trabajo para obtener el frente de Pareto aproximado con una carga mas densa en el sistema.

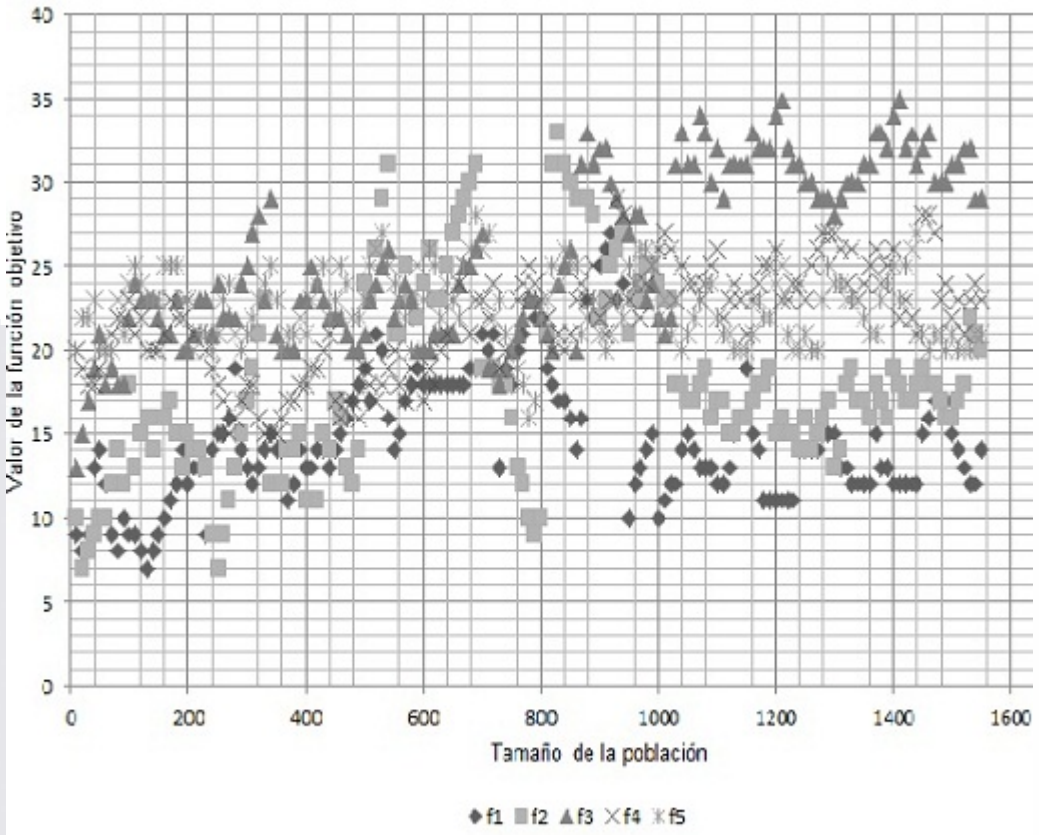

Figura 5.7: Segunda evaluación de las diferentes cargas de trabajo para obtener el frente de Pareto aproximado con una carga mas densa en el sistema.

5.4.2. Conclusiones de la aplicación de la extension al método propuesto

La creación de una estructuración algorítmica, para la evaluación de la extension al método propuesto nos permitió evaluar de manera mas exacta cada una de las cargas propuestas inicialmente en las primeras experimentaciones. El evaluar de manera aislada cada una de los objetivos, crea un ordenamiento de los mismos por un orden de importancia en el sistema paralelo. De esta manera y considerando los resultados de las experimentaciones exahustivas, se aplicó la secuencia de algoritmos en el orden establecido, para la generación de las poblaciones utilizando la matriz de incidencias y para lograr obtener poblaciones mas competitivas en sus resultados. Cabe destacar que la naturaleza del problema es dinámica, lo que provoca que se obtengan resultados

diferentes en cada evaluación.

Los frentes aproximados de Pareto, se obtienen con ciclos de evaluaciones repetitivas de las poblaciones. Las tendencias de los valores obtenidos, son aproximaciones a los valores objetivo, pero la mezcla de diferentes situaciones que se presentan durante la evaluación, provoca que el algoritmo UMDA decida sobre los valores mas cercanos a los óptimos para poder realizar la asignación de las tareas a la malla de procesadores.

CONCLUSIONES

La computación paralela es una opción viable para la ejecución de tareas que requieren poder computacional, que sobrepasa los límites de los sistemas de un solo procesador, pero establece condiciones que deben cumplirse durante las fases de planificación y asignación de las tareas a la malla de procesadores, para mejorar el desempeño y maximizar el uso de los recursos. En este trabajo hemos presentado la detección de los objetivos que se contraponen en la planificación y asignación de tareas en una malla de procesadores, y un método que realiza una planificación temprana de tareas que permanecen en la cola de espera, la cual busca minimizar el número de asignaciones a la malla, disminuir el tiempo de espera de las tareas, maximizar el número de procesadores ocupados en la malla, disminuir la inanición de tareas grandes y maximizar la contigüidad entre procesadores asignados a una tarea apoyándose en un mecanismo para evitar la inanición de tareas grandes en la cola de espera.

Dentro del estado del arte, encontramos diferentes formas para realizar la asignación de tareas en una malla de procesadores, métodos que utilizan formas geométricas que se mueven a través de la malla para detectar las submallas de procesadores libres y que pueden ser rotadas de diferentes formas para poder hacer caber la tarea, dividir el requerimiento de diferentes tamaños de submallas, buscar puntos de cruce en los procesadores libres para iniciar una búsqueda alrededor de dichos puntos, entre otros. La mayoría de los trabajos propuestos se enfocan en realizar una detección de procesadores libres, pero pasan por alto la planificación de las tareas y utilizan la política del primer elemento en llegar a la cola es el primer elemento en ser asignado, por considerarlo libre de inanición y ser una política justa de planificación.

Los sistemas de multicomputadoras comerciales que contienen miles de procesadores, prefieren ejecutar los requerimientos de manera lo más aislada posible para evitar que las aplicaciones puedan interferir entre ellas. Pero cuando el número de recursos es limitado para correr aplicaciones que requieren ejecutarse en paralelo, se busca optimizar de la mejor manera los recursos disponibles; una manera de lograrlo es provocar que las aplicaciones que se ejecutan dentro de una malla de procesadores lo hagan con la mayor

contigüidad posible para evitar que un gran número de procesadores se encuentren ociosos.

Pero no solo la proximidad y el uso adecuado de los recursos es indispensable en la asignación y la planificación de los recursos computacionales, sino también un análisis de los objetivos que se persiguen al planificar, o al asignar dichos recursos. En nuestro trabajo hemos considerado los primeros 5 objetivos como los más importantes y que más influencia tienen en el uso de los recursos computacionales en un sistema de multicomputadoras.

Este trabajo presenta un método novedoso en el que se considera en primer lugar, la planificación de las tareas que permanecen en la cola de espera; dicha planificación, basada en un método probabilístico, permite evitar que tareas grandes sean discriminadas permitiendo únicamente a tareas pequeñas ser programadas para su ingreso a la malla de procesadores; la búsqueda de tareas en la cola de espera permite que diferentes subconjuntos de tareas sean seleccionados y probados en las submallas libres usando una asignación cuadrática dinámica, lo que permite obtener diferentes costos de asignación y seleccionar entre éstos el mejor. En segundo lugar, y considerando que se realizó una planificación, se procede a activar el algoritmo asignador, para colocar en forma definitiva las tareas en la malla, en donde permanecerán hasta su terminación. Finalmente, cuando una o más tareas son retiradas de la malla, se activa un verificador de procesadores en la totalidad de la malla para detectar las posiciones libres. Los tres algoritmos trabajan en forma conjunta para poder obtener los mejores tiempos de ejecución de las tareas.

Los resultados obtenidos en los experimentos con el método propuesto basado en una planificación demuestran que disminuir el número de asignaciones en la malla permite minimizar los tiempos de procesamiento de las tareas, disminuir el tiempo que las tareas permanecen en la cola y disminuir la fragmentación interna; lo anterior permite también disminuir los riesgos de saturar la red de comunicación, apoyándose en el algoritmo de asignación que realiza un reconocimiento de la malla y mantiene la información de las submallas libres existentes mediante una estructura de datos dinámica. La inanición de tareas aún está presente en nuestro sistema, pero los casos que aparecen son mínimos; además, el algoritmo mantiene un control estricto sobre las tareas que empiezan a caer en inanición.

Los resultados obtenidos se encuentran por encima de los métodos utilizados tradicionalmente, aunque cabe señalar que es necesario un mejoramiento en el algoritmo para evitar la inanición de tareas en la malla, aunque un sistema libre de inanición de

tareas es aquel que utiliza una política que no realiza una planificación previa a la asignación de los trabajos en la malla; es decir, que si nuestro algoritmo propuesto planifica los trabajos de la cola de espera, siempre existirán tareas que esperarán determinados ciclos de asignación, pero el objetivo fundamental es que no esperen por siempre en la cola de espera.

Al conjuntar los resultados obtenidos del método propuesto y la extensión al método propuesto, ha sido posible llevar a cabo la evaluación de un problema multiobjetivo, y mostrar a través de los resultados que como tal, asignar tareas con un número dinámico de subtareas a una malla de procesadores, y tratar de obtener las mejores asignaciones, debe ser evaluado con diferentes cargas para determinar cuál objetivo es más incidente en la solución. Al tener diferentes escenarios, cada vez que se pretende realizar la evaluación de la población de tareas de la cola de espera, el algoritmo evolutivo decide la mejor asignación, considerando como entrada los valores obtenidos de cada función objetivo; lo anterior no se considera una tarea trivial, dado que decidir por una solución es motivo de empeorar otras soluciones obtenidas, al realizar esto de manera automatizada el algoritmo decidirá siempre por el empeoramiento de otros objetivos. Lo anterior conlleva a un mejoramiento paulatino del algoritmo que decide, probablemente mediante la utilización de una base de conocimiento o un algoritmo de decisión superior, que permita reevaluar la decisión tomada inicialmente por el primer algoritmo.

Finalmente podemos concluir esta tesis mencionando que desarrollar sistemas paralelos de cualquier tipo de arquitectura implica un esfuerzo mayor que desarrollar sistemas del tipo monousuario, dada la cantidad de recursos que deben ser controlados para resolver un problema inherentemente paralelo. Pero en contraposición, si queremos contar con un sistema de este tipo, nuestra segunda opción es adquirirlo, pero el costo es demasiado alto dado que se cotizan en el mercado mundial en el orden los cientos de miles de dólares, una cantidad que en la mayoría de las veces las Instituciones Educativas no tienen para invertir.

Finalizar una tesis que utiliza diferentes áreas del conocimiento para su desarrollo, conlleva a plantear diferentes líneas de investigación. En esta sección se describen los trabajos que se han planteado desarrollar posteriormente, después de la finalización de este trabajo de tesis. En algunos de ellos ya se están desarrollando propuestas de investigación y en otros únicamente existe la propuesta.

El desarrollo de un algoritmo para la verificación temprana de tareas que serán retiradas de la malla de procesadores. Mediante este algoritmo se pretende detectar de manera temprana la terminación de una lista de tareas, que serán puestas en el estado

de terminado. Esta detección permitirá al sistema realizar una planificación previa a la ejecución del algoritmo de verificacion de tiempos de terminación; considerando que para tal detección temprana, será necesario realizar un promedio de los tiempos de terminación de un conjunto de tareas debido a que los tiempos de las mismas no son iguales.

El desarrollo de una base de conocimiento que permita almacenar de manera periódica las soluciones obtenidas por el sistema, para posteriormente consultarla, y obtener estadísticas de las decisiones realizadas, con los valores de las funciones objetivo.

Se propone también la realización de pruebas de los algoritmos desarrollados en esta investigación, en un sistema de multicomputadoras con el sistema operativo Linux, que permita visualizar de manera real los resultados obtenidos en las experimentaciones realizadas para ambos métodos.

BIBLIOGRAFIA

Bibliografía

[1] Ahmad S. B. Processor Allocation with Reduced Internal and External Fragmentation in 2D Mesh-based Multicomputers. Journal of Applied Science 11 (6) 943-952, 2011 ISSN 1812-5654 2011 Asian Network for Scientific Information.

[2] Alba E., Tomassini M. Parallelism and Genetic Algorithms, IEEE Transactions on Evolutionary Computation 6, 5, pp. 443-462. Año 2002.

[3] Alessandro Amoroso, Keith Marzullo. Multiple Job Scheduling in a Connection-Limited Data Parallel System. IEEE Transactions on Parallel and Distributed Systems, Vol. 17, No. 2, February 2006.

[4] Andrews G. R. Foundations of Multithreaded, Parallel, and Distributed Programming Addison-Wesley. ISBN 0-201-35752-6. 2000.

[5] Araujo L. y Cervigón C. Algoritmos Evolutivos. Un enfoque práctico. Editorial: Alfaomega Ra-Ma. Año 2009.

[6] Aridor Y., Domany T., Goldshmidt O., Kliteynik Y., Moreira J., and Shmueli E., Open Job Management Architecture for the Blue Gene/L Supercomputer, Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'05), June 19, Cambridge, pp. 91-107, 2005.

[7] Bani A. S., Submesh Allocation in 2DMesh multicomputers: Partitioning at the Longest Dimension of Request. The International Arab Journal of Information Technology, Vol. 10, No.3, May 2013. Pp. 245252.

[8] S. Bani-Mohammad, M. Ould-Khaoua, and I. Ababneh, A New Processor Allocation Strategy with a High Degree of Contiguity in Mesh-Connected Multicomputers, Journal of Simulation Modelling, Practice & Theory, vol. 15, no. 4, pp. 465-480, 2007.

[9] S. BaniMohammad, M. OuldKhaoua, I. Ababneh, and L. Machenzie, Non-contiguous Processor Allocation Strategy for 2D Mesh Connected Multicomputers

Based on Sub-meshes Available for Allocation, Proc. of the 12th Int. Conference on Parallel and Distributed Systems (ICPADS006), Minneapolis, Minnesota, USA, IEEE Computer Society Press, vol. 2 , pp. 41-48, 2006.

[10] BaniMohammad S. Efficient Processor Allocation Strategies for Mesh-Connected Multicomputers. Tesis Doctoral. Faculty of Information and Mathematical Sciences University of Glasgow U. K. 2008.

[11] Bani-Mohammad S., Ould-Khaoua M., Ababneh I., and Machenzie L., A Fast and Efficient Processor Allocation Strategy which Combines a Contiguous and Noncontiguous Processor Allocation Algorithms, Technical Report; TR-2007-229, DCS Technical Report Series, Department of Computing Science, University of Glasgow, January 2007.

[12] Bani-Mohammad S., Ould-Khaoua M., Ababneh I., and Machenzie L., Non-contiguous Processor Allocation Strategy for 2D Mesh Connected Multicomputers Based on Sub-meshes Available for Allocation, Proceedings of the 12^{th} International Conference on Parallel and Distributed Systems (ICPADS′06), Minneapolis, Minnesota, USA, IEEE Computer Society Press, vol. 2 , pp. 41-48, 2006.

[13] Blue Gene Project, http://www.research.ibm.com/bluegene/index.html, 2007.

[14] Branke J., Mattfel D., Anticipation and Flexibility in Dynamic Scheduling., International Journal of Production Research., Vol. 43, Num. 15, Pp. 31033129, Agost 2005.

[15] Blum C. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison Universit'e Libre de Bruxelles AND ANDREA ROLI Universit'a degli Studi di Bologna

[16] Cantú Paz E. Efficient and Accurate Parallel Genetic Algorithms. Editorial: Kluwer Academic Publisher. Año 2001.

[17] Coello C., Van Veldhuizen D., Lamont G. Evolutionary Algorithms for Solving Multi-Objective Problems. Editorial: Kluwer Academic Publisher. Año 2002.

[18] Coello C., An Update Survey of GA-Based Multiobjective Optimization Techniques. Technical Report LANIA-RD-98-08 Laboratorio Nacional de Informática Avanzada (LANIA), México, 1998.

[19] Chen J., Taylor V. E. Mesh Partitioning for Efficient Use of Distributed Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 1, January 2002.

[20] C.Y. Chang and P. Mohapatra, Performance improvement of allocation schemes for mesh-connected computers, Journal of Parallel and Distributed Computing, vol. 52, no. 1, pp. 40-68, 1998.

[21] Cray, Cray XT3 Datasheet, 2005.

[22] Das D. S. and Dhiraj K. Pradhan, Fellow, IEEE. Job Scheduling in Mesh Multicomputers. IEEE Transactions on Parallel and Distributed Systems, Vol. 9, No. 1, January 1998.

[23] Davis L. Handbook of Genetic Algorithms. Editorial: Van Nostrand Reinhold, New York. Año 1991.

[24] Deb K., Agrawal S., Pratab A., Meyarivan T. (2000). A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II, Proceedings of the Parallel Problem Solving from Nature VI Conference, pages 849-858. Springer, 2000

[25] De Rose C. A. F., Heiss H. U., and Linnert B., Distributed Dynamic processor Allocation for Multicomputers, Parallel Computing, vol. 33, no. 3, pp. 145-158, 2007.

[26] Deb K. Multi-Objective Optimization using Evolutionary Algorithms. Editorial: John Wiley & Sons, LTD. New York U. S. A. 2001. ISBN 0-471-87339-X.

[27] Dutot P. F., Tchimou N.Takpe, Frederic Suter. Scheduling Parallel Task Graphs on (Almost) Homogeneous Multicluster Platforms. IEEE Transactions on Parallel and Distributed Systems, Vol. 20, No. 7, July 2009.

[28] Farina M., Deb K., Amato P. Dynamic Multiobjective Optimization Problem, Test Cases, Approximations and Applications. IEEE Trans. on Evolutionary Computation, Vol.8 No. 5., Pp. 342425. October 2004.

[29] A. Ferreira, A. G. vel Lejbman, and S. W. Song, Bus-based parallel computers: a viable way for massive parallelism, Proceedings of Parallel Architectures Languages Europe (PARLE '94), Lecture Notes in Computer Science 817, pp. 553-564, Springer-Verlag, 1994.

[30] Foster I. Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison Wesley 1995.

[31] Fujii H., Yasuda Y., Akashi H., Inagami Y., Koga M., Ishihara O., Kashiyama M., Wada H., and Sumimoto T., Architecture and performance of the Hitachi

SR2201 massively parallel processor system, Proceedings of the 11th International Parallel Processing Symposium (IPPS'97), pp. 233-241, IEEE Computer Society Press, 1997.

[32] Goldberg D. Genetic Algorithms in Search, Optimization and Machine Learning. Editorial: Addison-Wesley. Año 1989.

[33] Heiss H. U. Dynamic Partitioning of Large Multicomputer Systems. Proc. Int. Conf. on Massively Parallel Computing Systems (IEEE MPCS94), Ischia, May 2-6., 1994

[34] Intel Corporation, A Touchstone DELTA system description, 1991.

[35] Intel Corp., Paragon XP/S product overview, Supercomputer Systems Division, Beaverton, Oregon, 1991.

[36] Kruskal C. P. and Snir M., The performance of multistage interconnection networks for multiprocessors, IEEE Trans. Computers, vol. 32, no. 12, pp. 1091-1098, 1983.

[37] Larrañaga P., Lozano J. A. and Mühlenbein H. Estimation of Distribution Algorithms Applied To Combinatorial Optimization Problems. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, 2003.

[38] Levine M., CRAY XT3 at the Pittsburgh Supercomputing Centre, DEISA Symposium, Bologna, 4-5 May 2006.

[39] Lo V., Windisch K., Liu W., and Nitzberg B., Non-contiguous processor allocation algorithms for mesh-connected multicomputers, IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 7, pp. 712-726, 1997.

[40] Liu P., Hsu Ch. and Wu J. IO Processor Allocation for Mesh Cluster Computers, IEEE Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS 05), 2005.

[41] Mache J., Lo V., and Windisch K. Minimizing Message-Passing Contention in Fragmentation-Free Processor Allocation, Proceedings of the 10th International Conference on Parallel and Distributed Computing Systems, pp. 120-124, 1997.

[42] Mohapatra P., Wormhole routing techniques in multicomputer systems, ACM Computing Surveys, vol. 30, no. 3, pp. 375-411, 1998.

[43] Noakes M., Wallach D. A., and Dally W. J., The J-machine multicomputer: an architecture evaluation, Proceedings of the 20th International Symposium Computer Architecture, pp. 224-235, 1993.

[44] Quinn M. Parallel Computing: Theory and Practice. McGraw-Hill, New York U. S. A., 1994.

[45] Pérez M. J., Sancho F. C. Colección de Divulgacion Cientifica. Universidad de Sevilla. Secretariado de Publicaciones 2003.

[46] Russell S. Norving P. Inteligencia Artificial: Un Enfoque Moderno, segunda edición, Simón & Schuster Company Englewood Cliffs, New Jersey, Prentice Hall. 2008.

[47] Schaffer D. (1984). Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. PhD thesis, Vanderbilt University.

[48] Sinnen O. Task Scheduling for Parallel Systems. Editorial: Wiley Interscience a John Wiley & Sons Inc., Publication. 2007.

[49] Edi Shmueli E., G. Dror and Feitelson D. On Simulation of Parallel-Systems Schedulers: Are We Doing the Right Thing? IEEE Transactions on Parallel and Distributed Systems, Vol. 20, No. 7, July 2009.

[50] Srinivas N., Deb K. (1994), Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. Evolutionary Computation 2(3): 221-248, 1994.

[51] Srinivasan T., Seshadri J., Chandrasekhat A. and Siddharth Jonathana J. B. Minimal Fragmentation Algorithm for Task Allocation in Mesh-Connected Multicomputers Proceedings. IEEE International Conference on Advances in Intelligent Systems Theory and Applications AISTA 2004 in conjunction with IEEE Computer Society, ISBN 2-9599-7768-8, IEEE Press.

[52] Surana S. Quadratic Assignment Problem. En http://www.cs.berkeley.edu/ ejr/GSI/cs267-s04/homework-0/results/.

[53] Suzaki K., Tanuma H., Hirano S., Ichisugi Y., Connelly C., and Tsukamoto M. Multi-tasking Method on Parallel Computers which Combines a Contiguous and Non-contiguous Processor Partitioning Algorithm. Lecture Notes in Computer Science, Springer, London, pp. 641-650, 1996.

[54] Torres J., Rodriguez E. Conceptos de Cómputo Paralelo. Editorial: Trillas. Mayo 2000. ISBN: 968-24-6223-3.

[55] Tannenbaum A. S. Sistemas Operativos Modernos. Prentice Hall. 2009. ISBN 970-26-0315-3

[56] Xavier C., Iyengar S. Introduction to Parallel Algorithms. Editorial: Wiley-Interscience. New York U. S. A. 1998. ISBN 0-471-25182-8.

[57] Veldhuizen V., Zydallis J., Lamont G., Considerations in engineering parallel multiobjective evolutionary algorithms. IEEE Trans. Evolutionary Computation 7(2): 144-173, Año 2003.

[58] Velarde A., Ponce de Leon E. , Diaz E., Padilla A. Planning and Allocation of processors in 2D meshes. Doctoral Consortium. Mexican Internacional Conference on Artificial Intelligence MICAI 2010 Pachuca Hidalgo, México.

[9] Velarde A., Ponce de Leon E., Diaz E., Padilla A. Dynamic quadratic Assignment to Model Task Assignment Problem to Processors in a 2D Mesh. Advances in Soft Computing Algorithms, Editors: I. Batyrshin, G. Sidorov. Research in Computing Science Vol. 54, pp. 199-218, 2011. Puebla, México. RCS.

[10] Velarde A., Ponce de Leon E., Diaz E. Planning and Allocation Tasks in a Multicomputer System as a Multi-objective Problem. Advances in Intelligent Systems and Computing 227. EVOLVE 2013. International Conference held at Leiden University, July 10-13, 2013. Leiden, The Netherlands. Springer.

[59] Wu F., Hsu C.C. and Chou L.P., Processor Allocation in the Mesh Multiprocessors Using the Leapfrog Method, IEEE Transactions on Parallel and Distributed Systems, vol. 14, no. 3, pp. 276-289, 2003.

[60] Yassin A., Al-Dubai, Ould-Khaoua M., Mackenzie L. M. , An Efficient Path-Based Multicast Algorithm for Mesh Networks, ipdps, pp.283, International Parallel and Distributed Processing Symposium (IPDPS'03), 2003.

[61] YOO S.M. , Yong H. and CHOO H. Dynamic Scheduling and Allocation in Two-Dimensional Mesh-Connected Multicomputers for Real-Time Tasks. IEICE TRANS. INF. & SYST., VOL.E84-D, No. 5 MAY 2001 pp 613-621.

[62] Zolfaaghari R. Efficient Algorithm for Processor Allocation in Mesh Multicomputers Network with Limitations and Assumptions. IJCEM International Journal Of Computational Engineering & Management, Vol. 16 Issue 4, July 2013. ISSN (Online) 2230-7893. Pp. 513.

ANEXOS

Anexo A. Aceptación del articulo.

EVOLVE 2013 Apr 15, 2013

To Apolinar Velarde Martinez,

Dear Apolinar,

I am glad to inform you that your paper submitted to the EVOLVE 2013 entitled

Planning and Allocation Tasks in a Multicomputer System as a Multi-objective Problem

has been accepted for publication in the proceedings of the conference. Below you will find the reviewers' comments. Please follow the instructions of the reviewers and send your final camera ready manuscript together with the Latex sources to

Michael Emmerich emmerix@gmail.com

or Oliver Schuetze schuetze@cs.cinvestav.mx

not later than

Friday, May 03, 2013

It is mandatory that you use the Latex style required by the Springer book series, see

http : *//www.springer.com/series/*4240

for details. If you have any doubts to not hesitate to contact Michael Emmerich or me.

With kind regards, Oliver

Anexo B. Disco complemento.

Para la comprensión de este documento se sugiere consultar disco complemento.