Mapping sobre arquitecturas heterogéneas

Laura De Giusti

UNIVERSIDAD NACIONAL DE LA PLATA
FACULTAD DE INFORMÁTICA

TESIS DOCTORAL EN CIENCIAS INFORMÁTICAS

Mapping sobre arquitecturas heterogéneas

Laura De Giusti

Director: Dr. Guillermo Simari
Codirectores: Dr. Emilio Luque - Dr. Marcelo Naiouf

La Plata, Argentina
Septiembre de 2008

Edulp
Editorial de la Universidad de La Plata

De Giusti, Laura
Mapping sobre arquitecturas heterogéneas. - 1a ed. -
La Plata: Universidad Nacional de La Plata, 2011.
164 p.; 24x16 cm.

ISBN 978-950-34-0722-6

1. Arquitectura . 2. Modelos de Programación. I. Título
CDD 720

Mapping sobre arquitecturas heterogéneas

Laura De Giusti

Coordinación Editorial: Anabel Manasanch
Corrección: María Eugenia López, María Virginia Fuente, Magdalena Sanguinetti y Marisa Schieda.
Diseño y diagramación: Andrea López Osornio

Editorial de la Universidad Nacional de La Plata (Edulp)
47 N° 380 / La Plata B1900AJP / Buenos Aires, Argentina
+54 221 427 3992 / 427 4898
editorial@editorial.unlp.edu.ar
www.editorial.unlp.edu.ar

Edulp integra la Red de Editoriales Universitarias (REUN)

Primera edición, 2011
ISBN Nº 978-950-34-0722-6

Impreso en Argentina

A Luis que me ayudó a rehacer mi vida y poder volver a ser feliz.
A mi familia, en especial a papá y mamá.
A mi ahijado.

Agradecimientos

En primer lugar, quiero agradecer a Marcelo y a Franco por haberme ayudado en un montón de momentos inciertos de esta tesis, como así también en la redacción, y corrección de la misma.

A papá por su doble función, una de ellas como guía en la vida y la otra como director en mi carrera de investigación.

Además quiero agradecer al Dr. Emilio Luque, por su aporte fundamental tanto sea en la corrección como en los aportes para la tesis, más significativo aun si se considera su poca disponibilidad de tiempo.

Al Dr. Guillermo Simari, por su buena voluntad y disponibilidad en todos los pedidos de colaboración durante la tesis.

A mis compañeros de trabajo, que me ayudaron, me acompañaron y me cubrieron cuando no me alcanzaba el tiempo; entre ellos especialmente a Sabrina, Ariel, Eduardo e Ismael.

Para finalizar gracias a mi familia que me ayudó en momentos difíciles de mi vida.

Índice

CAPÍTULO I

Procesamiento paralelo

1.1 Introducción

Un conjunto de satélites en el espacio exterior recoge un rango 10^{10} bits por segundo. Los datos representan información del tiempo meteorológico, la contaminación, la agricultura y los recursos naturales. Para que esta información sea útil necesita ser procesada con una velocidad de al menos 10^{13} operaciones por segundo.
Por otra parte, un equipo de cirujanos desea ver en un dispositivo especial una imagen para obtener una sección transversal de un órgano, observándolo en detalle y ejecutar una cirugía simulada, todo ello sin tocar al paciente. Una velocidad de procesamiento mínima de 10^{15} operaciones por segundo es necesaria para realizar eficazmente los pasos descritos.
En los ejemplos mencionados anteriormente se necesitan ordenadores muy rápidos para procesar grandes cantidades de datos o para permitir realizar un gran número de cálculos a gran velocidad.
Entre otras aplicaciones, con estas características se pueden mencionar el desarrollo de nuevos medicamentos, exploración de petróleo, planificación económica, criptoanálisis, gestión de base de datos, análisis médicos, reconocimiento de voz en tiempo real, análisis de imágenes, solución de grandes sistemas de ecuaciones, etcétera.
Desde el surgimiento de las computadoras seriales, su velocidad se ha incrementado para cumplir con las necesidades de las aplicaciones del mundo real. Sin embargo, la limitación física fundamentalmente impuesta por la velocidad de la luz no hace posible obtener mejoras indefinidamente, y una manera natural de evitar esta restricción es usar múltiples procesadores para resolver problemas. [1][2]
La única manera de resolver la cuestión es usar una técnica de cómputo avanzada denominada paralelismo. La idea es que si una tarea se descompone en una serie de operaciones, y estas operaciones se realizan simultáneamente, el tiempo que tarda en realizarse dicha tarea puede reducirse significativamente. Esta noción es intuitiva y es a lo que se está acostumbrado en una sociedad organizada.
Entre todas las ideas esparcidas por la Ciencia de la Computación en los últimos años, pocas han transformado el área de manera tan profunda como la computación paralela. Virtualmente todos los aspectos

se vieron afectados, y se generó un gran número de conceptos nuevos. Desde la Arquitectura de Computadoras hasta los Sistemas Operativos, desde los Lenguajes de Programación y Compiladores hasta Bases de Datos e Inteligencia Artificial, y desde la Computación numérica hasta las Combinatorias, cada rama sufrió un renacimiento [3].
Para sintetizar lo expuesto anteriormente podemos decir que, existen diversas razones que justifican la importancia y la necesidad del paralelismo, y entre ellas pueden mencionarse:

- el crecimiento de la potencia de cómputo, dado por la evolución de la tecnología de los componentes y las arquitecturas de procesamiento (supercomputadoras, hipercubos de procesadores homogéneos, redes de procesadores no-homogéneos, procesadores de imágenes, de audio, etcétera).
- La transformación y creación de algoritmos que explotan la concurrencia implícita en el problema a resolver, de modo de distribuir el procesamiento minimizando el tiempo de respuesta. Naturalmente esta transformación debe adaptarse a la arquitectura física de soporte para lograr reales mejoras.
- La necesidad de tratar con sistemas de tiempo real distribuidos, donde los requerimientos con relación al tiempo de respuesta son críticos.
- El límite físico alcanzado por las computadoras secuenciales, que en algunos casos torna inaceptable el tiempo para resolver determinados problemas, y hace que la solución distribuida sea la única factible.
- La existencia de sistemas en los que no es tan importante la velocidad de cómputo sino la necesidad de resolver problemas en más de una ubicación física a la vez, capacidad que puede asociarse a una configuración paralela.
- Las posibilidades que el paradigma paralelo ofrece en términos de investigación de técnicas para el análisis, diseño y evaluación de algoritmos. Conceptualmente, usar varios procesadores que trabajan juntos en una computación dada representa un paradigma interesante. Brinda ideas teóricas renovadas en la mayor parte de los problemas computacionales, independientemente de su origen o complejidad.
- La capacidad del cómputo distribuido/paralelo de reducir el tiempo de procesamiento en problemas de cálculo intensivo (simulaciones, búsquedas, cómputo científico, sistemas inteligentes) o de grandes volúmenes de datos (imágenes, video, bases de datos, etcétera).

Áreas numerosas y muy variadas presentan problemas que pueden ser resueltos mediante procesamiento paralelo y distribuido. Están incluidas las aplicaciones de predicción del clima, monitoreo de contaminación, procesamiento de datos de satélites, optimizaciones discretas, modelización oceánica, tomografías computadas, análisis de estructu-

ras de proteínas, procesamiento de imágenes, búsquedas en árboles, sorting de grandes volúmenes de datos, exploración petrolera, procesamiento de lenguaje natural, aprendizaje en redes neuronales, visión por computadora, procesamiento de consultas en bases de datos, reconocimiento de patrones, etcétera.

En teoría el paralelismo es simple: aplicar múltiples CPU's a un único problema. Además de ofrecer soluciones más rápidas, las aplicaciones paralelizadas pueden resolver problemas más grandes y más complejos cuyos datos de entrada o resultados intermedios exceden la capacidad de memoria de una CPU; las simulaciones pueden ser corridas con mayor resolución; los fenómenos físicos pueden ser modelizados de manera más realista. [4]

Claramente, la creación de algoritmos paralelos/distribuidos, o la transformación de un algoritmo secuencial en paralelo, se encuentra lejos de ser un proceso directo y está influida por la arquitectura física del soporte. Un *sistema paralelo* es la combinación de un algoritmo paralelo y la máquina sobre la cual este se ejecuta; ambos factores poseen numerosas variantes y de un adecuado "matching" entre ambos depende el éxito de la aplicación.

Otro aspecto a tener en cuenta es que los problemas son paralelizables en distintos grados. Para algunos, asignar particiones a otros procesadores podría significar mayor consumo de tiempo que realizar el procesamiento localmente. Otros problemas pueden ser completamente secuenciales. Un problema puede tener distintas formulaciones paralelas, lo que puede resultar en beneficios variados, y todos los problemas no son igualmente adecuados para el procesamiento paralelo.

Es importante referirse a un algoritmo paralelo mencionando el modelo de computación paralela para el que se lo diseñó. Esto se debe a que, a diferencia de la computación secuencial, donde la mayoría de las máquinas pertenecen a un mismo modelo, se han propuesto y usado un gran número de modelos para estudiar la computación paralela en teoría y para construir máquinas paralelas en la práctica.

Estos modelos difieren de acuerdo a si los procesadores se comunican entre sí por memoria compartida o por una red, si la interconexión es en forma de arreglo, árbol o hipercubo, si los procesadores ejecutan el mismo o distintos algoritmos, si los procesadores operan sincrónica o asincrónicamente, etcétera. Ninguno de los modelos ha logrado imponerse, ya que cada uno enfatiza determinados aspectos a costa de otros.

Respecto de los algoritmos, pueden ser especificados utilizando una diversidad de paradigmas (cliente/servidor, pipeline, dividir y conquistar, SPMD); otra forma de clasificarlos es por la utilización de paralelismo de datos o de control. Por el lado de las arquitecturas de soporte, si bien todas poseen más de un procesador, pueden diferir en

varias dimensiones tales como el mecanismo de control (clasificación de Flynn [6]), la organización del espacio de direcciones (memoria compartida y memoria distribuida), la granularidad de los procesadores, la red de interconexión (estática o dinámica), la sincronicidad (sincrónico o asincrónico), y la clase de procesadores utilizados (homogéneos y heterogéneos) [6] [7].

1.1.1 Arquitecturas paralelas

Una arquitectura paralela es una colección de elementos de procesamiento que se comunican y cooperan. Las aplicaciones paralelas aprovechan esto con el objetivo de resolver un problema común, tratando de reducir el tiempo de ejecución de la aplicación. Además también permiten resolver problemas de mayor tamaño aprovechando las características de la arquitectura utilizada (por ejemplo memoria).
Por el lado de las arquitecturas paralelas, si bien todas poseen más de un procesador, pueden diferir en varias dimensiones tales como el mecanismo de control (clasificación de Flynn en SISD, SIMD, MISD y MIMD), la organización del espacio de direcciones (máquinas de memoria compartida y de memoria distribuida), la granularidad de los procesadores, la red de interconexión (estática o dinámica), la clase de procesadores utilizados (homogéneos y heterogéneos).
Las arquitecturas para el procesamiento paralelo han evolucionado, y en la actualidad las redes de computadoras constituyen una plataforma de cómputo paralelo muy utilizada por sus ventajas en términos de la relación costo/rendimiento.
Los Sistemas Paralelos integran los algoritmos de procesamiento (Software) con la arquitectura de soporte (Hardware). Cuando se quiere obtener rendimientos razonables de una arquitectura multiprocesador, es necesario conocer con bastante nivel de detalle la configuración de hardware disponible y analizar (en base a este dato) los métodos y algoritmos de software que se propongan.
Dado que las redes de computadoras no han sido concebidas inicialmente para realizar cómputo paralelo, se deben identificar con la mayor claridad posible las capacidades en cuanto a: procesamiento, interconexión de los procesadores, sincronización y escalabilidad. Con estas características bien definidas, es posible analizar los métodos paralelos (numéricos y no numéricos) que hacen a las aplicaciones, y decidir si es factible utilizar la arquitectura en una dada aplicación con un rendimiento razonable, así como si se requiere o no transformar los algoritmos para adecuarlos al tipo de arquitectura propuesto.

Entre las arquitecturas paralelas con memoria distribuida podemos mencionar los cluster, multiclusters y grids [8] [9] [10] [11].

1.2 Cluster

El término *cluster* se aplica a los conjuntos de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora. Juegan hoy en día un papel importante en la solución de problemas de las ciencias, las ingenierías y del comercio moderno [13].
La tecnología de clusters ha evolucionado en apoyo de actividades que van desde aplicaciones de supercómputo y software de misiones críticas, servidores Web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.
El cómputo con clusters surge como resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que la requieran.
Simplemente, cluster es un grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio. De un cluster se espera que presente combinaciones de los siguientes servicios:

1. Alto rendimiento (High Performance)
2. Alta disponibilidad (High Availability)
3. Equilibrio de carga (Load Balancing)
4. Escalabilidad (Scalability)

La construcción de los ordenadores del cluster es más fácil y económica debido a su flexibilidad: pueden tener todos la misma configuración de hardware y sistema operativo (cluster homogéneo), diferente rendimiento pero con arquitecturas y sistemas operativos similares (cluster semi-homogéneo), o tener diferente hardware y sistema operativo (cluster heterogéneo).
Para que un cluster funcione como tal, no basta solo con conectar entre sí los ordenadores, sino que es necesario proveer un sistema de manejo del cluster, el cual se encargue de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento.
Entre los componentes de un cluster se pueden mencionar: los nodos (ordenadores o servidores), el sistema operativo, la conexión de red, el

middleware (capa de abstracción entre el usuario y los sistemas operativos), los protocolos de comunicación y servicios, y por último las aplicaciones (pueden ser paralelas o no).

Clusters homogéneos como máquinas paralelas

Un cluster homogéneo es aquel que está compuesto por todas máquinas de iguales características. En realidad es difícil encontrar una red local con todas las computadoras interconectadas exactamente iguales, a menos que la red local se haya instalado recientemente.

Aun así es útil considerar el caso homogéneo a los efectos de identificar/describir las características de los clusters desde el punto de vista de cómputo paralelo ya que permite una separación más clara de las características, mejorando por lo tanto su análisis; y la mayoría de las características que se identifiquen para el caso homogéneo serán compartidas por los clusters heterogéneos. Esto se debe a que, las redes de computadoras tienen características que en sí mismas se deben tener en cuenta a la hora de la paralelización de aplicaciones independientemente de que las máquinas sean homogéneas o no. [13]

Resumiendo, las principales características técnicas de las redes de estaciones de trabajo homogéneas que se deben tener en cuenta para el procesamiento paralelo son: acoplamiento débil y bajo rendimiento de la red de interconexión de procesadores.

El acoplamiento débil se puede sintetizar en los siguientes aspectos:

- la memoria física de las computadoras en la red local está totalmente distribuida y no hay ningún medio de hardware que facilite compartirla.
- El procesamiento en una red local es totalmente asincrónico en cada máquina, y no hay ningún medio de hardware que facilite la sincronización.
- La única forma de hacer que un procesador lea o escriba una posición de memoria en otra máquina es utilizando apropiadamente la red de comunicaciones. De la misma manera, la única forma de sincronización entre los procesadores de cada computadora de la red local es utilizando apropiadamente la red de comunicaciones.
- Descartar el uso de memoria compartida, lo cual significa replantear algunos algoritmos diseñados para computadoras paralelas con memoria compartida.
- El rendimiento de la red de interconexión se puede ver afectado por los siguientes aspectos:
- El tiempo propio de las comunicaciones es alto frente a los accesos a memoria.
- El costo del tiempo de latencia (startup) de inicio de la comunicación entre dos procesadores.

- El bajo ancho de banda (normalmente total o parcialmente compartido) disponible para los procesadores de cada cluster.
- El ancho de banda disponible para las comunicaciones "intercluster" que requieren un costo adicional de sincronización entre máquinas de diferentes clusters.

Ambas características de los clusters homogéneos llevan a priorizar el desarrollo de algoritmos con la mayor capacidad de cómputo local y la posibilidad de solapamiento entre comunicaciones inter-procesadores y procesamiento de datos sobre cada procesador.

Clusters heterogéneos como máquinas paralelas

La diferencia de velocidad de cómputo relativa de las computadoras en los clusters heterogéneos es el aspecto más relevante que se agrega a lo que ya se ha identificado en cuanto a los clusters homogéneos.
Por otro lado, la posibilidad de red de interconexión heterogénea es realmente muy poco probable en cada cluster local (aunque resulta un aspecto a considerar en los multiclusters como se verá más adelante).
En general, las redes locales instaladas pueden ser ampliamente heterogéneas en cuanto al hardware de procesamiento. En redes de procesadores con un mínimo tiempo de existencia (y evolución), se pueden encontrar diferentes computadoras (PCs, SMPs, Workstations). Más aún dentro de una línea de PCs con igual procesador, existen diferentes capacidades de procesamiento (asociadas por ejemplo al tamaño y velocidad de memoria caché o frecuencia de acceso al bus).[12] [13]
Resumiendo, las principales características técnicas de las redes de estaciones de trabajo heterogéneas que se deben tener en cuenta para el procesamiento paralelo son:

- tiempo de existencia de la red local, con su consiguiente impacto en la reposición y/o actualización de las computadoras.
- Evolución en cuanto a requerimientos, su escala se ve restringida en función de la red local que constituye el cluster.
- Para obtener el máximo rendimiento posible en una red heterogénea necesariamente se debe llevar a cabo un balance de carga de procesamiento adecuado. Este balance puede partir de un análisis estático de las características de la arquitectura (capacidad de cómputo, memoria, caché, entre otras), aunque puede requerir un estudio dinámico en función de la aplicación.

1.3 Multiclusters

La noción de "multicluster" es una generalización del modelo de arquitectura que permite que redes de computadoras dedicadas a una aplicación paralela se interconecten y puedan cooperar en el desarrollo de un algoritmo (o en la implementación de un sistema de software), compartiendo parcialmente recursos e incrementando la potencia de cómputo global. [12]
Las alternativas tecnológicas para la interconexión de los procesadores de cada cluster y entre clusters son muy amplias y están en constante evolución. Como ejemplos de conexión se pueden mencionar: EtherNet (en el caso de redes LAN) e interconectados por InterNet e InterNet2 (en el caso de redes WAN). La caracterización y estudio de rendimiento del soporte de comunicaciones será de especial importancia para la predicción y optimización de algoritmos paralelos que se ejecuten sobre esta clase de arquitecturas distribuidas.
Una clase especial de multiclusters son los multiclusters dedicados, en los cuales cada nodo de procesamiento es a su vez una red local multiprocesador (homogénea o heterogénea) y donde los subsistemas de comunicaciones que interconectan los nodos pueden ser diferentes (o de diferente rendimiento en el tiempo como el caso de InterNet). Esto requiere nuevos esfuerzos de investigación para tratar de modelizarla y establecer los parámetros que permitan predecir su performance.
Por otra parte todo el diseño algorítmico (tal como ocurriera con la utilización de clusters reemplazando máquinas paralelas de memoria compartida) debe replantearse, con el objetivo de optimizar el rendimiento sobre un nuevo modelo de arquitectura.

1.4 Grid

Un grid de cómputo es una infraestructura de hardware y software que provee acceso confiable, consistente, extensivo y barato a sistemas de cómputo con gran capacidad de procesamiento. Un grid es un entorno compartido implementado mediante una infraestructura de servicios permanente y basada en estándares capaz de soportar, crear y compartir recursos. Los recursos pueden ser computadoras, espacio de almacenamiento, instrumentos, aplicaciones (software), y datos, todos ellos conectados mediante una red (típicamente Internet) y una capa de software (middleware), la cual provee servicios de seguridad, monitoreo, gestión de recursos, etcétera. Los recursos pertenecen potencialmente a diferentes organizaciones y por lo tanto son compartidas bajo

políticas que definen qué se comparte, quién puede acceder y bajo qué condiciones puede hacerlo [12] [44]. El problema real y específico del concepto grid es la coordinación de los recursos compartidos y cómo resolver problemas bajo una organización virtual multi-institucional [45].

Algunas características de un entorno grid son [12] [14]:

- los recursos y servicios pueden incorporarse y retirarse dinámicamente del grid.
- Los recursos son heterogéneos, distribuidos geográficamente y normalmente conectados vía WAN.
- Los recursos son accesibles "on-demand" por un conjunto de usuarios autorizados que conforman una comunidad virtual.
- Se configura utilizando equipamiento de propósito general y protocolos estándar. Un objetivo es alcanzar un nivel definido de calidad de servicio.

Algunos autores consideran que un grid es un "cluster de clusters", lo que resulta una definición algo restrictiva pero útil para el desarrollo de aplicaciones paralelas que evolucionan de clusters a grid.

También se pueden mencionar algunas diferencias entre una arquitectura tipo cluster y una tipo grid:

- en un cluster normalmente se configura una única máquina paralela virtual que puede estar ejecutando una aplicación dedicada. Un grid permite configurar múltiples máquinas paralelas virtuales para varios usuarios/aplicaciones simultáneas.
- Tanto clusters como grids se basan en procesadores heterogéneos. Sin embargo en el grid esta heterogeneidad se extiende a la red de comunicaciones y al tipo de componentes en cada nodo que pueden ser procesadores, instrumentos, sensores, entre otros.
- El middleware necesario para grid es más complejo que el de los clusters, generando un overead adicional [8]. Fundamentalmente, para configurar la máquina paralela virtual es necesaria una etapa de identificación de recursos físicos y su ubicación. Además en el grid es necesario monitorear la ejecución de tareas sobre múltiples máquinas virtuales con usuarios de diferente nivel y con distintos derechos de acceso a los recursos.
- Asimismo las herramientas para el desarrollo de aplicaciones requieren un mayor nivel de abstracción en grid, por la complejidad y variedad de los múltiples usuarios que pueden utilizar la arquitectura.

Es interesante notar que una estructura de multicluster, visualizada como un número limitado de clusters dedicados que cooperan en una única aplicación paralela, es un punto intermedio entre clusters y grid y requerirá algunos servicios especiales en su middleware (especial-

mente para autenticar derechos de usuarios que acceden a recursos remotos) [8].

1.5 Métricas paralelas

En los modelos secuenciales, la performance generalmente es medida teniendo en cuenta los requerimientos de tiempo y memoria de un programa. En la práctica, los requerimientos de memoria son importantes solo para que haya suficiente espacio disponible para resolver instancias del tamaño deseado; en la mayoría de los casos no hay beneficios por usar menos memoria, más allá de los económicos. Esta suposición mantiene como métrica de performance solo al tiempo.
Cuando se utiliza un algoritmo paralelo para la resolución de un problema, interesa saber cuál es la ganancia en la performance obtenida. En la computación paralela, como en la serial, las métricas más utilizadas son tiempo y memoria. Entre métodos alternativos que utilizan diferente cantidad de memoria preferiríamos el más rápido, es decir, no hay ventaja por utilizar menos memoria a menos que ese menor uso resulte en una reducción del tiempo. Así el tiempo de ejecución o complejidad en tiempo de un programa paralelo sigue siendo una métrica importante [2], aunque existen otras medidas que suelen tenerse en cuenta en el mundo paralelo siempre que favorezcan a sistemas con mejor tiempo de ejecución.
En el procesamiento paralelo no existe un modelo unificador de cómputo paralelo, el tiempo de ejecución de un algoritmo depende no solo del tamaño de su entrada sino también de la arquitectura de la máquina paralela y el número de procesadores. Por lo tanto un algoritmo paralelo no puede ser evaluado aisladamente de la máquina en que se ejecuta. Un sistema paralelo es la combinación de un algoritmo y la arquitectura paralela sobre la cual está implementado.
La variedad en los algoritmos paralelos hace complicado el análisis de performance del sistema paralelo. Algunas de las preguntas que uno se puede realizar son, ¿qué interesa medir?, ¿qué indica que un sistema paralelo es mejor que otro?, ¿qué sucede si se utiliza un algoritmo con mayor cantidad de procesadores?
En la medición de performance paralela es usual elegir un problema y testear el tiempo de ejecución variando la cantidad de procesadores. En este modelo aparecen definiciones como speedup y eficiencia.
En el estado actual de la tecnología es posible construir computadoras paralelas que emplean miles de procesadores, y la disponibilidad de tales sistemas llevó a interesarse en la performance de los mismos. Al resolver un problema en paralelo es razonable esperar una reducción en el tiempo

de ejecución que sea proporcional a la cantidad de recursos de procesamiento empleados. La escalabilidad de un algoritmo paralelo sobre una arquitectura paralela es una medida de su capacidad de usar efectivamente un número creciente de procesadores. El análisis de escalabilidad de una combinación algoritmo–arquitectura paralela puede usarse para una variedad de propósitos, como caracterizar la cantidad de paralelismo inherente en un algoritmo paralelo, o estudiar el comportamiento con respecto a cambios en parámetros de hardware tales como la velocidad de los procesadores y canales de comunicación [2].
Entre las métricas utilizadas en el procesamiento paralelo se encuentran desbalance de carga de trabajo de tareas y procesadores, demoras debido a la sincronización, overhead por scheduling, topología, efecto de la granularidad, grado de escalabilidad del sistema, mapeo de procesos y datos a procesadores que pueden acarrear mayor o menor comunicación, distintos niveles de memoria involucrados, etcétera.

Tamaño del problema
Se define el tamaño del problema (W) como una medida del número total de operaciones básicas necesarias para resolverlo. Dado que puede haber varios algoritmos distintos para resolver el mismo problema, para mantener único el tamaño se lo define como el número de operaciones básicas requeridas por el algoritmo secuencial conocido más rápido en un solo procesador [15] [16] [17].

Tiempo de ejecución de un algoritmo paralelo
El tiempo de ejecución de un algoritmo paralelo (Tp) es el tiempo transcurrido desde el momento en que comienza a ejecutarse el algoritmo paralelo hasta que el último procesador termina su ejecución. Para un sistema paralelo dado, Tp normalmente es una función del tamaño del problema (W) y el número de procesadores (p) y suele escribirse como Tp(W,p).

Speedup
Una de las mediciones de performance más usadas en el dominio paralelo intenta describir cuánto más rápido corre la aplicación sobre una máquina paralela. En otras palabras, cual es el beneficio derivado del uso de paralelismo, o cuál es el speedup que resulta [2] [16] [17].
El speedup es el cociente (S) entre el tiempo de ejecución serial del algoritmo serial conocido más rápido (Ts) y el tiempo de ejecución paralelo del algoritmo elegido (Tp):

$$S = \frac{Ts}{Tp}$$

Overhead paralelo
El overhead paralelo total (To) es la suma de los overheads en que incurren todos los procesadores debido al procesamiento paralelo. Incluye los costos de comunicación, trabajo no esencial y tiempo ocioso debido a la sincronización y componentes seriales del algoritmo:

$$To = \quad pTp - Ts$$

Asumiendo que To es una cantidad no negativa, el speedup está acotado por p. Para un sistema paralelo dado To normalmente es una función de W y p, por lo que suele denotarse To(W,p) [16] [17].

Eficiencia
La eficiencia (E) es una medida de performance paralela estrechamente relacionada con el speedup. Está dada por el cociente entre el speedup (S) y el número de procesadores (p):

$$E = \quad \frac{S}{p} = \frac{Ts}{pTp} = \frac{1}{1 + \frac{To}{Ts}}$$

Puede pensarse en la eficiencia como el speedup promedio por procesador. Los procesadores no brindan 100 por ciento de su tiempo para cómputo, de modo que la eficiencia mide la fracción de tiempo que son útiles.
El valor de eficiencia se encuentra entre 0 y 1, dependiendo del grado de efectividad con el cual se utilizan los procesadores. Cuando es 1 el speedup es perfecto [2] [16] [17].

Costo
El costo de un sistema paralelo se define como el producto del tiempo de ejecución paralelo (Tp) y el número de procesadores utilizados (p). Refleja la suma del tiempo que cada procesador utiliza resolviendo el problema.
Se dice que el sistema paralelo es de costo óptimo si y solo si el costo es asintóticamente del mismo orden de magnitud que el tiempo de ejecución serial, es decir pTp = O(W). Esto es, el costo de resolver un problema en una máquina paralela es proporcional al tiempo de ejecución del algoritmo secuencial conocido más rápido en un solo procesador. Dado que la eficiencia es el cociente entre el costo secuencial y

el costo paralelo, un sistema paralelo de costo óptimo tiene una eficiencia de O (1) [2] [16] [17].

Grado de concurrencia
El grado de concurrencia o grado de paralelismo C(W) es el número máximo de tareas que pueden ser ejecutadas simultáneamente en cualquier momento en el algoritmo paralelo. Para un W dado, el algoritmo paralelo no puede usar más de C(W) procesadores. C(W) depende solo del algoritmo paralelo, y es independiente de la arquitectura. Es una función discreta de tiempo, y refleja cómo el paralelismo de software matchea con el de hardware [2] [16] [17].
Así definido, el grado de concurrencia supone un número ilimitado de procesadores y otros recursos necesarios disponibles, aunque esto no siempre puede ser alcanzable en una computadora real con recursos limitados.

Resumen del capítulo

En este capítulo se definieron los conceptos relacionados con las aplicaciones y arquitecturas paralelas. Es decir los conceptos de cluster, multicluster, grid, sistema paralelo y modelo paralelo. Por otra parte, se mencionó el concepto de métrica, las cuales sirven (en su mayoría) para evaluar la performance o el comportamiento del algoritmo paralelo, entre las métricas se mencionaron: speedup, eficiencia, overhead paralelo, costo, grado de concurrencia.

CAPÍTULO II

Modelado de aplicaciones paralelas

2.1 Introducción

En general toda solución a un problema paralelo requiere del desarrollo de las siguientes fases: detección de paralelismo, definición del grafo de tareas y asignación de tareas a procesadores. En la figura 2.1 se muestran dichas fases [3] [18] [19].

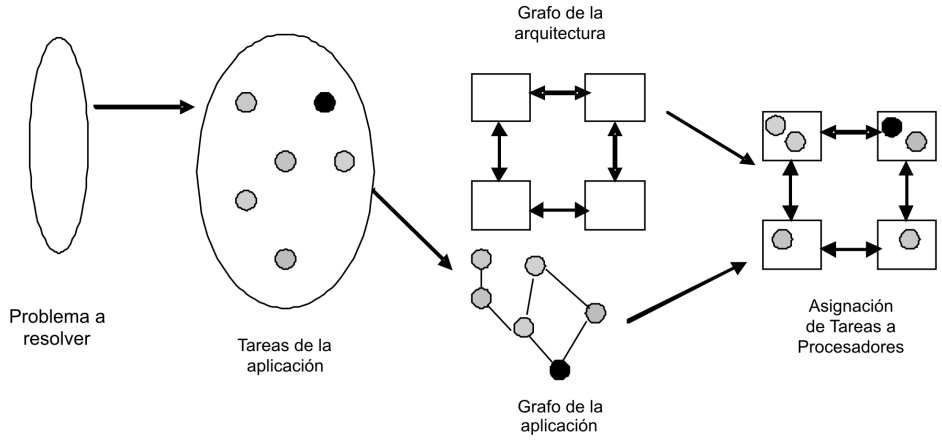


Figura 2.1 – Fases de la aplicación

La fase de *detección del paralelismo* consiste en poder determinar y descomponer las actividades secuenciales que pueden ser realizadas en paralelo, con sus interdependencias. Cada una de estas actividades se denominan fases de cómputo, siendo esta la mínima unidad de concurrencia que puede ser ejecutada en paralelo. El número de fases de cómputo disponibles puede variar a lo largo de la ejecución del programa, y el número máximo de tareas disponibles simultáneamente (grado de paralelismo) proporciona la cota superior sobre el número de procesadores a utilizar para ejecutar una aplicación paralela.
Por lo tanto el objetivo de esta fase no es solo la detección de las actividades que pueden hacerse en paralelo, sino alcanzar una descomposición de estas actividades de tal forma que el programa disponga de la concurrencia suficiente para tener los procesadores ocupados al

máximo durante su ejecución, manteniendo al mismo tiempo un overhead aceptable para el manejo de las distintas fases de cómputo. Cabe hacer notar que estos objetivos son difíciles de conseguir (ya que en ambos casos se contraponen), y el diseño de un buen algoritmo está supeditado a la experiencia del programador, y diferentes decisiones en esta etapa pueden llevar a distintas soluciones de rendimiento.
La segunda fase es la *definición del grafo de tareas de la aplicación*, el cual utiliza las tareas identificadas en la fase anterior. En esta etapa se deben identificar las interacciones entre las tareas, que se llevan a cabo mediante un mecanismo de comunicación. Además aquí es necesario definir el grafo de la aplicación y en el mismo deben considerarse problemas importantes como: la granularidad de las tareas, descomposición de los datos a tratar por cada tarea y organización de las comunicaciones entre tareas.
El concepto de granularidad está asociado a la cantidad de trabajo que realiza cada una de las tareas que forma la aplicación y su relación con la comunicación. La granularidad es del tipo fino cuando el grafo está formado por gran cantidad de tareas que ejecutan una pequeña parte de cómputo global, y se comunican frecuentemente. El caso opuesto sería considerado granularidad gruesa (mucho cómputo y poca comunicación).
La descomposición de los datos es un factor importante en el modelo de programación paralela con memoria distribuida donde las comunicaciones suponen un costo alto.
Por último la organización de las comunicaciones implica determinar los mecanismos de comunicación y sincronización, y la cantidad de datos a transmitir entre las mismas.
El grafo resultante de la aplicación estará formado por un conjunto de tareas que se comunican y que realizan iguales o distintas funciones según el modelo de programación al que pertenecen (aplicaciones con paralelismo de datos, aplicaciones con paralelismo funcional y aplicaciones mixtas).
La tercera fase es la *asignación de tareas a procesadores*. En esta fase se busca realizar la asignación de manera de minimizar el tiempo de ejecución final. Dicha asignación se lleva a cabo mediante un proceso de scheduling, el cual determina dónde y cuándo debe ejecutarse cada tarea, es decir en qué procesador y en qué orden se ejecutarán las tareas de la aplicación. Cuando el proceso de asignación se centra únicamente en decir dónde deben ejecutarse se denomina mapping.
La asignación de tareas puede realizarse en forma dinámica durante la ejecución (donde se tienen en cuenta parámetros que reflejan la situación del sistema en cada instante de tiempo debido a que no se conoce

el trabajo requerido por la aplicación antes que la misma sea ejecutada), o en forma estática momentos previos a realizar la ejecución.
En este trabajo se utilizó la asignación estática. Dado que la misma se realiza de manera previa a la ejecución de la aplicación, el overhead requerido por el algoritmo de scheduling no tiene efectos sobre la ejecución de la aplicación. Este mecanismo brinda buenos resultados para aplicaciones con patrones de comportamiento estáticos conocidos.
Para realizar esta asignación mediante el scheduling, se debe encontrar una forma de poder representar las características más importantes del comportamiento del programa paralelo. Entre ellas se puede nombrar: tiempo de cómputo de cada tarea, comunicación entre tareas, dependencias de datos, requerimientos de sincronización. Una manera de poder representar todas estas características es lo que se denomina *grafo ponderado*.
Es importante hacer notar que las estrategias de asignación aplicables generarán resultados esperados en cuanto al tiempo de ejecución, siempre que los parámetros estimados en el modelo guarden relación con los del programa real. Cuando estos parámetros sufran grandes variaciones dependiendo de los datos de entrada, la asignación estática no será adecuada y habrá que utilizar la dinámica que realice la asignación en función de los valores que se van dando en tiempo de ejecución [18].
Existen numerosos modelos para representar una aplicación paralela, cada uno tiene sus ventajas y desventajas. En las siguientes secciones se analizan alguno de ellos.

2.2 Modelos de comportamiento de aplicaciones paralelas existentes

Teniendo en cuenta el punto de vista del programador, el desarrollo de las aplicaciones paralelas se puede llevar a cabo según dos modelos de cómputo [20]:

- ✓ Modelo implícito: el programador escribe su programa en forma secuencial, sin ocuparse de tener en cuenta estrategias para explotar el paralelismo de la aplicación. El compilador es el que se encarga de detectar el paralelismo dentro del algoritmo secuencial, de definir el grafo que modela la aplicación y realizar su ejecución de manera transparente al usuario. En general no permiten obtener el máximo de ganancia por la aplicación del paralelismo, ya que el programador no tiene el control total de los detalles.

- ✓ Modelo explícito: el algoritmo paralelo que resuelve una aplicación es definido por completo por el programador. La mayoría de los estándares de programación actuales pertenecen a esta categoría. A su vez estos están divididos en dos categorías los de alto nivel (el programador define únicamente el conjunto de tareas y las actividades que realiza cada una de ellas, ejemplos: High Performance Fortran, Dataparallel C y Fortran D); y los de bajo nivel (aquellos en los que el programador se encarga del manejo de la sincronización y comunicación entre tareas de manera explícita, ejemplos: PVM y MPI).

Esta tesis se centra en modelar el comportamiento de las aplicaciones con paralelismo explícito definidas mediante el pasaje de mensajes con el propósito de lograr una asignación estática eficiente. Esto se debe a que estos entornos no imponen restricciones en cuanto a número de tareas que forman la aplicación, ni la granularidad e interconexión de las mismas.
Entre los modelos basados en grafos que se utilizan para representar aplicaciones paralelas se puede mencionar: Grafo de Precedencia de Tareas (Task Precedence Graph TPG), Grafo de Interacción de Tareas (Task Interaction Graph TIG) y Temporal Task Interaction Graph (TTIG) [21] [22] [23].

2.2.1 Grafo de Precedencia de Tareas (TPG)

La aplicación se representa como un grafo dirigido acíclico G= (N,E), donde N es el conjunto de nodos del grafo (cada uno representa una tarea), y E es el conjunto de arcos entre tareas adyacentes. Dichos arcos marcan tanto las comunicaciones como las relaciones de precedencia entre tareas, e indican el orden parcial de ejecución de las mismas.
A cada tarea $T_i \in N$ se le asigna un valor no negativo $w(T_i)$, que representa su tiempo de cómputo estimado, y a cada arco $(T_i,T_j) \in E$ se le asocia un valor $c(T_i,T_j)$, que indica el volumen de comunicación que se transmite de T_i a T_j durante la ejecución.
La figura 2.2 muestra un ejemplo de grafo TPG con seis tareas $\{T_0..T_5\}$. Del ejemplo mostrado puede verse que las tareas T_0 y T_2 no tienen ninguna precedencia y por lo tanto pueden comenzar a ejecutarse desde el primer momento. Caso contrario ocurre con las tareas T_1 y T_4 las cuales deben esperar que T_0 y T_2 terminen su ejecución respectivamente para que luego les comuniquen sus datos; a su vez la tarea

T_3 debe esperar a que la tareas T_1 y T_2 terminen su ejecución y así recibir los datos enviados por las mismas. Por último, de la figura se desprende que la tarea T_5 debe esperar a que las tareas T_3 y T_4 terminen su ejecución y que luego le envíen sus datos para poder comenzar a ejecutar.
El volumen de información que se envía entre tareas está dado por el valor que se encuentra al lado de cada arco. Además, el tiempo que requiere para su ejecución cada tarea se encuentra dentro del círculo que la representa junto al número de tarea.

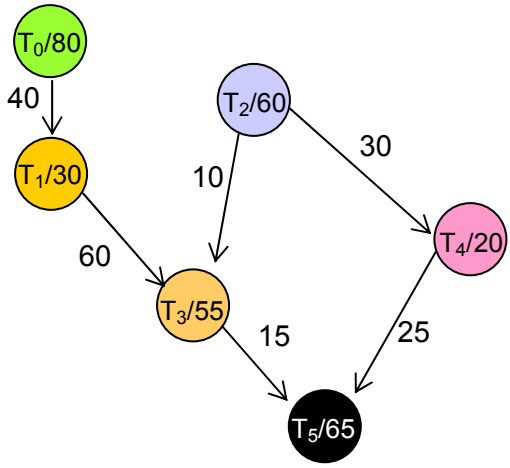


Figura 2.2 – Grafo TPG

En este tipo de grafos se representan las relaciones de precedencia de forma explícita, por lo tanto suele ser adecuado para modelar las aplicaciones paralelas de granularidad fina cuyas tareas se comunican únicamente al principio y al final de su ejecución. Ejemplos de aplicaciones con un patrón de interacciones adecuado para ser modeladas como un TPG pueden encontrarse en el ámbito de las aplicaciones numéricas como las de cálculo matricial y transformadas de Fourier.

2.2.2 Grafo de Interacción de Tareas (TIG)

En este caso el programa paralelo se modeliza mediante un grafo no dirigido G= (V,E), donde V es el conjunto de vértices, cada uno representando a una tarea del programa, y E es el conjunto de aristas que representan las interacciones entre tareas adyacentes. Al igual que sucede en el modelo TPG, en el TIG se asocian pesos a los vértices y aristas, representando los tiempos de cómputo de las tareas y los volúmenes de comunicación que se intercambian respectivamente.
TIG no incluye ninguna información relativa al orden de ejecución de las tareas y por lo tanto la mayoría de los autores asumen que todas las tareas pueden ejecutarse concurrentemente.
La figura 2.3 muestra un grafo TIG en el cual se visualizan cuatro tareas $\{T_0..T_3\}$, en este se muestra el tiempo de cómputo que requiere cada tarea junto al costo de comunicación entre las mismas, sin embargo el grafo no refleja el orden en el cual deben ejecutarse las tareas, ni cuando se produce la comunicación.

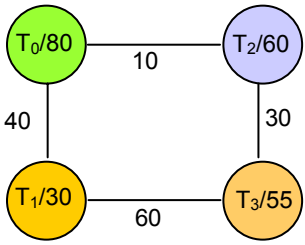


Figura 2.3 – Grafo TIG

Este modelo es menos refinado que el TPG, puesto que no captura información temporal relativa al orden de ejecución de la tareas, pero resulta adecuado para modelar las aplicaciones con paso de mensajes de granularidad gruesa que alternan fases de cómputo y comunicación dentro de las tareas [24] [25][26][27][28].

2.2.3 Grafo Temporal de Interacción de Tareas (TTIG)

En los entornos de programación, en general, las aplicaciones con paso de mensajes están formadas por un conjunto de tareas con granularidad gruesa, las cuales realizan una determinada función sobre un conjunto de datos propios. El pasaje de información entre las tareas se

lleva a cabo mediante sentencias de comunicación de envío y recepción. De los grafos clásicos estudiados, el adecuado para modelar estáticamente la aplicación dependerá de la estructura de tareas definida por el usuario.
Supongamos como ejemplo el algoritmo de Floyd [29]; dado un grafo de n nodos y una matriz de distancias A(n*n) entre los mismos, el algoritmo calcula el valor de mínima distancia para cada par de nodos Ai y Aj, comparando el valor de Aij actual con la distancia que se obtiene pasando por el camino Ai -> Ak -> Aj, donde Ak es un nodo intermedio en el camino Ai -> Aj.
Una solución paralela de este algoritmo consiste en dividir la matriz inicial en m particiones de n/m columnas contiguas de A, y aplicar Floyd a cada partición separadamente [18]. En este caso, el programa estará formado por una tarea TM, que descompone la matriz en varias particiones y compone el resultado final, y un conjunto de tareas TS_i que calculan el algoritmo de Floyd para su partición de datos.

TM	**TSi**

Obtener matriz A
Crear m tareas TSi
For i = 1 to m
Crear partición
Enviar partición a la tareas TSi
For i = 1 to m recibir resultados
Componer la matriz de distancias mínimas
	Recibir partición
	For k = 1 to n-1
	If (k pertenece a mi partición)
	Enviar columna k al resto de las tareas TSi
	Calcular Floyd
	Else
	Recibo columna k de las otras tareas TSi
	Calcular Floyd
	Enviar resultado a TM para obtener la matriz de distancias mínimas

Como se ve en el seudocódigo, cada tarea TS_i hará k iteraciones y en cada iteración la tarea TS_i procesa la parte de la matriz que le corresponde recorriéndola mediante los índices i y j. Como el cálculo de cada A_{ij} se realiza en base a los valores intermedios de A_{ik}, a cada iteración de k la tarea posee la columna k de la matriz y debe enviarla a los demás. Al finalizar el cómputo, cada tarea TS_i envía los resultados a la tarea TM, quien compone la matriz resultado.
Si se considera un ejemplo donde el algoritmo trabaja con una matriz de 10 elementos en dos particiones, entonces el programa contará con tres tareas: TM (encargada de repartir las particiones correspondientes y luego generar la matriz resultado final), y TS_1 y TS_2 (las cuales realizan el cálculo del algoritmo de Floyd de sus particiones correspondientes). La figura 2.4 muestra la interacción entre las tareas.

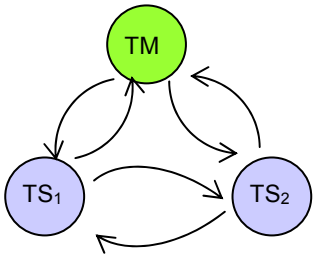


Figura 2.4 – Interacción entre las Tareas

De los dos modelos planteados previamente TPG y TIG el último es el que más se adecua a esta aplicación, ya que las tareas S_1 y S_2 tienen interacciones reiteradas en puntos internos de las mismas durante su ejecución. La figura 2.5 muestra esta interacción para el modelo TIG.

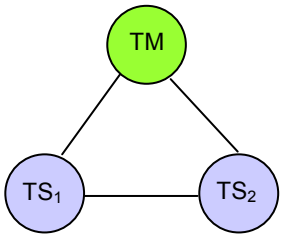


Figura 2.5 – Interacción de tareas para el modelo TIG

Si deseáramos representar esta aplicación mediante el modelo TPG deberíamos abrir la iteración y subdividir las tareas S_1 y S_2 de tal forma que cada iteración del bucle forme una tarea del grafo TPG, y así las tareas queden dispuestas como un grafo dirigido acíclico con las interacciones entre las tareas por el principio y final de las mismas. La figura 2.6 muestra el grafo correspondiente al modelo TPG.

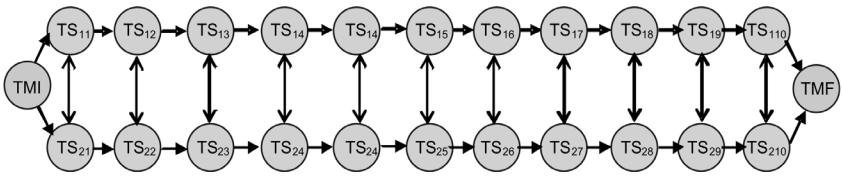


Figura 2.6 – Grafo correspondiente al modelo TPG

De los gráficos se puede observar que el modelo TIG usa como modelo un grafo con las mismas tareas definidas por el programador, pero no refleja ninguna información sobre el orden de ejecución de las mismas. En el caso del TPG si queda explícito en el grafo el orden parcial de ejecución de las tareas, pero el grafo está formado por muchas más tareas que las definidas por el programador.
Para solucionar estos problemas se desarrolló el modelo TTIG [18] [37] el cual es una combinación de los dos modelos anteriores. Extrayendo las características más sobresalientes de cada uno tratando de solucionar los problemas presentados por los modelos TIG y TPG, introduce el grado de concurrencia.
El grafo del modelo TTIG está formado por el mismo número de nodos que las tareas que componen el programa, e incluye además de los costos de cómputo y comunicación, el máximo grado de concurrencia entre las tareas adyacentes (tareas que se comunican), en virtud de las dependencias mutuas; es decir el máximo tiempo en que las tareas pueden ejecutarse en paralelo. El uso del modelo TTIG permite representar las aplicaciones paralelas de forma más realista, y constituye una alternativa que unifica los dos modelos anteriores.
La información adicional sobre la posibilidad de concurrencia entre tareas adyacentes es un información fundamental a tener en cuenta en el proceso de asignación de tareas a procesadores, puesto que las tareas con escasa posibilidad de ejecutarse paralelamente pueden ser buenas candidatas a ser asignadas a un mismo procesador, mientras que las que tienen gran potencial de concurrencia pueden ser asignadas a diferentes procesadores. [18]

2.2.4 Limitaciones de los modelos anteriores

Los modelos presentados anteriormente manifiestan un conjunto de limitaciones para representar ciertas situaciones en aplicaciones paralelas.
Como se mencionó anteriormente, TIG usa como modelo un grafo con las mismas tareas definidas por el programador, pero no refleja ninguna información acerca del orden de ejecución de las mismas. En el caso del TPG si queda explícito en el grafo el orden parcial de ejecución de las tareas, pero dicho grafo está formado por muchas más tareas de las que en realidad tiene el programa inicial, con lo cual para que dicho programa pueda ser tratado como un TPG como tal, será necesario definir en el programa cada subtarea como una tarea a efectos de la asignación.

Es importante notar que, aunque el modelo TPG es más refinado, plantea problemas a la hora de usarlo para los programas paralelos con paso de mensajes, donde las tareas las define el programador y pueden incluir primitivas de comunicación en cualquier punto.
El modelo TTIG tiene en cuenta más características de la aplicación que los modelos previamente mencionados (TIG,TPG); esto permite reducir el tiempo de ocio de los procesadores al ser utilizado para generar una asignación más eficiente.
En la actualidad, la mayoría de las aplicaciones paralelas son ejecutadas sobre clusters/multiclusters de procesadores heterogéneos, ya que es difícil formar una arquitectura cuyos procesadores tengan exactamente las mismas características. Una de las desventajas de los tres modelos mencionados anteriormente (TPG, TIG, TTIG) es que al momento de ser utilizados para realizar la asignación de procesos a procesadores no consideran la posible heterogeneidad de la arquitectura (en cuanto a los procesadores y a la red de interconexión).

Resumen del capítulo

En este capítulo se definieron los modelos existentes TPG, TIG y TTIG para representar aplicaciones paralelas mediante grafos. El último de los modelos mencionados soluciona problemas en la representación de diferentes tipos de aplicaciones paralelas.
En el final de este capítulo se menciona las limitaciones del modelo TTIG lo cual genera la necesidad de construir otro modelo que trate de subsanar las limitaciones mencionadas anteriormente.

CAPÍTULO III

Algoritmos de asignación de tareas a procesadores

3.1 Introducción

Una de las fases de la solución de aplicaciones paralelas es la asignación de tareas a procesadores. En esta fase se trata de realizar la asignación de manera de minimizar el tiempo final de la aplicación. Dicha asignación se lleva a cabo mediante un proceso de scheduling, el cual, determina dónde y cuándo debe ejecutarse cada tarea, es decir, en que procesador y en qué orden dentro de cada procesador deben ejecutarse las tareas de la aplicación. Cuando el proceso de asignación se centra únicamente en decir donde deben ejecutarse las tareas se denomina mapping.

Es vital para una aplicación paralela disponer de un scheduling eficiente para lograr una alta performance en la ejecución [30].

La asignación de tareas puede realizarse en forma dinámica durante la ejecución; en este caso no se tienen en cuenta características del comportamiento de la aplicación pero si puede hacerse teniendo en cuenta los parámetros que reflejan la situación del sistema en cada momento de la ejecución. O en forma estática cuando el tiempo de ejecución de tareas, dependencias comunicación y sincronización son conocidas a priori.

El *objetivo de un scheduling estático* es asignar los nodos del grafo de tareas a los procesadores minimizando el schedule (esquema que indica cuales son las tareas de la aplicación y en el momento en que cada una debe ejecutarse teniendo en cuenta las comunicaciones entre las mismas y sin violar las restricciones de precedencia). Para que un schedule sea eficiente debe tener una longitud corta y el número de procesadores utilizados ser razonable [31] [32]. Este tema también se encuentra explicado en [33] [34] [35].

El *scheduling dinámico* realiza actividades de planificación concurrentemente en tiempo de ejecución y se aplica a problemas dinámicos. Es un enfoque general adecuado para un amplio rango de aplicaciones, ya que puede ajustar la distribución de la carga basado en información del sistema en tiempo de ejecución, pero en muchos casos no utiliza información adicional que incluya la carga global de la aplicación [30].

Una estrategia que combine las ventajas del scheduling estático y dinámico debe ser capaz de generar una carga balanceada sin incurrir en un gran overhead. Esto es posible con técnicas avanzadas de scheduling paralelo, en el cual todos los procesadores cooperan para planificar el trabajo.
En este trabajo se utilizará la asignación estática mediante el scheduling. Dado que la asignación se realiza de manera previa a la ejecución de la aplicación, el overhead requerido por el algoritmo de asignación no tiene efectos sobre la ejecución de la aplicación.
Para realizar esta asignación, se debe encontrar una forma de representar las características más importantes del comportamiento de la aplicación paralela, entre ellas se pueden mencionar: tiempo de cómputo de cada tarea, comunicación entre tareas, dependencia de datos y requerimientos de sincronización. Estas características pueden ser representadas por medio de los grafos ponderados descriptos en el capítulo II.
En estos grafos, un nodo representa un tarea, y a cada nodo se asocia su costo de cómputo ($w(n_i)$) que indica el tiempo de ejecución. Los arcos corresponden a los mensajes de comunicación y restricciones de precedencia entre los nodos. Cada arco tiene asociado un número que indica el tiempo requerido para comunicar los datos de un nodo a otro (costo de la comunicación $c_{i,j}$). Los nodos origen y destino se denominan padre e hijo respectivamente. Un nodo no puede comenzar su ejecución antes de recibir el mensaje de su/s padre/s.

3.2 Algoritmos de asignación de tareas

Entre los algoritmos relacionados con el problema de asignación pueden mencionarse: Critical Path (CP), Static Scheduling Edge-Zeroning algorithm (EZ), Modified Critical Path algorithm (MCP), Mobility Directed algorithm (MD), Earliest Task Fisrt algorithm (ETF), Dynamic Level Scheduling algorithm (DLS), Heterogeneuos Relative Mobility Scheduling algorithm (HRMS), Duplication Scheduling Heuristics (DSH), Mesh Walking algorithm (MW), entre otros [30] [36].
Algunos de estos algoritmos se describen a continuación.

3.2.1. Critical Path (CP)

Este algoritmo se basa en la utilización del grafo de tareas, el cual contiene todas las tareas que forman la aplicación y refleja las comunicaciones que se dan entre las mismas.

El camino crítico de un grafo es un conjunto de nodos y arcos que forman un camino desde un nodo de entrada a un nodo de salida, donde la suma del costo de cómputo más la comunicación es máxima. Este camino se forma teniendo en cuenta las relaciones de precedencia y las comunicaciones entre las tareas que forman el grafo.
Cualquier retraso generado en las tareas incluidas dentro del camino crítico genera un retraso en la ejecución de la aplicación.
Una virtudimportante del camino crítico es que, de entretodas las tareas que conforman la aplicación, permitecentrarse solo sobre aquéllas que son realmente relevantes para la duración de la misma. Otra concepción del camino crítico es que da una cota inferior del tiempo de ejecución de la aplicación.
La figura 3.1 muestra un grafo compuesto por 5 tareas {T0..T4} que representan cada uno de los nodos del grafo y las comunicaciones entre los mismos. Cada nodo incluye el costo de ejecutar la tarea, la el arco de comunicación indica el costo de la misma.

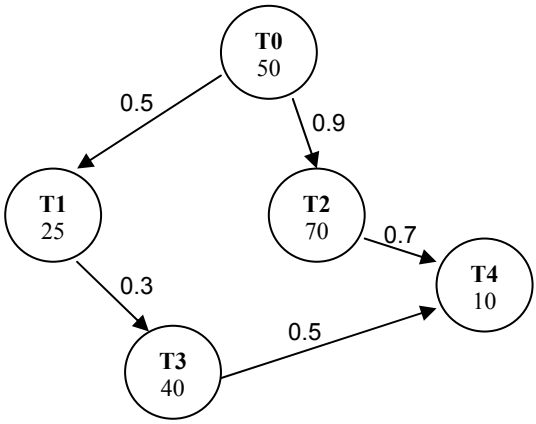


Figura 3.1 – Grafo para calcular Camino Crítico

Para el grafo anterior el camino crítico estaría formado por las tareas {T0,T2,T4}. En este momento puede notarse que este camino no es el de mayor longitud en el grafo.

3.2.2 Static Scheduling Edge-Zeroning algorithm (EZ)

En contraposición con los algoritmos basados en CP, EZ trata de reducir la longitud parcial de la planificación en cada etapa, al considerar el arco de mayor costo en el grafo de tareas.
En cada etapa de la planificación, el algoritmo planifica los dos nodos cuyo arco es el de mayor costo de comunicación para el mismo procesador. Esto ocurre siempre y cuando la longitud parcial de la planificación no aumente. Para esta tarea (planificación) el algoritmo primero construye una lista de arcos en orden descendente de costos de comunicación. Luego extrae el primer arco de la lista y planifica los dos nodos incidentes para el mismo procesador si la longitud parcial de la planificación no se ve incrementada. Si esto no ocurre (o sea, aumenta), los nodos son planificados en dos procesadores distintos. Los nodos dentro de un mismo procesador se mantienen en un orden descendente de sus niveles (el nivel de un nodo es la distancia más costosa en cuanto a tiempo de cómputo desde el nodo hasta la salida). Este proceso se repite hasta que se hayan planificado todos los nodos.
La complejidad del algoritmo es O (e(e+v)), siendo v la cantidad de nodos y e la cantidad de arcos del grafo.

3.2.3 Modified Critical Path (MCP)

El algoritmo MCP está diseñado en base a un atributo: el *tiempo de inicio más tardío* de un nodo (ALAP). El tiempo de inicio más tardío posible de un nodo se determina mediante una restricción as-late-as-possible (ALAP binding), que se lleva a cabo atravesando el grafo de tareas hacia arriba desde los nodos de salida hacia los nodos de entrada y llevando los nodos lo más abajo posible que permita la longitud del CP.
Este algoritmo primero calcula todos los ALAP posibles para cada nodo. Luego, cada nodo se asocia con una lista de los ALAP posibles, o sea, con el tiempo de inicio de cada nodo, seguido por un orden descendente de tiempo de inicio más tardío posible de sus nodos hijos. Como siguiente paso, MCP construye una lista de nodos en un orden lexicográfico descendente de las listas de los tiempos de inicio más tardío posibles.
En cada etapa de planificación, el primer nodo se extrae de la lista y se planifica para un procesador que permite el tiempo de inicio más temprano. El algoritmo MCP fue diseñado originalmente para un número acotado de procesadores.
La complejidad de este algoritmo es O ($v^2 \log v$), donde v es la cantidad de nodos del grafo.

3.2.4 Earliest Time Finish (ETF)

Este algoritmo es similar al MCP, la diferencia radica en que ETF utiliza prioridades estáticas para los nodos y supone un número de procesadores limitados. Sin embargo puede ocurrir que los nodos de mayor prioridad no sean agregados al schedule antes que los nodos con menor prioridad. Esto se debe a que en cada paso de ETF se calcula el tiempo de ejecución más temprano de aquellos nodos que se encuentran listos para su ejecución y luego selecciona de esos nodos el nodo con menor tiempo de comienzo de ejecución. Un nodo se encuentra listo si todos sus nodos padres han sido asignados al scheduler. El tiempo temprano de un nodo se obtiene examinando el tiempo de comienzo del nodo en todos los procesadores. Cuando dos nodos tienen el mismo tiempo de comienzo el algoritmo ETF elige al nodo con mayor prioridad estática.
La prioridad estática de un nodo puede ser calculada teniendo en cuenta el nivel del nodo como la distancia más costosa desde el mismo hasta la salida.
La complejidad de este algoritmo es $O(pv^2)$ donde p es el número de procesadores de la arquitectura y v la cantidad de nodos del grafo.

3.2.5 Heterogeneous - Earliest Finish - Time (HEFT)

El algoritmo HEFT es una aplicación de un algoritmo de scheduling para un número acotado de procesadores heterogéneos consistente en dos fases. La fase "Task Priorizing" en la cual se obtiene la prioridad de cada una de las tareas por medio de un valor $rank_u$ (t); este valor es la longitud del camino crítico desde la tarea t hasta una tarea de salida del scheduling, este atributo considera el tiempo de cómputo y comunicación que requiere la tarea; (existen algunas versiones que solo consideran el tiempo de cómputo). Una vez que se ha calculado el $rank_u$ de cada tarea se genera una lista ordenada de forma decreciente según el valor de $rank_u$. En los casos de empate elige en forma aleatoria el orden entre las tareas (existen otras alternativas para la elección del orden de las tareas si el $rank_u$ coincide pero requieren mucha más complejidad para el algoritmo).
La segunda fase es "Processor selection", en la cual para todas las tareas del scheduling el tiempo más temprano en que un procesador p_j está disponible para ejecutar una tarea, es cuando p_j ha terminado la ejecución de todas las tareas que tenía asignadas. Sin embargo, el algoritmo HEFT utiliza la política insertion-based la cual considera la posibilidad de insertar una tarea en el primer slot de tiempo libre que

el procesador tenga disponible, incluso antes de otras tareas ya asignadas al procesador. Para esto el slot de tiempo libre del procesador debe ser suficiente para ejecutar la tarea que se quiere insertar.
Este algoritmo es uno de los más utilizados, por su eficiencia en el scheduling y su baja complejidad, para realizar el scheduling de tareas a procesadores cuando la arquitectura es heterogénea. Esta razón hace que sea utilizado para ser como un "benchmark" de los algoritmos de scheduling.
El algoritmo HEFT tiene una complejidad de O(e*p) donde e representan los arcos y p la cantidad de procesadores. En grafos densos donde e se aproxima a v^2 (v números de tareas) la complejidad del algoritmo puede expresarse como $O(v^2*p)$.

3.2.6 Mobility Directed Algorithm (MD)

Selecciona un nodo en cada etapa para planificar en base a un atributo llamado *movilidad relativa*. La movilidad de un nodo está definida como la diferencia entre el tiempo de inicio más temprano del nodo y el tiempo de inicio más tardío. Al igual que la restricción ALAP mencionada anteriormente, se asigna el tiempo de inicio más temprano posible a cada nodo por medio de la restricción as-soon-as-possible (ASAP) y se lleva a cabo atravesando el grafo de tareas hacia abajo desde los nodos de entrada hacia los nodos de salida, llevando los nodos lo más arriba posible. La movilidad relativa se obtiene dividiendo la movilidad por el costo de cómputo del nodo. Esencialmente un nodo con movilidad 0 es un nodo en el camino crítico (CP). En cada etapa, el algoritmo MD planifica el nodo con la menor movilidad para el primer procesador que cuenta con un intervalo de tiempo suficiente para acomodarlo sin considerar la minimización del tiempo de inicio del nodo.
Después de que se ha planificado el nodo, se actualizan todas las movilidades relativas. La complejidad del algoritmo MD es O (v^3). A diferencia de MCP, el algoritmo MD determina las prioridades de los nodos dinámicamente. A pesar que MD puede identificar correctamente los nodos CP para planificar cada etapa, la selección de un intervalo de tiempo y un procesador adecuados no es precisa.

3.2.7 Dynamic Level Scheduling Algorithm (DLS)

Al igual que MD, el algoritmo DLS determina las prioridades de los nodos asignando dinámicamente un atributo denominado *nivel dinámico* (DL) a todos los nodos no planificados en cada etapa de la asignación. El DL se calcula utilizando dos cantidades. La primera cantidad es el nivel estático $SL(n_i)$ de un nodo n_i , que se define como la suma máxima de los costos de cómputo a lo largo del camino desde n_i hasta un nodo de salida. La segunda cantidad es el tiempo de inicio $ST(n_i,J)$ de n_i en el procesador J. El nivel dinámico $DL(n_i,J)$ para el par nodo-procesador (n_i,J) se define entonces como $SL(n_i) – ST(n_i,J)$. En cada etapa de la planificación, el algoritmo DLS calcula el DL para cada nodo listo en todos los procesadores.
Luego, se selecciona el par nodo-procesador que constituye el DL más grande entre todos los demás pares de manera que se planifique el nodo para el procesador. Este proceso se repite hasta que todos los nodos hayan sido planificados.
La complejidad de este algoritmo es $O(v^3 pf(p))$, donde p es el número de los procesadores dados, f (p) es la complejidad del algoritmo de ruteo de datos para calcular ST de un nodo en cada etapa, y v la cantidad de nodos del grafo.
Este algoritmo no asigna prioridades en base al camino crítico, sino que realiza un macheo exhaustivo de pares de los nodos en los procesadores en cada etapa para encontrar el nodo con mayor prioridad. La idea de este algoritmo es utilizar un parámetro compuesto DL para seleccionar un nodo con un nivel estático más elevado y menor tiempo de inicio para planificar. Sin embargo se debería tener en cuenta que el nivel del nodo seleccionado puede no ser el más alto y su tiempo de inicio puede no ser el más temprano entre todos los nodos listos. Esta es la sutil diferencia entre DLS y EFT.

3.2.8 Dominant Sequence Clustering Algorithm (DSC)

El algoritmo DSC se basa en un atributo denominado *secuencia dominante* que es esencialmente el camino crítico del grafo de tareas parcialmente planificado en cada etapa.
En cada etapa, DSC verifica si el nodo del camino crítico más alto (ubicado al principio) es un nodo listo. Si es así, lo planifica para un procesador que permite el tiempo de inicio mínimo. Tal tiempo de inicio mínimo puede ser alcanzado “replanificando” algunos de los nodos padres del nodo para el mismo procesador. Por otro lado, si el

nodo camino crítico más elevado no es un nodo listo, este algoritmo no lo selecciona para la planificación y en este caso el nodo seleccionado es el más elevado que se encuentran en el camino crítico de la planificación.
DSC planifica el nodo para un procesador que permite el tiempo de inicio mínimo del nodo siempre y cuando dicha selección de procesador no atrase el tiempo de inicio de un nodo camino crítico no planificado aún.
La planificación retrasada de los nodos camino crítico hacen que este algoritmo determine incrementalmente el próximo nodo más elevado.
Esta estrategia también lleva a la baja complejidad del algoritmo DSC, que es O((e+v) log v), donde v representa el número de nodos del grafo y e la cantidad de arcos del mismo.

3.2.9 Mapping Algorithm based on Task Dependencies (MATE)

Este algoritmo parte de una lista inicial donde las tareas están ordenadas de menor a mayor nivel $LN(T_i)$ (el nivel de una tarea $T_i$está dado por la cantidad mínima de arcos desde un nodo inicial del grafo hasta T_i). A partir de dicha lista, las tareas se tratan y se asignan por niveles. Para un nivel dado, las tareas se ordenan de forma decreciente según el valor de máxima ganancia. Dicho valor indica la ganancia en tiempo que puede ser obtenida cuando una tarea T_i se ubica en el mismo procesador que una tarea adyacente T_j previamente asignada, frente a la ganancia obtenida si se ejecutaran en diferentes procesadores. La ganancia es calculada teniendo en cuenta el grado de concurrencia de las tareas si son ejecutadas en el mismo procesador, y el costo de la comunicación si se ejecutan en diferentes procesadores. Asume procesadores homogéneos [18] [37].

3.3 Clasificación de los algoritmos anteriores

Existen numerosas formas para la clasificación de los algoritmos de asignación estática. En esta sección mencionaremos tres de ellas.
Una de las formas de clasificar los algoritmos de scheduling estáticos consiste en dividirlos en dos grupos principales: *basados en heurísticas* y *basados en búsquedas aleatorias guiadas*. Figura 3.2

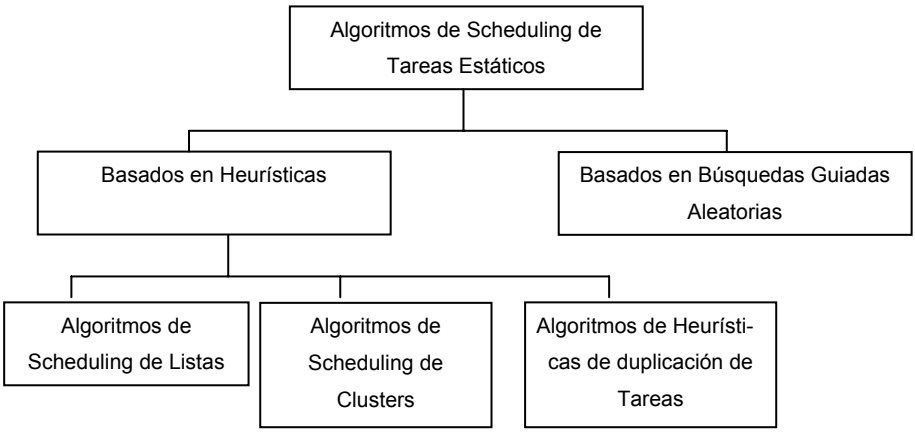


Figura 3.2 – Clasificación de Algoritmos de Scheduling

En este trabajo se tiene en cuenta el primer grupo de algoritmos, los cuales pueden ser divididos a su vez, en tres grupos: heurísticas de listas de scheduling, heurísticas de clustering y heurísticas de duplicación de tareas.

- ✓ Heurísticas de listas de scheduling: mantienen una lista de todas las tareas de un grafo dado de acuerdo a sus prioridades. Estos algoritmos tienen dos etapas: la de "selección de tarea", en la que se elige la tarea de mayor prioridad lista para ejecutar, y la fase "selección de procesador", en la cual se elige el procesador que minimice el costo de una función predefinida (por ejemplo tiempo de inicio de ejecución). Algunos ejemplos de este tipo de algoritmos son: Modified Critical Path (MCP), Dynamic Level Scheduling (DLS). La mayoría de estos algoritmos se utilizan en arquitecturas de procesadores homogéneos totalmente conectados. Además estos algoritmos son generalmente más prácticos y obtienen una mejor performance en los resultados minimizando el tiempo del scheduling.
- ✓ Heurística de clusters: un algoritmo en este grupo mapea las tareas de un grafo dado en un número ilimitado de clusters. En cada paso, la tarea seleccionada puede ser cualquiera y no tiene que estar necesariamente lista para su ejecución. Cada iteración refina el agrupamiento previo mezclando algunos clusters. Si dos tareas son asignadas al mismo cluster, entonces ambas serán ejecutadas en el mismo procesador. La heurística de clustering requiere pasos adicionales para gene-

rar el scheduling final: una etapa de "mezcla de clusters" tratando de mezclar los clusters para que el número sea igual al número de procesadores, una etapa de "mapping de clusters" para mapear los cluster a procesadores disponibles, y un paso de "ordenamiento de tarea' el cual sirve para ordenar el mapeo de las tareas a procesadores. Algunos ejemplos de este tipo de algoritmos son: DSC y MD.

- ✓ Heurísticas de duplicación de tareas: la idea de los algoritmos de este tipo es mapear tareas en forma redundante en distintos procesadores para reducir el overhead de comunicación entre procesos. Los algoritmos que utilizan esta técnica difieren en la estrategia de duplicación de las tareas. En este grupo generalmente se ejecutan un número ilimitado de procesadores homogéneos y los algoritmos tienen una mayor complejidad que los algoritmos de otros grupos.

Otra forma consistiría en clasificarlos en función del *criterio seguido para asignar prioridades* a las tareas, y el criterio para seleccionar el procesador al cual se asigna cada tarea. Con esta clasificación se obtienen los siguientes cuatro grupos:

- Camino crítico: los algoritmos basados en el camino crítico asignan mayor prioridad, en el proceso de scheduling a los nodos que forman parte de dicho camino. Para construir el camino crítico se elige el conjunto de nodos y arcos que forman un camino desde un nodo inicial hasta un nodo final cuya suma de costos de computación y comunicación es el máximo. Ejemplos de este caso son: MD y DSC.
- Lista estática: cuando la lista de prioridades se construye antes de iniciar la asignación, y se mantiene igual a lo largo de todo el proceso, se tratará de algoritmos de lista estática (como el ETF y EZ). El principal problema de estos algoritmos es que la asignación estática de prioridades no siempre ordena correctamente los nodos, generando schedules muy ineficientes si no puede asignar prioridades precisas.
- Lista dinámica: en este caso la lista se va actualizando a medida que transcurre el proceso de schedulling. Algoritmos representativos de esta categoría son DSC, MD y MCP.
- Greedy: se denominan algoritmos greedy aquellos que en el proceso de asignación tratan de minimizar el tiempo de inicio de las tareas. Esta es una estrategia seguida por varias algoritmos de scheduling como el ETF, MCP, MD, y EZ.

Por último otra manera de dividir a los algoritmos en dos grandes categorías dependiendo de la *disponibilidad que se considere del número de procesadores*.
Con esta clasificación se tiene dos tipos: BNP (Bounded Number of Processors), que consideran un número limitado de procesadores y basan su asignación en este número concreto, y UNC (Unbounded Number of Clusters) que trabajan con un número de procesadores ilimitado y van agrupando las tareas en clusters en función de sus dependencias, sin tener en cuenta la arquitectura.
Para realizar la comparación de los algoritmos BNP y UNC de forma equitativa se puede asumir que para los algoritmos BNP se dispone de un número de procesadores muy elevado que haga que a efectos del mapping se vea como ilimitado, ya que es la premisa con la que trabajan los algoritmos UNC.
En la tabla 1 se presenta una clasificación de los algoritmos de scheduling más representativos, en función de sus características.

Algoritmo	Basados en Heurísticas	Basado en Prioridades	Nro Proc.
ETF	Scheduling de Listas	Lista Estática – Greedy	BNP
CP	Scheduling de Listas	Camino Crítico	BNP
MCP	Scheduling de Listas	Lista Dinámica - Greedy	BNP
MATE	Scheduling de Listas	Lista Dinámica - Greedy	BNC
DSC	Scheduling de Cluster	Camino Crítico - Lista Dinámica	UNC
DLS	Scheduling de Listas	Camino Crítico - Lista Dinámica	BNP
MD	Scheduling de Cluster	Camino Crí tico - Lista Dinámica – Greedy	UNC
EZ	Scheduling de Listas	Lista Estática - Greedy	UNC
HEFT	Scheduling de Listas	Lista Estática - Greedy	BNP

Tabla 1 – Clasificación de los Algoritmos de Scheduling más representativos.

3.4 Limitaciones de los algoritmos anteriores

Los algoritmos presentados anteriormente muestran un conjunto de problemas para representar ciertas situaciones en aplicaciones paralelas.
Existen casos en los que la planificación EZ no se identifica de manera correcta el nodo más importante en cada etapa. Esto genera que en muchos casos las planificaciones sean ineficientes [31] [32].
El algoritmo MCP asigna prioridades mayores a los nodos que tienen un menor tiempo de inicio más tardío posible. Sin embargo, este algoritmo no necesariamente planifica nodos en el CP en primer lugar. Además existen situaciones en las que no asigna prioridades de nodos

de manera precisa, a pesar de que toma en cuenta la comunicación entre nodos para calcular las prioridades.
La principal deficiencia del algoritmo EFT es que puede no ser capaz de reducir la longitud parcial de la planificación en cada etapa de la misma. Esto se debe a que un nodo que tiene el menor valor del tiempo de inicio más temprano puede no necesariamente pertenecer al CP. Esto provoca que los intervalos de tiempo más tempranos en el procesador pueden ser ocupados, y por lo tanto, los nodos CP pueden no ser planificados de manera oportuna.
Además, una característica de la mayoría de los algoritmos considerados previamente (excepto DLS y HLFT) es que no consideran que la aplicación pueda ser ejecutada en una arquitectura de cluster o multiclusters heterogéneos. Esta consideración es bastante importante ya que la mayoría de las arquitecturas de clusters y multicluster son heterogéneas.
El algoritmo HEFT tiene como ventaja que si considera que la aplicación puede ser corrida en un cluster heterogéneo, sin embargo como el resto de los algoritmos analizados (salvo el MATE) no permite representar a las tareas de la aplicación como un conjunto de subtareas. Esto ocasiona que la representación de la aplicación sea de menor grado de abstracción generando un modelo de "fine grane".
Por último, se puede mencionar que todos los algoritmos (excepto MATE) se aplican sobre el modelo TPG, el cual presenta los problemas descriptos en el capítulo II.

Resumen del capítulo

En este capítulo se mencionaron el conjunto de algoritmos de mapping existentes para aplicaciones paralelas ejecutadas sobre arquitecturas homogéneas. De cada uno se desarrollaron sus características junto a sus ventajas y desventajas.

CAPÍTULO IV

Modelo TTIGHA

4.1 Introducción

En el capítulo II, se han descripto un conjunto de modelos los cuales intentan representar las características más salientes de las aplicaciones paralelas. En dicho capítulo también se mencionaron algunos de los problemas que presentaban cada uno de ellos. Por ejemplo: el modelo TIG no refleja el orden en el cual deben ejecutarse las tareas; en el caso del TPG si queda explícito en el grafo el orden parcial de ejecución de las tareas, pero dicho grafo está formado por muchas más tareas de las que en realidad tiene el programa inicial; y por último en TTIG [37] (combinación de los dos anteriores) se consigue solucionar esos problemas, pero este modelo continua siendo aplicable solo en arquitecturas homogéneas.
En este capítulo se propone un nuevo modelo llamado TTIGHA (Temporal Task Interaction Graph in Heterogeneous Architecture) que permite representar aplicaciones paralelas en forma más realista que los mencionados en el capítulo II, considerando la heterogeneidad de la arquitectura, tanto en lo que respecta a los procesadores como a las comunicaciones.

4.2 Definición del modelo TTIGHA

El modelo TTIGHA se basa en la construcción de un grafo G (V,E,B) donde:

- V, es el conjunto de nodos donde cada uno representa una tarea T_i del programa paralelo.
- E, es el conjunto de aristas que representan la comunicación entre los nodos del grafo.
- B es un conjunto de identificadores que indica cuales son los nodos (o tareas) básicos (o iniciales) en la aplicación, es decir aquellos que pueden comenzar su ejecución sin necesidad de interacción con otro nodo.

Detalle de los parámetros del modelo

En el primer parámetro del grafo (V) cada nodo representa una tarea T_i del programa paralelo. Dada la posibilidad de contar con una arquitectura heterogénea se deben tener en cuenta los tiempos de cómputo en cada uno de los tipos de procesadores que la componen. Es decir, el nodo *i* ($V_i \in V$) almacena el tiempo de cómputo de la tarea T_i en cada uno de los tipos diferentes de procesador. Por lo tanto,

V_i (p) = tiempo de ejecución de la tarea T_i en el tipo de procesador p.

En el segundo parámetro del grafo (E), las aristas representan las comunicaciones que existen entre cada par de tareas. En este conjunto una arista A entre dos tareas T_i y T_j contiene el volumen de comunicación en bytes y el "grado de concurrencia" entre ambas.

Es importante notar que dado la heterogeneidad de la red de interconexión solo se mantiene el volumen de comunicación entre dos subtareas, en lugar del tiempo requerido para la misma.

El "grado de concurrencia" es una matriz h_{ij}, donde $h_{ij}(s,d)$ representa el grado de concurrencia entre las tarea T_i en un procesador de tipo s y la tarea T_j en un procesador de tipo d. Este índice ("grado de concurrencia") está normalizado entre 0 y 1. Para dos tareas T_i y T_j que se comunican de T_i a T_j, el grado de concurrencia se define como el máximo porcentaje del tiempo de cómputo de T_j que puede ser realizado en paralelo con T_i, teniendo en cuenta sus dependencias mutuas provocadas por las comunicaciones existentes entre ambas tareas, y sin contemplar el costo de comunicación asociado a las mismas (esto genera un valor independiente a los datos a transmitir).

Este nuevo índice (grado de concurrencia) hace que el modelo TTIGHA combine los dos modelos explicados anteriormente (TIG y TPG). En TPG todas las tareas adyacentes tienen una relación de precedencia estricta, de tal forma que una tarea no puede empezar su ejecución hasta que sus predecesoras hayan terminado y le hayan transmitido sus datos. Esta situación refleja un grado de concurrencia igual a 0 en el modelo TTIGHA. Por otro lado, en el modelo TIG no se explicita en el grafo ninguna información temporal entre tareas, y en general se asume que todas son completamente concurrentes con sus adyacentes. Esta situación queda reflejada con el grado de concurrencia igual a 1 en el modelo TTIGHA.

4.3 Generación del modelo TTIGHA

Para generar el modelo TTIGHA de una aplicación es indispensable poder determinar o estimar el valor de cada uno de los parámetros que componen el modelo. Esta estimación puede realizarse analizando el código escrito por el programador, de manera manual (considerando los tiempos requeridos para cada instrucción) o por medio de la utilización de programas creados para este fin [17]; o realizar la estimación de los parámetros analizando los archivos de trazas generados en varias ejecuciones del algoritmo.
En esta tesis se utilizó la técnica de obtener la estimación de los parámetros del modelo por medio del análisis del archivo de trazas. Por lo tanto los pasos aplicados para obtener la estimación de los parámetros del modelo son:

1. Ejecución de la aplicación para generar el archivo de trazas en cada uno de los tipos de procesadores.
2. Análisis del archivo de trazas para la generación de datos intermedios.
3. Generación del modelo TTIGHA a partir de los datos obtenidos en el paso anterior.

Ejecución de la aplicación para generar el archivo de trazas
En este paso se ejecuta la aplicación paralela. A partir de esta ejecución se genera un archivo de trazas por cada uno de los tipos de procesadores en donde las tareas podrían ejecutarse. Cada archivo contiene los tiempos de comienzo y finalización de cada una de las subtareas (bloques de cómputo indivisibles) de la aplicación.
Análisis del archivo de trazas para la generación de datos intermedios
Para generar el grafo G que representa a la aplicación se deben obtener los siguientes valores:

- Correspondencia de subtareas a tareas (M): es un vector donde M(i) indica de que tarea es miembro la subtarea i.
- Tiempos de subtareas (Ts): es un vector que almacena los datos de cada subtarea, donde Ts(i) contiene un vector que guarda cada uno de los tiempos de cómputo de la subtarea i en los diferentes tipos de procesadores. El tiempo de cómputo de una subtarea i en un procesador de tipo p (Ts(i,p)), se calcula como la diferencia entre el tiempo de fin y de inicio de la subtarea almacenado en el archivo de trazas correspondiente al procesador p.
- Volumen de comunicación entre subtareas (Cs): es una matriz donde cada elemento Cs(i,j) representa el volumen de comunicación de la subtarea ST_i a la subtarea ST_j, el cual puede ser estimado a partir del código de la aplicación.

Generación del modelo TTIGHA a partir de los datos obtenidos en el paso anterior

Luego de que se han obtenidos los valores anteriores se procede a construir el grafo G que representará la aplicación en el modelo TTIGHA. Esta construcción se realiza en tres etapas, la primera es la creación del conjunto de nodos V, la segunda es la formación del conjunto de aristas E, y en la tercera se determina el conjunto de nodos iniciales B:

- Creación de (V): en esta etapa se calcula el tiempo de ejecutar cada tarea en cada tipo de procesador. Para esto, para cada tarea se obtienen las subtareas que la forman (dato obtenido a partir de M), y a partir de esta información se calcula el tiempo de la tarea en cada tipo de procesador como la suma de los tiempos respectivos de cada una de sus subtareas (dato que se encuentra en Ts).
- Creación de (E): esta etapa se divide en dos subetapas. En la primera subetapa se obtiene para cada arista del conjunto E el volumen total de comunicación entre los nodos que conecta dicha arista. El valor E_{ij}, es decir, el volumen de comunicación entre la tarea Ti y la tarea Tj, se calcula como la suma de los volúmenes de comunicación de aquellas subtareas de Ti que le envían información a alguna subtarea perteneciente a Tj. La segunda subetapa calcula a partir de los tiempos de ejecución generados en la creación de V, el valor del grado de concurrencia para cada arista. Este valor $h_{ij}(s,d)$ es calculado como el tiempo en que las tareas T_i y T_j en los tipos de procesadores s y d respectivamente pueden ejecutarse en paralelo sobre el tiempo de ejecución de T_j en el tipo de máquina d.
- Creación de (B): en esta etapa se determina cuales nodos del conjunto V pueden comenzar su ejecución desde el instante 0 (es decir que no necesitan una previa interacción con otro nodo), llamado de aquí en adelante nodo inicial. Para construir este conjunto se evalúa cada tarea T_i, y si es un nodo inicial se agrega el identificador i al conjunto B. Para que una tarea T sea un nodo inicial, la primer subtarea de T según el orden de precedencia (ST_p) no debe participar en ninguna comunicación como destino de la misma. Es decir que $Cs(x, p) = 0$ para cualquier valor de x.

4.3.1 Ejemplo de la generación del modelo TTIGHA

Para la mejor comprensión del proceso de generación del grafo G, se desarrolla un ejemplo, el cual mostrará el funcionamiento en cada paso de la creación del grafo G. Para el ejemplo la aplicación paralela está formada por 8 tareas (T_0...T_7) compuestas por diferentes cantidades de subtareas como se muestra en la figura 4.1 y se cuenta con dos tipos diferentes de procesador.

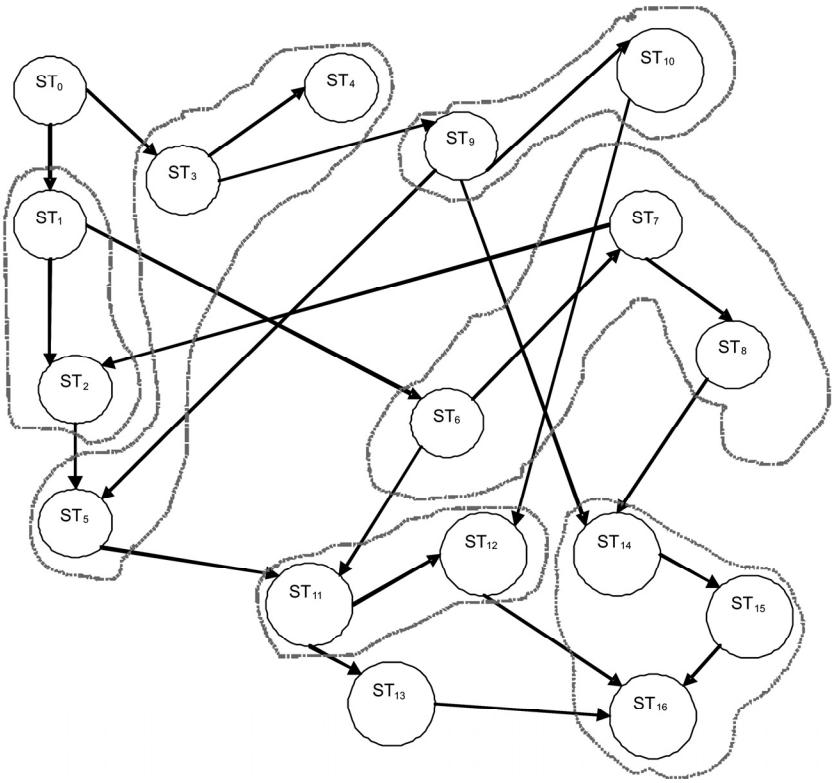


Figura 4.1 – Subtareas que componen la aplicación

La tabla 4.1 muestra la composición de cada una de las tareas de la aplicación (T_0...T_7), junto a los tiempos de ejecución en cada tipo de procesador y los volúmenes de comunicación de las subtareas que las componen.

Tarea	Subtarea (M)	Tiempo Ejecución en el tipo de procesador 0 (Ts)	Tiempo Ejecución en el tipo de procesador 1 (Ts)
T_0	ST_0	5	7
T_1	ST_1	20	35
	ST_2	100	145
T_2	ST_3	25	40
	ST_4	15	25
	ST_5	10	15
T_3	ST_6	15	20
	ST_7	100	115
	ST_8	25	40
T_4	ST_9	35	60
	ST_{10}	20	35
T_5	ST_{11}	10	15
	ST_{12}	15	20
T_6	ST_{13}	120	160
T_7	ST_{14}	80	85
	ST_{15}	20	25
	ST_{16}	10	12

Tabla 4.1 – Composición de las tareas de la aplicación y tiempos de las mismas

La tabla 4.2 indica el volumen de comunicación (en bytes) involucrado para cada par de subtareas.

	ST_0	ST_1	ST_2	ST_3	ST_4	ST_5	ST_6	ST_7	ST_8	ST_9	ST_{10}	ST_{11}	ST_{12}	ST_{13}	ST_{14}	ST_{15}	ST_{16}
ST_0	0	10.000	0	5.000	0	0	0	0	0	0	0	0	0	0	0	0	0
ST_1	0	0	0	0	0	0	1.500	0	0	0	0	0	0	0	0	0	0
ST_2	0	0	0	0	0	8.000	0	0	0	0	0	0	0	0	0	0	0
ST_3	0	0	0	0	0	0	0	0	0	1.000	0	0	0	0	0	0	0
ST_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ST_5	0	0	0	0	0	0	0	0	0	0	0	6.700	0	0	0	0	0
ST_6	0	0	0	0	0	0	0	0	0	0	0	1.500	0	0	0	0	0
ST_7	0	0	1.800	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ST_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.100	0
ST_9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0	0
ST_{10}	0	0	0	0	0	0	0	0	0	0	0	0	2.000	0	0	0	0
ST_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0
ST_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
ST_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2500
ST_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ST_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ST_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 4.2 – Comunicación entre subtareas

Los datos presentados en las tablas anteriores fueron obtenidos después de analizar el archivo de trazas generado para cada tipo de procesador por la ejecución de la aplicación sobre un representante de cada uno de ellos. En la tabla 4.1 se encuentra reflejada la información de M (membrecía de subtarea a tareas) y Ts (tiempos de las subtareas); a su vez en la tabla 4.2 se muestra la matriz de comunicación entre subtareas Cs.
Con los datos generados anteriormente comienza el paso 3 en el cual se realiza el armado del grafo de la aplicación. Para esto se realizan las siguientes acciones:

a) calcular los tiempos de cada una de las tareas en cada uno de los tipos de procesador.
b) Calcular el grado de concurrencia entre los distintos pares de tareas.
c) Calcular el volumen de comunicación entre las diferentes tareas.
d) Determinar el conjunto de nodos básicos o iniciales.

- Calcular los tiempos de cada tarea T_i: donde el tiempo de T_i es la suma de los tiempos de sus subtareas. La tabla 4.3 muestra dichos tiempos.

Tarea	Tipo de procesador 0	Tipo de procesador 1
T_0	5	7
T_1	120	180
T_2	50	80
T_3	140	175
T_4	55	95
T_5	25	35
T_6	120	122
T_7	110	160

Tabla 4.3 – Tiempos de cada tarea en cada procesador

- Calcular el grado de concurrencia para cada par de tareas adyacentes de acuerdo al tipo de procesador en que se encuentren. Las tablas 4.4,..,4.17 muestran esta información.

$T_0 \setminus T_1$	Tipo 0	Tipo 1
Tipo 0	0	0
Tipo 1	0	0

Tabla 4.4 – Grado de concurrencia entre T_0/T_1

$T_0 \setminus T_2$	Tipo 0	Tipo 1
Tipo 0	0	0
Tipo 1	0	0

Tabla 4.5 – Grado de concurrencia entre T_0/T_2

$T_1 \setminus T_2$	Tipo 0	Tipo 1
Tipo 0	0.8	0.81
Tipo 1	0.8	0.81

Tabla 4.6 – Grado de concurrencia entre T_1/T_2

$T_1 \setminus T_3$	Tipo 0	Tipo 1
Tipo 0	0.18	0.23
Tipo 1	0.18	0.23

Tabla 4.7 – Grado de concurrencia entre T_1/T_3

$T_2 \setminus T_4$	Tipo 0	Tipo 1
Tipo 0	0.45	0.26
Tipo 1	0.73	0.42

Tabla 4.8 – Grado de concurrencia entre T_2/T_4

$T_2 \setminus T_5$	Tipo 0	Tipo 1
Tipo 0	0	0
Tipo 1	0	0

Tabla 4.9 – Grado de concurrencia entre T_2/T_5

$T_3 \setminus T_1$	Tipo 0	Tipo 1
Tipo 0	0.21	0.14
Tipo 1	0.33	0.22

Tabla 4.10 – Grado de concurrencia entre T_3/T_1

$T_3 \setminus T_5$	Tipo 0	Tipo 1
Tipo 0	1	1
Tipo 1	1	1

Tabla 4.11 – Grado de concurrencia entre T_3/T_5

$T_3 \setminus T_7$	Tipo 0	Tipo 1
Tipo 0	0.73	0.70
Tipo 1	0.73	0.70

Tabla 4.12 – Grado de concurrencia entre T_3/T_7

$T_4 \setminus T_5$	Tipo 0	Tipo 1
Tipo 0	0.40	0.43
Tipo 1	0.40	0.43

Tabla 4.13 – Grado de concurrencia entre T_4/T_5

$T_4 \setminus T_7$	Tipo 0	Tipo 1
Tipo 0	0.18	0.16
Tipo 1	0.32	0.29

Tabla 4.14 – Grado de concurrencia entre T_4/T_7

$T_5 \setminus T_6$	Tipo 0	Tipo 1
Tipo 0	0.12	0.09
Tipo 1	0.17	0.12

Tabla 4.15 – Grado de concurrencia entre T_5/T_6

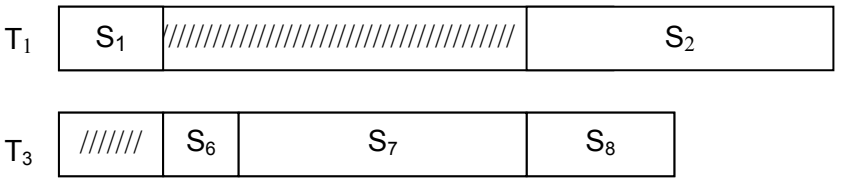
$T_5 \setminus T_7$	Tipo 0	Tipo 1
Tipo 0	0.23	0.20
Tipo 1	0.32	0.23

Tabla 4.16 – Grado de concurrencia entre T_5/T_7

$T_6 \setminus T_7$	Tipo 0	Tipo 1
Tipo 0	0.91	0.90
Tipo 1	0.91	0.90

Tabla 4.17 – Grado de concurrencia entre T_6/T_7

A modo de ejemplo se explica cómo se obtienen los valores para la tabla 4.7, la cual muestra el grado de concurrencia entre las tareas T_1 y T_3, donde T_1 es la que inicia la comunicación y T_3 recibe los datos.
Según el grafo de tareas mostrado en la figura 4.1, la ejecución de dichas tareas puede verse en el siguiente esquema:



El esquema refleja que T_3 no puede comenzar su ejecución hasta que la subtarea S_1 haya finalizado, dado que requiere que esta le envíe información. A partir de ese momento la subtarea S_6 y S_7 pueden ejecutarse. Recién en el momento en que estas terminan ambas tareas (T_1 y T_3) pueden ejecutarse en forma simultánea al correr S_2 y S_8 respectivamente.
Teniendo en cuenta que el tiempo requerido por S_2 es siempre mayor al requerido por S_8 (sin importar la máquina en que esté cada tarea), se puede asegurar que el tiempo en que ambas tareas (T_1 y T_3) se pueden ejecutar en forma simultánea es igual al tiempo requerido por S_8.
De acuerdo a la definición de grado de concurrencia, en este caso se calcularía como el tiempo de ejecución de S_8 sobre el tiempo de la tarea T_3 (ambos tiempos varían según el tipo de máquina en la que se ubique T_3). De esta manera el grado de concurrencia entre T_1 y T_3 es:

- Al considerar T_1 y T_3 en procesadores de tipo 0, el grado de concurrencia es 25 / 140.
- Al considerar T_1 y T_3 en procesadores de tipo 1 y 0 respectivamente, el grado de concurrencia es 25 / 140.
- Al considerar T_1 y T_3 en procesadores de tipo 0 y 1 respectivamente, el grado de concurrencia es 40 / 175.
- Al considerar T_1 y T_3 en procesadores de tipo 1, el grado de concurrencia es 40 / 175.

- Calcular el volumen de comunicación entre cada par de tareas. La tabla 4.18 muestra esta información.

	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7
T_0	0	10.000	5.000	0	0	0	0	0
T_1	0	0	8.000	1.500	0	0	0	0
T_2	0	0	0	0	1.000	6.700	0	0
T_3	0	1.800	0	0	0	1.500	0	1.100
T_4	0	0	0	0	0	2.000	0	500
T_5	0	0	0	0	0	0	100	2.000
T_6	0	0	0	0	0	0	0	2.500
T_7	0	0	0	0	0	0	0	0

Tabla 4.18 – Volumen de comunicación entre las tareas

- Determinar el conjunto de nodos básicos o iniciales. La tabla 4.19 indica para cada tarea la primer subtarea que la compone y si es o no nodo inicial.

	Subtarea	Comunicación	Nodo Inicial
T_0	S_0	--	Si
T_1	S_1	Comunicación de S_0 a S_1	No
T_2	S_3	Comunicación de S_0 a S_2	No
T_3	S_6	Comunicación de S_1 a S_6	No
T_4	S_9	Comunicación de S_3 a S_9	No
T_5	S_{11}	Comunicación de S_5 a S_{11} Comunicación de S_6 a S_{11}	No
T_6	S_{13}	Comunicación de S_{11} a S_{13}	No
T_7	S_{14}	Comunicación de S_9 a S_{14}	No

Tabla 4.19– Evaluación de nodos iniciales

A partir de los datos mencionados anteriormente se genera el grafo G que representa la aplicación.

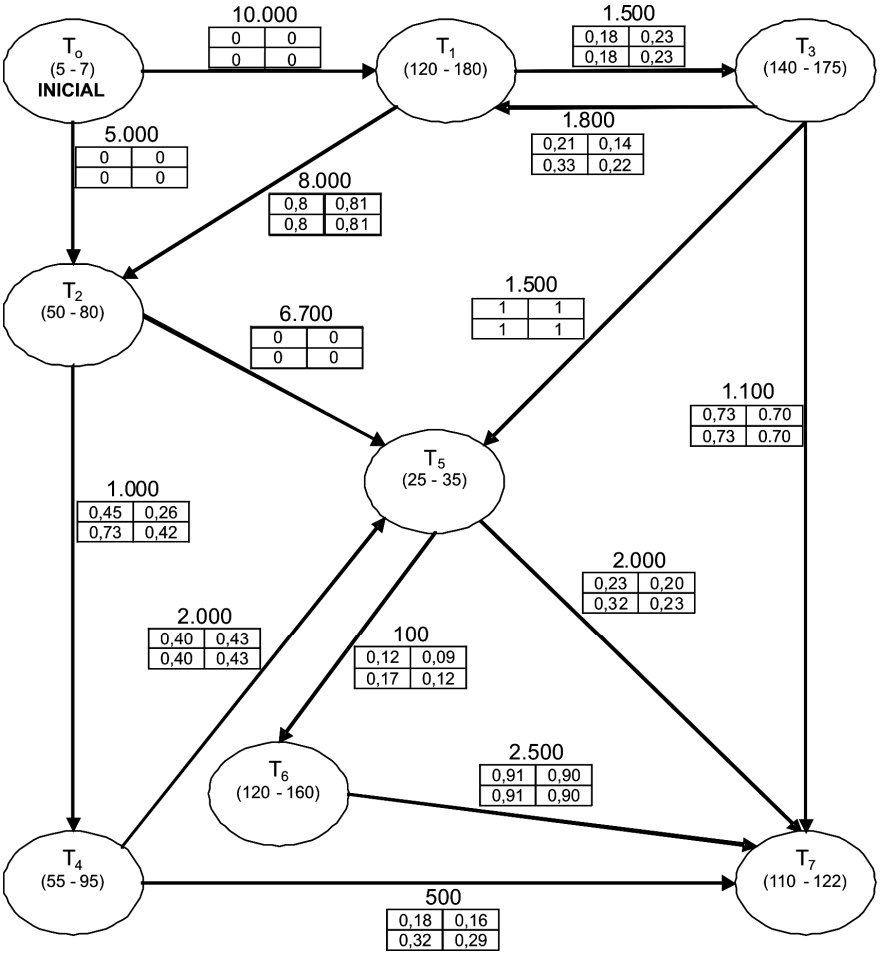


Figura 4.2 – Grafo TTIGHA de la aplicación.

Resumen del capítulo

En este capítulo se desarrolló la explicación del nuevo modelo TTIGHA el cual permite representar aplicaciones paralelas considerando que las mismas pueden ejecutarse sobre arquitecturas heterogéneas, esta heterogeneidad se da en cuanto a los procesadores y a la red de comunicación entre los mismos.
Una vez representada la aplicación mediante el grafo generado por el modelo TTIGHA los algoritmos de mapping que se describen en el capítulo V lo utilizan para realizar la asignación de las diferentes tareas que componen la aplicación en cada uno de los procesadores de la arquitectura, teniendo en cuenta las características de la misma.
Además en este capítulo se presenta un ejemplo que muestra como a partir de una aplicación se obtienen los valores de los parámetros que forman parte del modelo.

CAPÍTULO V

Algoritmos de Mapping MATEHA y METHAIB

5.1 Introducción

En el capítulo anterior se ha desarrollado y estudiado el modelo TTIGHA, el cual considera el problema del mapping en arquitecturas heterogéneas. A su vez, en el mismo capítulo se han mencionado y analizado las ventajas de este nuevo modelo frente a modelos existentes (TPG, TIG, TTIG), para la toma de decisiones eficientes.
En el capítulo III, se mencionaron un conjunto de algoritmos de mapping existentes aplicados a grafos. De los mismos se realizó la descripción junto al análisis de sus ventajas y desventajas. El problema común de los algoritmos descriptos es que no pueden ser aplicados al modelo TTIGHA, ya que o bien no consideran la heterogeneidad de la arquitectura, o no permiten el agrupamiento de subtareas en tareas dando un mayor grado de abstracción en la representación de las aplicaciones. De todos estos algoritmos el único que permite el agrupamiento de subtareas en tareas es MATE, pero solo puede ser aplicado a arquitecturas homogéneas; por otro lado el algoritmo HEFT no permite el agrupamiento de subtareas en tareas, pero considera la posible heterogeneidad de la arquitectura, razón por la cual es uno de los algoritmos más utilizados.
En este capítulo se describen dos algoritmos de mapping desarrollados en esta tesis MATEHA (Mapping Algorithm based on Task dEpendencies in Heterogeneous Architectures) y MATEHAIB (Mapping Algorithm based on Task dEpendencies in Heterogeneous Architectures using the policity Inserted Based), los cuales se aplican para el modelo TTIGHA. Estos algoritmos permiten determinar la asignación de tareas a los procesadores de la arquitectura a utilizar, buscando minimizar los tiempos de ejecución de la aplicación en dicha arquitectura.
Ambos algoritmos consideran una arquitectura con un número acotado de procesadores heterogéneos en cuanto a su potencia de cálculo. Con respecto a la red de interconexión el algoritmo también considera que pueda ser heterogénea en cuanto al ancho de banda y a la velocidad de transmisión.

5.2 Algoritmo MATEHA

La estrategia de MATEHA consiste en determinar, para cada una de las tareas del grafo G formado por el modelo TTIGHA, a qué procesador debe ser asignada para lograr el mayor rendimiento de la aplicación en la arquitectura utilizada.
Dicha asignación usa los valores generados en la construcción del grafo: tiempo de cómputo de una tarea en cada tipo de procesador, volumen de comunicación con sus adyacentes y por último el grado de concurrencia entre tareas. Este último valor es útil para tomar la decisión de asignar al mismo procesador aquellas tareas con menor grado de concurrencia, o asignar a procesadores diferentes aquellas tareas que pueden ejecutarse en forma paralela (el valor de su grado de concurrencia es alto).

5.2.1 Descripción del algoritmo

El algoritmo de mapping extrae los valores mencionados anteriormente del modelo TTIGHA, en los que se fundamenta la heurística de asignación del mismo.
En primer lugar, para cada nodo del grafo del modelo TTIGHA se define el *nivel,* estevalor será utilizado para realizar la asignación de las tareas del grafo con cierta prioridad. Luego, para cada tarea se calcula el procesador que genera la máxima ganancia.
En resumen, el algoritmo de mapping puede dividirse en dos pasos: calcular el nivel de cada nodo del grafo y asignar las tareas a los procesadores.
A continuación se describen cada uno de estos dos pasos.

5.2.1.1 Cálculo del nivel de un nodo

Dado un grafo G, el nivel de un nodo LN(T) se define como el mínimo número de arcos que hay que atravesar desde un nodo inicial T_{in} hasta T. La fórmula 5.1 expresa la definición antes mencionada:

$$LN\ (T) = \min_{T\,in\ \in\ B}\ d(T_{in}, T) \qquad \text{Fórmula 5.1}$$

donde $d(T_{in},T)$ corresponde al mínimo número de arcos que hay que recorrer desde T_{in} a T.

A modo de ejemplo, si se considera el grafo de la figura 5.1, el nivel de la tarea T_0 es 0 ya que es una tarea o nodo inicial, el nivel de las tareas T_1 y T_2 es 1, el de las tareas T_3, T_4 y T_5 es 2, y finalmente el de las tareas T_6 y T_7 es 3.

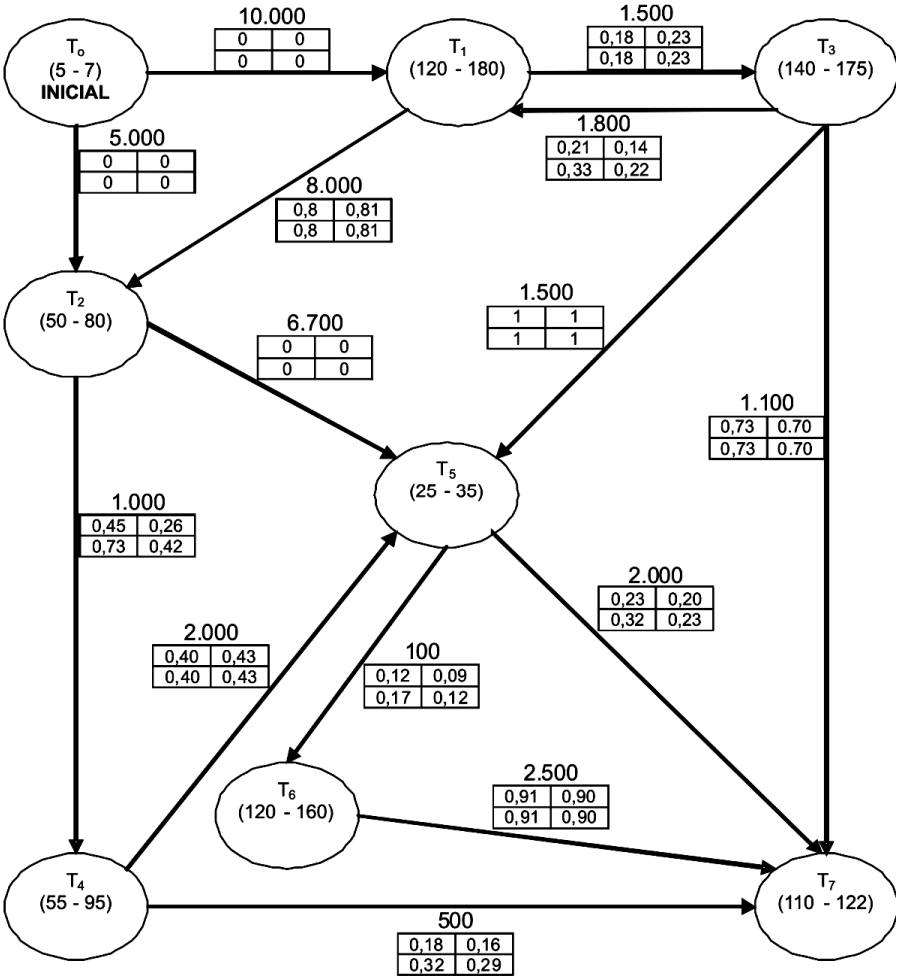


Figura 5.1- Grafo TTIGHA de la aplicación

El algoritmo de cálculo de nivel requiere tantas iteraciones como tareas componen el grafo, por lo tanto la complejidad computacional del mismo será O(n), donde n es el número de nodos.

5.2.1.2 Asignación de tareas a procesadores

Como se definió en el capítulo anterior, el modelo TTIGHA mantiene para cada par de tareas adyacentes T_i y T_j como mínimo tres valores: el tiempo de ejecución estimado de cada una de ellas en los diferentes tipos de procesadores de la arquitectura, el volumen de comunicación y el grado de concurrencia entre ambas tareas (este último valor depende del tipo de procesador en donde se encuentran T_i y T_j).
En esta etapa de asignación se utiliza el conjunto de tareas ordenadas de manera creciente según el nivel de las mismas. La etapa de asignación se realiza por niveles, es decir, primero se asignan todas las tareas de nivel 0, una vez que las mismas han sido asignadas se prosigue con las de nivel 1 y así sucesivamente para todos los niveles del grafo.
Dado un nivel n, para realizar la asignación se elige aquella tarea aún no asignada perteneciente al nivel n que genera la máxima ganancia al asignar dicha tarea a un procesador. Luego se elige el procesador p que maximice esa ganancia, es decir que si la ganancia es mayor a 0 significa que es conveniente ejecutar T_i en el mismo procesador que la tarea adyacente T_a que produjo esa ganancia; en caso contrario se elige el procesador p que minimice el tiempo de ejecución acumulado en p más el tiempo de ejecutar T_i.
La ganancia de una tarea T_i se calcula de la siguiente manera: para cada una de sus tareas adyacentes ya asignadas a un procesador T_a ($T_a \in$ al conjunto de adyacentes asignadas de T_i) se calcula la diferencia entre el costo de ejecutar T_i en un procesador diferente al que ya ha sido asignado T_a (tiempo separado) y el costo de ejecutar T_i en el mismo procesador al que ya ha sido asignado T_a (tiempo junto). Siendo la ganancia de la tarea T_i el máximo entre la mayor de esas diferencias y 0. En el caso particular en que la tarea no tenga tareas adyacentes asignadas, la ganancia toma valor 0.
El "tiempo junto" de $T_{junto}(T_i,T_a)$ se obtiene como la suma del tiempo acumulado de T_a más el tiempo de ejecutar T_i en el mismo procesador al que está asignado T_a. El tiempo acumulado de T_a es el tiempo de T_a más los tiempos de todas sus tareas adyacentes que fueron asignadas después de T_a en el mismo procesador.
Para obtener el "tiempo separado" $T_{sep}(T_i,T_a)$ se debe determinar en cual procesador (p) conviene ejecutar la tarea T_i, este procesador elegido es aquel que minimice el tiempo de ejecución acumulado en p más el tiempo de ejecutar la tarea T_i. Una vez que p ha sido elegido $T_{sep}(T_i,T_a)$ se calcula como se muestra en la fórmula 5.2:

$$T_{sep}(T_i, T_a) = Va\,(y) + Vi\,(x) - Th_{i,a}\,(x, y) + TC_{i,a}(p, q) + TC_{a,i}(q, p)$$

Fórmula 5.2

donde:

$V_a(y)$ es tiempo de ejecución de T_a en el tipo de procesador y.
x es el tipo de procesador al que pertenece p.
y es el tipo de procesador al que está asignada la tarea T_a.
$V_i(x)$ es tiempo de ejecución de T_i en el tipo de procesador x.
$Th_{i,a}(x,y)$ es igual a $h_{i,a}(x,y)$ * tiempo de ejecutar T_a en el tipo de procesador y.
$TC_{i,a}(p,q)$ es igual a $C_{i,a}$ * el costo de comunicación (de un byte) entre p (procesador en el que se quiere asignar) y q (procesador al que está asignada T_a) y sumado al tiempo de startup de p.
$TC_{a,i}(q,p)$ es igual a $C_{a,i}$ * el costo de comunicación (de un byte) entre q (procesador al que está asignada T_a) y p (procesador en el que se quiere asignar) y sumado al tiempo de startup de q.

En el siguiente pseudocódigo se muestra el algoritmo explicado anteriormente:

Procedure AsignarTareaAProcesador (G)

{Este algoritmo parte de una lista inicial donde las tareas están ordenadas de menor a mayor valor de $LN(T_i)$, y realiza los siguientes pasos para obtener la asignación de las tareas a los procesadores:

a. Seleccionar el primer nivel n sin asignar
b. Calcular la máxima ganancia max_ganancia (T_i) para aquellas tareas de nivel n que aun no han sido asignadas, junto con el procesador proc_opt (T_i) al cual debe ser asignada la tarea. Ordenar las tareas en forma decreciente de acuerdo a max_ganancia (T_i).
c. Asignar la primer tarea T_i de la lista generada en b. al procesador proc_opt (T_i).
d. Si aún existen tareas sin asignar de nivel n, se repite el paso b.
e. Si aún existen niveles sin procesar se vuelve al paso a.

}

Procedure CalcularMaximaGanancia
(G, i, proc_opt,max_ganancia)

```
{
    Para cada tarea Ta adyacente a t ya asignada
      {
          tjunto = acum(Ta) + Vi(x)
tsep= calcularTiempoSeparado(i,Ta,proc)
if (tjunto<tsep) proc=procesador en donde está asignada Ta.
if (tsep - tjunto > max_ganacia)
            {
              max_ganancia= tsep-tjunto
              proc_opt=proc
             }
     }
 }
```

Function calcularTiempoSeparado (i,T_a,proc) :float

```
{
  Para cada procesador p (diferente a donde se asigno Ta)
    {
tiempo = Tacum(p) + Vi(x)
if (tiempo < min)
         {
min = tiempo
proc = p
         }
    }
tsep= VTa(y) + Vi(x) – (VTa (y) * hiTa, (x,y)) + TC(i,a) + TC(a,i)
calcularTiempoSeparado = tsep
}
```

Una vez que el algoritmo MATEHA ha decidido qué tareas deben asignarse a cada uno de los procesadores que componen la arquitectura, se debe decidir el orden en el cual deben ejecutarse dentro de cada procesador. Para esto se desarrolló el algoritmo SIMULAHA, el cual decide el orden en que se ejecutarán las tareas dentro de un procesador [41].

5.2.2 Ejemplo del algoritmo de mapping MATEHA

Basado en el grafo generado por el modelo TTIGHA (mostrado en el capítulo 4) se ilustra el funcionamiento del algoritmo MATEHA. En este caso el grafo G está formado por 8 tareas ($T_0..T_7$) como se ve en la figura 5.1.
En la figura 5.2 se visualiza un esquema que representa la arquitectura utilizada para este ejemplo. La misma está formada por dos tipos diferentes de procesadores con dos procesadores correspondiente al primer tipo (P_0 y P_1) y uno al segundo (P_3). Además la red de interconexión está formada por dos grupos de comunicación, uno que conecta a los dos procesadores del tipo 1; y otra para conectar cada uno de estos procesadores con el restante.

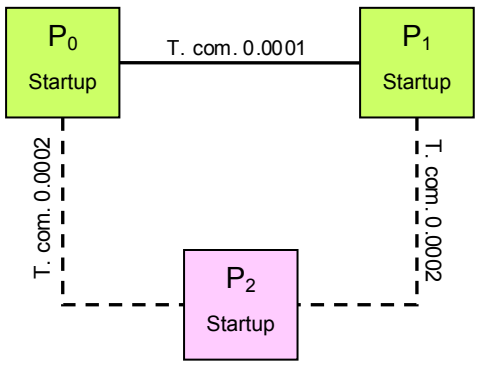


Figura 5.2- Esquema de la arquitectura

A partir de la información de la aplicación y de la arquitectura mostradas en las figuras 5.1 y 5.2 respectivamente, comienza la ejecución del algoritmo MATEHA.

El primer paso de este algoritmo consiste en calcular nivel de cada una de las tareas, generando lo siguiente:
Nivel 0 = $\{T_0\}$
Nivel 1 = $\{T_1, T_2\}$
Nivel 2 = $\{T_3, T_4, T_5\}$
Nivel 3 = $\{T_6, T_7\}$
Luego, para cada uno de los cuatro niveles se asignan las tareas a los procesadores.

Asignación de las tareas de nivel 0
Como existe una única tarea perteneciente a este nivel se asigna al procesador que minimice su tiempo de ejecución, es decir, P_0.
En la tabla 5.1 se visualiza como se va actualizando la asignación de las tareas:

Procesador	Tareas Asignadas	Tiempo Acumulado
P_0	T_0	5
P_1		
P_2		

Tabla 5.1- Asignación de tareas a procesadores y Tiempo de ejecución en cada procesador después de haber asignado T_0

Además la tabla 5.2 muestra el acumulado de cada tarea asignada:

Tarea	Tiempo Acumulado
T_0	5
T_1	
T_2	
T_3	
T_4	
T_5	
T_6	
T_7	

Tabla 5.2 - Acumulado en cada tarea

Asignación de las tareas de nivel 1
En este nivel existen 2 tareas (T_1 y T_2), por lo tanto se debe calcular la ganancia de cada una de ellas de la siguiente manera:

- ✓ Ganancia de T_1: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la única adyacente asignada es T_0

- tjunto (T0):120 (tiempo ejecución de la tarea T1 en P0) + 5 (tiempo acumulado de la tarea T0 en P0)= 125
- tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 5 (tiempo T0) + 120 (tiempo de T1 en P1) – 0 (grado concurrencia entre T0 y T1 * tiempo de T1 en P1) + 1.01 (0.0001 * 10000 + 0.01) = 126.01
- dif = tsep – tjunto = 1.01
- ganancia = 1.01 {Procesador P0}

✓ Ganancia de T_2: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la única adyacente asignada es T_0

- tjunto: 50 (tiempo ejecución de la tarea T2 en P0) + 5 (tiempo acumulado de la tarea T0 en P0) = 55
- tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 5 (tiempo T0) + 50 (tiempo de T2 en P1) – 0 (grado concurrencia entre T0 y T2 * tiempo de T2 en P1) + 0.51 (0.0001 * 5000 + 0.01) = 55.51
- dif = tsep – tjunto = 0.51
- ganancia = 0.51 {Procesador P0}

Luego, para la asignación, se ordenan en forma decreciente según el valor de la ganancia, quedando en este caso T_1, T_2. Por lo tanto, dado que la ganancia es positiva, se asigna T_1 al procesador P_0, lo cual se visualiza en la tabla 5.3:

Procesador	Tareas Asignadas	Tiempo Acumulado
P_0	T_0,T_1	125
P_1		
P_2		

Tabla 5.3- Asignación de tareas a procesadores y Tiempo de ejecución en cada procesador luego de haber asignado T_0, T_1

A partir de la asignación de T_1 también se actualiza la tabla de los valores de cómputo acumulados por tarea. Esto se muestra en la tabla 5.4

Tarea	Tiempo Acumulado
T_0	125
T_1	120
T_2	
T_3	
T_4	
T_5	
T_6	
T_7	

Tabla 5.4- Acumulado en cada tarea

A continuación queda por asignar T_2 para lo cual se vuelven a calcular los valores de ganancia para poder determinar el procesador en el cual debe ejecutarse:

- ✓ Ganancia de T_2
  - Con respecto a T_0
    - Tjunto: 50 (tiempo ejecución de la tarea T2 en P0) + 125 (tiempo acumulado de la tarea T0 en P0) = 175
    - Tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 5 (tiempo T0) + 50 (tiempo de T2 en P1) – 0 (grado concurrencia entre T0 y T2 * tiempo de T2 en P1) + 0.51 (0.0001 * 5000 + 0.01) = 55.51
    - dif: tsep – tjunto = -119.49
  - Con respecto a T_1
    - Tjunto: 50 (tiempo ejecución de la tarea T_2 en P_0) + 120 (tiempo acumulado de la tarea T_1 en P_0) = 170
    - Tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 120 (tiempo T_1) + 50 (tiempo de T_2 en P_1) – 40 (grado concurrencia entre T_1 y T_2* tiempo de T_2 en P_1) + 0.81 (0.0001 * 8000 + 0.01) = 130.81
    - dif: tsep – tjunto = -39.19
  - Ambas diferencias son menores a 0 ⇒ganancia = 0 {Procesador P_1}.

Dados los valores calculados anteriormente, la tarea T_2 debe asignarse al procesador P_1, y la tabla 5.5 refleja esta asignación.

Procesador	Tareas Asignadas	Tiempo Acumulado
P0	T_0,T_1	125
P1	T_2	50
P2		

Tabla 5.5- Asignación de tareas a procesadores y Tiempo de ejecución en cada procesador luego de haber asignado T_0, T_1y T_2

También se muestra la tabla 5.6 la cual mantiene los acumulados de cada una de las tareas ya asignadas

Tarea	Tiempo Acumulado
T_0	125
T_1	120
T_2	50
T_3	
T_4	
T_5	
T_6	
T_7	

Tabla 5.6- Acumulado en cada tarea

Asignación de las tareas de nivel 2

En este nivel existen 3 tareas (T_3 , T_4 y T_5), por lo tanto se debe calcular la ganancia de cada una de ellas de la siguiente manera:

- ✓ Ganancia de T_3: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la única adyacente asignada es T_1
  - Tjunto: 140 (tiempo ejecución de la tarea T3 en P0) + 120 (tiempo acumulado de la tarea T1 en P0)= 260
  - Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 120 (tiempo T1) + 175 (tiempo de T3 en P2) – 40 (grado concurrencia entre T1 y T3 * tiempo de T3 en P2) + 0.31 (0.0002 * 1500 + 0.01) + 0.38 (0.0002 * 1800 + 0.02) = 255.69
  - ganancia: Tsep – Tjunto = -4.31 $\Rightarrow$ganancia = 0
- ✓ Ganancia de T4: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la única adyacente asignada es T2

- Tjunto: 55 (tiempo ejecución de la tarea T4 en P1) + 50 (tiempo acumulado de la tarea T2 en P1) = 105
- Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 50 (tiempo T2) + 95 (tiempo de T4 en P2) – 25 (grado concurrencia entre T2 y T4 * tiempo de T4 en P2) + 0.21 (0.0002 * 1000 + 0.01) = 120.21
- ganancia: tsep – tjunto = 15.21

✓ Ganancia de T5: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la única adyacente asignada es T2

- Tjunto: 25 (tiempo ejecución de la tarea T5 en P1) + 50 (tiempo acumulado de la tarea T2 en P1) = 75
- Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 50 (tiempo T2) + 35 (tiempo de T5 en P2) – 0 (grado concurrencia entre T2 y T5 * tiempo de T5 en P2) + 1.35 (0.0002 * 6700 + 0.01) = 86.35
- ganancia: tsep – tjunto = 11.35

Luego, para la asignación, se ordenan en forma decreciente según el valor de la ganancia, quedando en este caso T_4,T_5,T_3. Por lo tanto se asigna T_4 a P_1, lo cual se visualiza en la tabla 5.7:

Procesador	Tareas Asignadas	Tiempo Acumulado
P0	T_0,T_1	125
P1	T_2 T_4	105
P2		

Tabla 5.7- Asignación de tareas a procesadores y Tiempo de ejecución en cada procesador luego de haber asignado T_0, T_1y T_2 yT_4

También se muestra la tabla 5.8 la cual mantiene los acumulados de cada una de las tareas ya asignadas

Tarea	Tiempo Acumulado
T_0	125
T_1	120
T_2	105
T_3	
T_4	55
T_5	
T_6	
T_7	

Tabla 5.8- Acumulado en cada tarea

A continuación quedan por asignar T_3 y T_5 para lo cual se vuelven a calcular los valores de ganancia para poder determinar el procesador en el cual deben ejecutarse:

- ✓ Ganancia de T_3: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la única adyacente asignada es T_1
  - Tjunto: 140 (tiempo ejecución de la tarea T3 en P0) + 120 (tiempo acumulado de la tarea T1 en P0)= 260
  - Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 120 (tiempo T1) + 175 (tiempo de T3 en P2) – 40 (grado concurrencia entre T1 y T3 * tiempo de T3 en P2) + 0.31 (0.0002 * 1500 + 0.01) + 0.38 (0.0002 * 1800 + 0.02) = 255.69
  - ganancia: Tsep – Tjunto = -4.31 ⇒ganancia = 0

- ✓ Ganancia de T5: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso las adyacentes asignadas son T2 yT4
  - Con respecto a T2
    - Tjunto: 25 (tiempo ejecución de la tarea T5 en P1) + 105 (tiempo acumulado de la tarea T2 en P1) = 130
    - Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 50 (tiempo T2) + 35 (tiempo de T5 en P2) – 0 (grado concurrencia entre T2 y T5 * tiempo de T2 en P1) + 1.35 (0.0002 * 6700 + 0.01) = 86.25
    - dif: tsep – tjunto = -43.65

- Con respecto a T4
  - Tjunto: 25 (tiempo ejecución de la tarea T5 en P1) + 55 (tiempo acumulado de la tarea T4 en P1) = 80
  - Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 55 (tiempo T4) + 35 (tiempo de T5 en P2) – 20 (grado concurrencia entre T4 y T5* tiempo de T5 en P2) + 0.41 (0.0002 * 2000 + 0.01) = 70.41
  - dif: tsep – tjunto = -9.59

- Ambas diferencias son menores a 0 $\Rightarrow$ganancia = 0.

Dados los valores calculados anteriormente, se elige la tarea T_5 debe asignarse al procesador P_2, y la tabla 5.9 refleja esta asignación.

Procesador	Tareas Asignadas	Tiempo Acumulado
P_0	T_0,T_1	125
P_1	T_2 T_4	105
P_2	T_5	35

Tabla 5.9- Asignación de tareas a procesadores y Tiempo de ejecución en cada procesador luego de haber asignado T_0, T_1, T_2 , T_4 yT_5

También se muestra la tabla 5.10 la cual mantiene los acumulados de cada una de las tareas ya asignadas

Tarea	Tiempo Acumulado
T_0	125
T_1	120
T_2	105
T_3	
T_4	55
T_5	35
T_6	
T_7	

Tabla 5.10- Acumulado en cada tarea

La única tarea que falta asignar de este nivel es T_3, por lo tanto se vuelve a calcular la ganancia:

- ✓ Ganancia de T3: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso las adyacentes asignadas son T1 yT5

- Con respecto a T1
  - Tjunto: 140 (tiempo ejecución de la tarea T3 en P0) + 120 (tiempo acumulado de la tarea T1 en P0) = 260
  - Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 120 (tiempo T1) + 175 (tiempo de T3 en P2) – 40 (grado concurrencia entre T3 y T1 * tiempo de T1 en P0) + 0.31 (0.0002 * 1500 + 0.01) + 0.38 (0.0002 * 1800 + 0.02) = 255.69
  - dif: tsep – tjunto = -4.31

- Con respecto a T5
  - Tjunto: 175 (tiempo ejecución de la tarea T3 en P2) + 35 (tiempo acumulado de la tarea T5 en P2) = 210
  - Tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 35 (tiempo T5) + 140 (tiempo de T3 en P1) – 35 (grado concurrencia entre T3 y T5* tiempo de T5 en P2) + 0.31 (0.0002 * 1500 + 0.01) = 140.31
  - dif: tsep – tjunto = -69.69

- Ambas diferencias son menores a 0 $\Rightarrow$ganancia = 0.

Dados los valores calculados anteriormente, se elige que la tarea T_3 debe ejecutarse de manera separada, por lo tanto se elige el procesador P_2, y la tabla 5.11 refleja esta asignación.

Procesador	Tareas Asignadas	Tiempo Acumulado
P_0	T_0,T_1	125
P_1	T_2 T_4	105
P_2	T_5 T_3	210

Tabla 5.11- Asignación de tareas a procesadores y Tiempo de ejecución en cada procesador luego de haber asignado T_0, T_1, T_2 , T_4 ,T_5 yT_5

También se muestra la tabla 5.12 la cual mantiene los acumulados de cada una de las tareas ya asignadas

Tarea	Tiempo Acumulado
T_0	125
T_1	120
T_2	105
T_3	175
T_4	55
T_5	35
T_6	
T_7	

Tabla 5.12- Acumulado en cada tarea

Asignación de las tareas de nivel 3

En este están las tareas T_6 yT_7, por lo tanto se debe calcular la ganancia de las mismas de la siguiente manera:

- ✓ Ganancia de T6: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la adyacente asignada es T5
  - Tjunto(T5): 160 (tiempo ejecución de la tarea T6 en P2) + 35 (tiempo acumulado de la tarea T5 en P2) = 195
  - Tsep(T5): de acuerdo a los cálculos se elige el procesador P1, entonces = 35 (tiempo T5) + 120 (tiempo de T6 en P1) – 20 (grado concurrencia entre T5 y T6 * tiempo de T5 en P2) + 0.04 (0.0002 * 100 + 0.02) = 135.04
  - dif: Tsep – Tjunto = -59.96

- ✓ Ganancia de T7: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso se tienen dos adyacentes asignadas T3 T4 yT5
  - o Con respecto a T3
    - Tjunto: 122 (tiempo ejecución de la tarea T7 en P2) + 175 (tiempo acumulado de la tarea T3 en P2) = 297
    - Tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 175 (tiempo T3) + 110 (tiempo de T7 en P1) – 80 (grado concurrencia entre T3 y T7 * tiempo de T7 en P1) + 0.24 (0.0002 * 1100 + 0.02) = 205.24
    - dif1: tsep – tjunto = -91.76

- Con respecto a T4
  - Tjunto: 110 (tiempo ejecución de la tarea T7 en P1) + 55 (tiempo acumulado de la tarea T4 en P2) = 165
  - Tsep: de acuerdo a los cálculos se elige el procesador P0, entonces = 55 (tiempo T4) + 110 (tiempo de T7 en P0) – 20 (grado concurrencia entre T4 y T7* tiempo de T7 en P2) + 0.11 (0.0001 * 500 + 0.01) = 145.11
  - dif2: tsep – tjunto = -19.89

- Con respecto a T5
  - Tjunto: 122 (tiempo ejecución de la tarea T7 en P2) + 35 (tiempo acumulado de la tarea T5 en P2) = 157
  - Tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 35 (tiempo T5) + 110 (tiempo de T7 en P1) – 35 (grado concurrencia entre T5 y T7* tiempo de T7 en P1) + 0.42 (0.0002 * 2000 + 0.02) = 110.42
  - dif: tsep – tjunto = -46.58

- Todaslas diferencias son menores a 0 $\Rightarrow$ganancia = 0.

Dado que la ganancia es 0 para ambas tareas se decide la ejecución de T_6 en el procesador P_1, la tabla 5.13 muestra dicha asignación:

Procesador	Tareas Asignadas	Tiempo Acumulado
P_0	T_0, T_1	125
P_1	$T_2\ T_4\ T_6$	225
P_2	$T_5\ T_3$	210

Tabla 5.13- Asignación de tareas a procesadores y Tiempo de ejecución en cada procesador luego de haber asignado T_0, T_1, T_2 , T_4 ,T_5 yT_3 yT_6

A su vez la tabla 5.14 la cual mantiene los acumulados de cada una de las tareas ya asignadas

Tarea	Tiempo Acumulado
T_0	125
T_1	120
T_2	105
T_3	175
T_4	55
T_5	35
T_6	120
T_7	

Tabla 5.14- Acumulado en cada tarea

Por último falta asignar la tarea T_7, por lo cual hay que recalcular su ganancia:

- ✓ Ganancia de T7: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso se tienen dos adyacentes asignadas T3 T4 T5 y T6

  - Con respecto a T3
    - Tjunto: 122 (tiempo ejecución de la tarea T7 en P2) + 175 (tiempo acumulado de la tarea T3 en P2) = 297
    - Tsep: de acuerdo a los cálculos se elige el procesador P0, entonces = 175 (tiempo T3) + 110 (tiempo de T7 en P0) – 80 (grado concurrencia entre T3 y T7 * tiempo de T7 en P0) + 0.24 (0.0002 * 1100 + 0.02) = 205.24
    - dif1: Tsep – Tjunto = -91.76

  - Con respecto a T4
    - Tjunto: 110 (tiempo ejecución de la tarea T7 en P1) + 55 (tiempo acumulado de la tarea T4 en P2) = 165
    - Tsep: de acuerdo a los cálculos se elige el procesador P0, entonces = 55 (tiempo T4) + 110 (tiempo de T7 en P0) – 20 (grado concurrencia entre T4 y T7* tiempo de T7 en P2) + 0.11 (0.0001 * 500 + 0.01) = 145.11
    - dif2: Tsep – Tjunto = -19.89

  - Con respecto a T5

- Tjunto: 122 (tiempo ejecución de la tarea T7 en P2) + 35 (tiempo acumulado de la tarea T5 en P2) = 157
- Tsep: de acuerdo a los cálculos se elige el procesador P0, entonces = 35 (tiempo T5) + 110 (tiempo de T7 en P0) – 35 (grado concurrencia entre T5 y T7* tiempo de T7 en P0) + 0.42 (0.0002 * 2000 + 0.02) = 110.42
- dif: tsep – tjunto = -46.58

o Con respecto a T6
- Tjunto: 110 (tiempo ejecución de la tarea T7 en P1) + 120 (tiempo acumulado de la tarea T6 en P1) = 230
- Tsep: de acuerdo a los cálculos se elige el procesador P0, entonces = 120 (tiempo T6) + 110 (tiempo de T7 en P0) – 100 (grado concurrencia entre T6 y T7* tiempo de T7 en P0) + 0.26 (0.0001 * 2500 + 0.01) = 130.26
- dif: tsep – tjunto = -99.74

o Todaslas diferencias son menores a 0 $\Rightarrow$ganancia = 0.

Dado que la ganancia es 0 se elige ejecutar separado en P_0, la tabla 5.15 muestra dicha asignación:

Procesador	Tareas Asignadas	Tiempo Acumulado
P_0	$T_0, T_1 T_7$	235
P_1	$T_2 T_4 T_6$	225
P_2	$T_5 T_3$	210

Tabla 5.15- Asignación de tareas a procesadores y Tiempo de ejecución en cada procesador luego de haber asignado T_0, T_1, T_2 , T_4 ,T_5 , T_3 ,T_6 yT_7

A su vez la tabla 5.16 la cual mantiene los acumulados de cada una de las tareas ya asignadas

Tarea	Tiempo Acumulado
T_0	125
T_1	120
T_2	105
T_3	175
T_4	55
T_5	35
T_6	120
T_7	110

Tabla 5.16- Acumulado en cada tarea

Como se puede ver en la tabla 5.15, en el procesador P_0 quedan asignadas las tareas T_0, T_1, y T_7; en el procesador P_1 quedan asignadas T_2, T_4 yT_6; por último en el procesador P_2 quedan asignados T_5 yT_3.
Luego de generar la asignación utilizando el algoritmo MATEHA se ejecuta el algoritmo de simulación SIMULAHA por medio del cual se determina el orden de ejecución de las tareas de un procesador y el tiempo final de cada uno. Para realizar esta simulación se dispone de información que incluye como es la composición de tareas, es decir, que subtareas la componen; y el tiempo de ejecución que requiere cada una de las subtareas en cada tipo de procesador. Estos datos son mostrados en la figura 5.3:

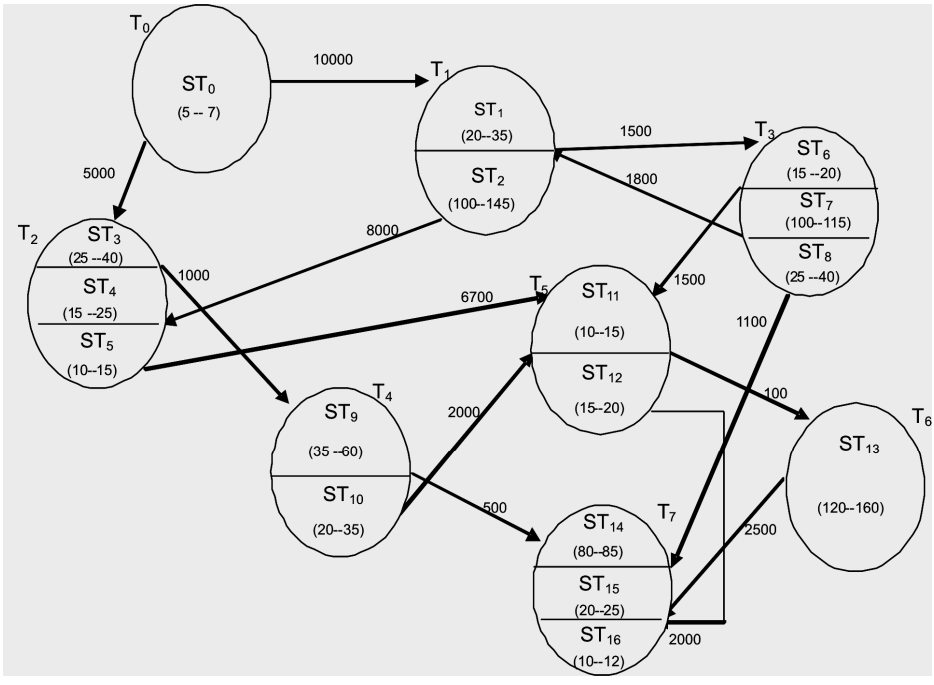


Figura 5.3 Composición de las tareas y tiempos de cada subtarea

A continuación la figura 5.4 muestra los resultados de la simulación:

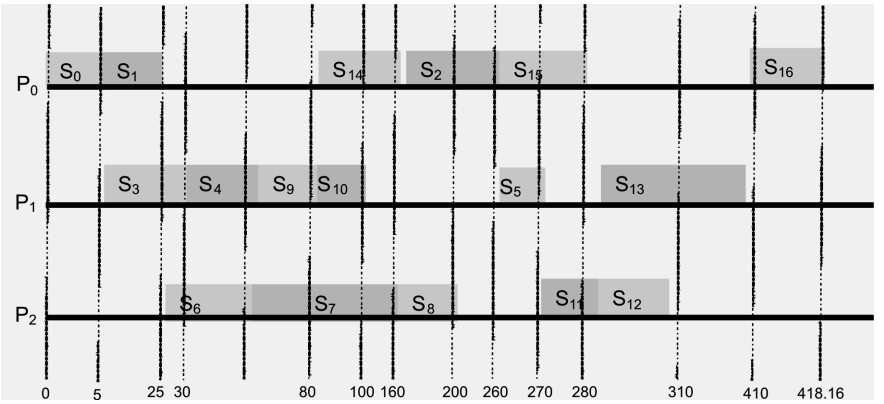


Figura 5.4 Simulación de la ejecución de la aplicación

De la figura anterior se puede ver que la aplicación termina su ejecución en el instante 418.16 luego de la ejecución de la subtarea S_{16}. Además se pueden notar espacios ociosos en todos los procesadores.

5.3 Algoritmo MATEHAIB

Para realizar la elección del procesador (p) en el cual una tarea (t) debe ejecutarse, el algoritmo MATEHA solo considera el tiempo de ejecución acumulado de las tareas asignadas a p; sin tener en cuenta el momento en que realmente cada una de esas tareas podría ejecutarse. Este hecho podría provocar que el procesador elegido para la ejecución de una tarea no sea el óptimo.
Otra desventaja que produce utilizar el algoritmo MATEHA es que luego que el mismo termina su ejecución se debe ejecutar un algoritmo que realice la simulación de la ejecución de la aplicación. Esto ocurre ya que el algoritmo MATEHA decide donde debe ejecutarse cada tarea pero no en que momento debe ejecutarse cada tarea. En esta tesis esta simulación se hace por medio del algoritmo de simulación SIMULAHA.
Para tratar de encontrar una solución a estos problemas planteados en el algoritmo MATEHA se implementó un nuevo algoritmo MATEHAIB. Este algoritmo está basado en el MATEHA incorporando el concepto de "insertion-based" considerado en el algoritmo HEFT.

5.3.1 Descripción del algoritmo

Este algoritmo de mapping extrae los parámetros del modelo TTIGHA, en los que se fundamenta la heurística de asignación del algoritmo.
Al igual que el algoritmo MATEHA primero se define el *nivel* de cada tarea, estevalor será utilizado para realizar la asignación de las tareas del grafo con cierta prioridad y luego para cada tarea se calcula el procesador que genera la máxima ganancia. La forma de elegir el procesador que maximice la ganancia es una de las principales diferencias con el algoritmo MATEHA.
En resumen, el algoritmo MATEHAIB puede dividirse en dos pasos: calcular el nivel de cada nodo del grafo y asignar las tareas a los procesadores.
La forma de calcular el nivel de cada nodo es igual a la explicada en el algoritmo MATEHA (apartado 5.2.1.1). Por lo tanto solo de detallará el paso que explica el procesador que se debe elegir para obtener la máxima ganancia.

5.3.1.1 Asignación de tareas a procesadores

Como se definió en el capítulo anterior, el modelo TTIGHA mantiene para cada par de tareas adyacentes T_i y T_j como mínimo tres valores: el tiempo de ejecución estimado de cada una de ellas en los diferentes tipos de procesadores de la arquitectura, el volumen de comunicación y el grado de concurrencia entre ambas tareas (este último valor depende del tipo de procesador en donde se encuentran T_i y T_j). Además de estos valores, se dispone de información más detallada que modeliza que subtareas forman cada una de las tareas y que comunicación existe entre las subtareas.
Toda esta información adicional es necesaria para implementar las modificaciones de este algoritmo con respecto al MATEHA. Estos cambios se ven reflejados en dos puntos, el primero de ellos es la forma de elegir el procesador en el cual asignar una determinada tarea, y el otro está relacionado en la forma de asignar una tarea dentro del procesador ya elegido.
Para la etapa de asignación, se utiliza el conjunto de tareas ordenadas de manera creciente según el nivel de las mismas.
La etapa de asignación se realiza por niveles, es decir, primero se asignan todas las tareas de nivel 0, una vez que las mismas han sido asignadas se prosigue con las de nivel 1 y así sucesivamente para todos los niveles del grafo.

Dado un nivel n, para realizar la asignación se elige aquella tarea aún no asignada perteneciente al nivel n que genera la máxima ganancia al asignar dicha tarea a un procesador.
La ganancia de una tarea T_i se calcula de la siguiente manera: para cada una de sus tareas adyacentes ya asignadas a un procesador T_a ($T_a \in$ al conjunto de adyacentes asignadas de T_i) se calcula la diferencia entre el costo de ejecutar T_i en un procesador diferente al que ya ha sido asignado T_a (tiempo separado) y el costo de ejecutar T_i en el mismo procesador al que ya ha sido asignado T_a (tiempo junto). Luego se elige el procesador p que maximice esa ganancia. Un caso particular es que la tarea T_i no tenga ninguna tarea adyacente asignada, y en ese caso se elige el procesador p en el cual conviene ejecutar la tarea T_i.
El "tiempo junto" de $T_{junto}(T_i,T_a)$ se obtiene como la suma del tiempo acumulado de $T_a$más el tiempo de ejecutar T_i en el mismo procesador al que está asignado T_a. El tiempo acumulado de T_a es el tiempo de $T_a$más los tiempos de todas sus tareas adyacentes que fueron asignadas después de T_a en el mismo procesador.
Para obtener el "tiempo separado" $T_{sep}(T_i,T_a)$ se debe determinar en cual procesador (p) conviene ejecutar la tarea T_i (p es un procesador diferente al cual T_a fue asignada), este procesador elegido es aquel que minimice el "tiempo intermedio" en p. Una vez que p ha sido elegido $T_{sep}(T_i,T_a)$ se calcula como se muestra en la fórmula 5.3:

$$\mathrm{Tsep}(T_i, T_a) = Va\,(y) + Vi\,(x) - h_{i,a}\,(x,y) + TC_{i,a}\,(p,q) + TC_{a,i}\,(q,p)$$

Fórmula 5.3

donde:

$V_a(y)$ es tiempo de ejecución de T_a en el tipo de procesador y.
x es el tipo de procesador al que pertenece p.
y es el tipo de procesador al que está asignada la tarea T_a.
$V_i(x)$ es tiempo de ejecución de T_i en el tipo de procesador x.
$Th_{i,a}(x,y)$ es igual a $h_{i,a}(x,y)$ * tiempo de ejecutar T_a en el tipo de procesador y.
$TC_{i,a}(p,q)$ es igual a $C_{i,a}$ * el costo de comunicación (de un byte) entre p (procesador en el que se quiere asignar) y q (procesador al que está asignada T_a) y sumado al tiempo de startup de p.
$TC_{a,i}(q,p)$ es igual a $C_{a,i}$ * el costo de comunicación (de un byte) entre q (procesador al que está asignada T_a) y p (procesador en el que se quiere asignar) y sumado al tiempo de startup de q.

Cada procesador mantiene dos listas. La primera (LU) contiene las subtareas que ya fueron ubicadas en él junto al intervalo de tiempo en

el cual se ejecutará. La otra (LNU) contiene el conjunto de subtareas que pertenecen a tareas asignadas al procesador pero aún no han podidos ser ubicadas en un intervalo de tiempo dado que alguna de sus adyacentes no ha sido ubicada.
Al asignar una tarea T_i a un procesador p, cada subtarea St_k perteneciente a T_i se intenta ubicar en el procesador en un instante de tiempo en el cual todas las subtareas adyacentes a ella han finalizado (inclusive su predecesora dentro de T_i en caso de que posea), esto puede ser en un intervalo libre entre dos subtareas ya ubicadas en p, o bien en un intervalo al final de estas. Si la subtarea St_k no puede ser ubicada se agrega a la lista LNU de p. Cada vez que una subtarea St_k es agregada a la lista LU de algún procesador, se intenta ubicar todas sus subtareas predecesoras que pertenezcan a tareas ya asignadas.
Al intentar asignar una tarea T_i dentro de un procesador p, el "tiempo intermedio" del mismo se calcula dependiendo de cuál de estas situaciones se cumple:

- todas las subtareas de T_i han sido ubicadas: en este caso el tiempo intermedio es el instante en el cual la última subtarea de T_i termina su ejecución.
- No se han ubicado todas las subtareas de T_i: en este caso el tiempo intermedio es el instante en el cual la última subtarea ubicada en p termina su ejecución más el tiempo de todas las subtareas que contiene la lista LNU de p.

Una vez que el algoritmo MATEHAIB terminó su ejecución, todas las tareas han sido asignadas a algún procesador de los que componen la arquitectura. Esta asignación además determina en que instante debe ejecutarse cada subtarea dentro de un procesador.

5.3.2 Ejemplo del algoritmo de mapping MATEHAIB

Para realizar el ejemplo se considera el mismo grafo G generado por el modelo TTIGHA (explicado en el capítulo 4). El grafo G está formado por 8 tareas (T_0..T_7) como se ve en la figura 5.1.
A su vez, en la figura 5.2 se visualiza la arquitectura utilizada para el ejemplo anterior la cual también será utilizada en este ejemplo. Dicha arquitectura está formada por dos tipos diferentes de procesadores, dos de ellos corresponden al primer tipo (P_0, P_1) y el restante pertenece al segundo tipo (P_2). En la figura también se muestra la red de interconexión formada por dos grupos de comunicaciones, uno que conecta a los dos procesadores del primer grupo; y otra para conectar cada uno de estos dos procesadores con el restante.

También se utiliza para este ejemplo la figura 5.3, la cual muestra como es la composición de cada una de las tareas en subtareas, y los tiempos de cada una en los diferentes tipos de procesadores. Además en la figura se muestra el volumen de información entre las subtareas.
A partir de los datos mostrados en las figuras 5.1, 5.2 y 5.3 comienza la ejecución del algoritmo MATEHAIB.
Al igual que en el algoritmo MATEHA, el primer paso del algoritmo MATEHAIB consiste en calcular nivel de cada una de las tareas, generando lo siguiente:

Nivel 0 = $\{T_0\}$
Nivel 1 = $\{T_1, T_2\}$
Nivel 2 = $\{T_3, T_4\}$
Nivel 3 = $\{T_5\}$

Luego, para cada uno de los cuatro niveles se asignan las tareas a los procesadores.

Asignación de las tareas de nivel 0
Como existe una única tarea perteneciente a este nivel se asigna al procesador que minimice su tiempo de ejecución, es decir, P_0.
En la Tabla 5.17 se visualiza como se va actualizando la asignación de las tareas, en cada procesador. A su vez, se muestra los intervalos en los cuales se asigna cada tarea:

Procesador	LU	LNU
P_0	ST_0 [0..5]	
P_1		
P_2		

Tabla 5.17- Asignación de la tarea T_0 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 5.18 muestra el acumulado de cada tarea asignada:

Tarea	Tiempo Acumulado
T_0	5
T_1	
T_2	
T_3	
T_4	
T_5	
T_6	
T_7	

Tabla 5.18- Acumulado en cada tarea

Asignación de las tareas de nivel 1

En este nivel existen 2 tareas (T_1 y T_2), por lo tanto se debe calcular la ganancia de cada una de ellas de la siguiente manera:

- ✓ Ganancia de T_1: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la única adyacente asignada es T_0
  - tjunto (T0):120 (tiempo ejecución de la tarea T1 en P0) + 5 (tiempo acumulado de la tarea T0 en P0)= 125
  - tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 5 (tiempo T0) + 120 (tiempo de T1 en P1) – 0 (grado concurrencia entre T0 y T1 * tiempo de T1 en P1) + 1.01 (0.0001 * 10000 + 0.01) = 126.01
  - dif = tsep – tjunto = 1.01
  - ganancia = 1.01 {Procesador P0}

- ✓ Ganancia de T2: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la única adyacente asignada es T0
  - tjunto: 50 (tiempo ejecución de la tarea T2 en P0) + 5 (tiempo acumulado de la tarea T0 en P0) = 55
  - tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 5 (tiempo T0) + 50 (tiempo de T2 en P1) – 0 (grado concurrencia entre T0 y T2 * tiempo de T2 en P1) + 0.51 (0.0001 * 5000 + 0.01) = 55.51
  - dif = tsep – tjunto = 0.51
  - ganancia = 0.51 {Procesador P0}

Luego, para la asignación, se ordenan en forma decreciente según el valor de la ganancia, quedando en este caso T_1, T_2. Por lo tanto, dado que la ganancia es positiva, se asigna T_1 al procesador P_0, lo cual se visualiza en la tabla 5.19:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_1 [5..25]	ST_2
P_1		
P_2		

Tabla 5.19- Asignación de la tarea T_1 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 5.20 muestra el acumulado de cada tarea asignada:

Tarea	Tiempo Acumulado
T_0	5
T_1	120
T_2	
T_3	
T_4	
T_5	
T_6	
T_7	

Tabla 5.20- Acumulado en cada tarea

A continuación queda por asignar T_2 para lo cual se vuelven a calcular los valores de ganancia para poder determinar el procesador en el cual debe ejecutarse.

- ✓ Ganancia de T2: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso se tienen dos adyacentes asignadas T0 yT1
  - Con respecto a T0
    - Tjunto: 50 (tiempo ejecución de la tarea T2 en P0) + 125 (tiempo acumulado de la tarea T0 en P0) = 175
    - Tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 5 (tiempo T0) + 50 (tiempo de T2 en P1) – 0 (grado concurrencia entre T0 y T2 * tiempo de T2 en P1) + 0.51 (0.0001 * 5000 + 0.01) = 55.51
    - dif: tsep – tjunto = -119.49
  - Con respecto a T1
    - Tjunto: 50 (tiempo ejecución de la tarea T2 en P0) + 120 (tiempo acumulado de la tarea T1 en P0) = 170
    - Tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 120 (tiempo T1) + 50 (tiempo de T2 en P1) – 40 (grado concurrencia entre T1 y T2* tiempo de T2 en P1) + 0.81 (0.0001 * 8000 + 0.01) = 130.81
    - dif: tsep – tjunto = -39.19
  - Ambas diferencias son menores a 0 ⇒ganancia = 0 {Procesador P1}.

Dados los valores calculados anteriormente, la tarea T_2 debe asignarse al procesador P_1, y la tabla 5.21 refleja esta asignación.

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_1 [5..25]	ST_2
P_1	ST_3 [5.51..30.51] ST_3 [30.51..45.51]	ST_5
P_2		

Tabla 5.21- Asignación de la tarea T_2 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 5.22 muestra el acumulado de cada tarea asignada:

Tarea	Tiempo Acumulado
T_0	5
T_1	120
T_2	50
T_3	
T_4	
T_5	
T_6	
T_7	

Tabla 5.22- Acumulado en cada tarea

Asignación de las tareas de nivel 2

En este nivel existen 3 tareas (T_3 , T_4 y T_5), por lo tanto se debe calcular la ganancia de cada una de ellas de la siguiente manera:

- ✓ Ganancia de T3: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la única adyacente asignada es T1
  - Tjunto: 140 (tiempo ejecución de la tarea T3 en P0) + 120 (tiempo acumulado de la tarea T1 en P0)= 260
  - Tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 120 (tiempo T1) + 140 (tiempo de T3 en P1) – 25 (grado concurrencia entre T1 y T3 * tiempo de T3 en P1) + 0.16 (0.0001 * 1500 + 0.01) + 0.19 (0.0001 * 1800 + 0.01) = 235.35
  - ganancia: Tsep – Tjunto = -24.65 $\Rightarrow$ganancia = 0

- ✓ Ganancia de T4: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la única adyacente asignada es T2
  - Tjunto: 55 (tiempo ejecución de la tarea T4 en P1) + 50 (tiempo acumulado de la tarea T2 en P1) = 105
  - Tsep: de acuerdo a los cálculos se elige el procesador P0, entonces = 50 (tiempo T2) + 55 (tiempo de T4 en P0) – 25 (grado concurrencia entre T2 y T4 * tiempo de T4 en P0) + 0.11 (0.0001 * 1000 + 0.01) = 80.11
  - ganancia: tsep – tjunto = -24.89 $\Rightarrow$ganancia = 0

- ✓ Ganancia de T5: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la única adyacente asignada es T2
  - Tjunto: 25 (tiempo ejecución de la tarea T5 en P1) + 50 (tiempo acumulado de la tarea T2 en P1) = 75
  - Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 50 (tiempo T2) + 35 (tiempo de T5 en P2) – 0 (grado concurrencia entre T2 y T5 * tiempo de T5 en P2) + 1.35 (0.0002 * 6700 + 0.01) = 86.35
  - ganancia: tsep – tjunto = 11.35

Luego, para la asignación, se ordenan en forma decreciente según el valor de la ganancia, quedando en este caso T_5,T_3,T_4. Por lo tanto se asigna T_5 a P_1, lo cual se visualiza en la tabla 5.23:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_1 [5..25]	ST_2
P_1	ST_3 [5.51..30.51] ST_3 [30.51..45.51]	ST_5 ST_{11} ST_{12}
P_2		

Tabla 5.23- Asignación de la tarea T_5 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 5.24 muestra el acumulado de cada tarea asignada:

Tarea	Tiempo Acumulado
T_0	5
T_1	120
T_2	75
T_3	
T_4	
T_5	25
T_6	
T_7	

Tabla 5.24- Acumulado en cada tarea

A continuación quedan por asignar T_3 y T_4 para lo cual se vuelven a calcular los valores de ganancia para poder determinar el procesador en el cual deben ejecutarse:

- ✓ Ganancia de T3: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso se tienen dos adyacentes asignadas T1 yT5
  - Con respecto a T1
    - Tjunto: 140 (tiempo ejecución de la tarea T3 en P0) + 120 (tiempo acumulado de la tarea T1 en P0) = 260
    - Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 120 (tiempo T1) + 175 (tiempo de T3 en P2) – 40 (grado concurrencia entre T1 y T3* tiempo de T3 en P2) + 0.31 (0.0002 * 1500 + 0.01) + 0.38 (0.0002 * 1800 + 0.02) = 255.69
    - dif: tsep – tjunto = -4.31
  - Con respecto a T5
    - Tjunto: 140 (tiempo ejecución de la tarea T3 en P1) + 25 (tiempo acumulado de la tarea T5 en P1) = 165
    - Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 25 (tiempo T5) + 175 (tiempo de T3 en P2) – 25 (grado concurrencia entre T5 y T3* tiempo de T3 en P2) + 0.17 (0.0002 * 1500 + 0.01) = 175.17
    - dif: tsep – tjunto = 10.17
  - Ganancia T3= 10.17

- ✓ Ganancia de T4: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta las asignadas son T2 yT5
  - o Con respecto a T2
    - Tjunto: 55 (tiempo ejecución de la tarea T4 en P1) + 75 (tiempo acumulado de la tarea T2 en P1) = 130
    - Tsep: de acuerdo a los cálculos se elige el procesador P0, entonces = 50 (tiempo T2) + 55 (tiempo de T4 en P0) – 25 (grado concurrencia entre T2 y T4* tiempo de T4 en P0) + 0.11 (0.0001 * 1000 + 0.01) = 80.11
    - dif: tsep – tjunto = -49.89
  - o Con respecto a T5
    - Tjunto: 55 (tiempo ejecución de la tarea T4 en P1) + 25 (tiempo acumulado de la tarea T5 en P1) = 80
    - Tsep: de acuerdo a los cálculos se elige el procesador P0, entonces = 25 (tiempo T5) + 55 (tiempo de T4 en P0) – 10 (grado concurrencia entre T4 y T5* tiempo de T5 en P1) + 0.21 (0.0001 * 2000 + 0.01) = 70.21
    - dif: tsep – tjunto = -9. 79
  - o Ambas diferencias son negativas Ganancia T4 = 0

Dados los valores calculados anteriormente, se elige la tarea T_3 y debe asignarse al procesador P_1, junto a T_5 ; la tabla 5.25 refleja esta asignación.

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_1 [5..25]	ST_2
P_1	ST_3 [5.51..30.51] ST_3 [30.51..45.51] ST_6 [45.51..60.51] ST_7 [60.51..160.51] ST_8 [160.51..185.51]	ST_5 ST_{11} ST_{12}
P_2		

Tabla 5.25- Asignación de la tarea T_3 a un procesador, manteniendo las listas de c/ procesador

Al terminar la ejecución de la subtarea ST_7, ya pueden ubicarse ST_2 que además permite ubicar ST_5, la cual permite a su vez ubicar ST_{11} . Por lo tanto la tabla 5.26 muestra estos cambios:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_1 [5..25] ST_2 [160.70..260.70]	
P_1	ST_3 [5.51..30.51] ST_3 [30.51..45.51] ST_6 [45.51..60.51] ST_7 [60.51..160.51] ST_8 [160.51..185.51] ST_5 [261.51..271.51] ST_{11} [271.51..281.51]	ST_{12}
P_2		

Tabla 5.26- Asignación de la tarea T_3 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 5.27 muestra el acumulado de cada tarea asignada:

Tarea	Tiempo Acumulado
T_0	5
T_1	120
T_2	75
T_3	140
T_4	
T_5	165
T_6	
T_7	

Tabla 5.27- Acumulado en cada tarea

En este nivel falta recalcular la ganancia de T_4

- ✓ Ganancia de T4: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta las asignadas son T2 yT5
  - o Con respecto a T2
    - Tjunto: 55 (tiempo ejecución de la tarea T4 en P1) + 75 (tiempo acumulado de la tarea T2 en P1) = 130
    - Tsep: de acuerdo a los cálculos se elige el procesador P0, entonces = 50 (tiempo T2) + 55 (tiempo

de T4 en P0) – 25 (grado concurrencia entre T2 y T4* tiempo de T4 en P0) + 0.11 (0.0001 * 1000 + 0.01) = 80.11

- dif: tsep – tjunto = -49.89

- Con respecto a T5
  - Tjunto: 55 (tiempo ejecución de la tarea T4 en P1) + 165 (tiempo acumulado de la tarea T5 en P1) = 220
  - Tsep: de acuerdo a los cálculos se elige el procesador P0, entonces = 25 (tiempo T5) + 55 (tiempo de T4 en P0) – 10 (grado concurrencia entre T4 y T5* tiempo de T5 en P1) + 0.21 (0.0001 * 2000 + 0.01) = 70.21
  - dif: tsep – tjunto = -149. 79

- Ambas diferencias son negativas ⇒ganancia = 0

Dado que la ganancia es 0, debe ejecutarse T_4 de manera separada y se elige el procesador P_0. La tabla 5.28 muestra esta asignación:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_1 [5..25] ST_9 [30.62...65.62] ST_{10} [65.62...85.62] ST_2 [160.70..260.70]	
P_1	ST_3 [5.51..30.51] ST_3 [30.51..45.51] ST_6 [45.51..60.51] ST_7 [60.51..160.51] ST_8 [160.51..185.51] ST_5 [261.51..271.51] ST_{11} [271.51..281.51]	ST_{12}
P_2		

Tabla 5.28- Asignación de la tarea T_4 a un procesador, manteniendo las listas de c/ procesador

De la tabla 5.28 se puede ver que las subtareas ST_9y ST_{10} fueron asignadas en un hueco existente del procesador P_0. Además al ubicar la subtarea ST_{10} esto permite que la subtarea ST_{12} pueda ser ubicada. La tabla 5.29 muestra este cambio:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_1 [5..25] ST_9 [30.62...65.62] ST_{10} [65.62...85.62] ST_2 [160.70..260.70]	
P_1	ST_3 [5.51..30.51] ST_3 [30.51..45.51] ST_6 [45.51..60.51] ST_7 [60.51..160.51] ST_8 [160.51..185.51] ST_5 [261.51..271.51] ST_{11} [271.51..281.51] ST_{11} [281.51..296.51]	
P_2		

Tabla 5.29- Asignación de la tarea T_4 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 5.30 muestra el acumulado de cada tarea asignada:

Tarea	Tiempo Acumulado
T_0	5
T_1	120
T_2	75
T_3	140
T_4	55
T_5	165
T_6	
T_7	

Tabla 5.30- Acumulado en cada tarea

Asignación de las tareas de nivel 3

En este están las tareas T_6 yT_7, por lo tanto se debe calcular la ganancia de las mismas de la siguiente manera:

- ✓ Ganancia de T6: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso la adyacente asignada es T5
  - Tjunto(T5): 120 (tiempo ejecución de la tarea T6 en P1) + 165 (tiempo acumulado de la tarea T5 en P1) = 285

- Tsep(T5): de acuerdo a los cálculos se elige el procesador P0, entonces = 25 (tiempo T5) + 120 (tiempo de T6 en P0) – 15 (grado concurrencia entre T5 y T6 * tiempo de T5 en P1) + 0.02 (0.0001 * 100 + 0.01) = 110.02
- dif: Tsep – Tjunto = -174.98 ⇒ganancia = 0

✓ Ganancia de T7: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso se tienen tres adyacentes asignadas T3 T4 yT5

- Con respecto a T3
  - Tjunto: 110 (tiempo ejecución de la tarea T7 en P1) + 140 (tiempo acumulado de la tarea T3 en P1) = 250
  - Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 140 (tiempo T3) + 122 (tiempo de T7 en P2) – 85 (grado concurrencia entre T3 y T7 * tiempo de T7 en P2) + 0.23 (0.0002 * 1100 + 0.01) = 177.23
  - dif1: tsep – tjunto = -72.77

- Con respecto a T4
  - Tjunto: 110 (tiempo ejecución de la tarea T7 en P0) + 55 (tiempo acumulado de la tarea T4 en P0) = 165
  - Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 55 (tiempo T4) + 122 (tiempo de T7 en P2) – 20 (grado concurrencia entre T4 y T7* tiempo de T7 en P2) + 0.12 (0.0002 * 500 + 0.01) = 157.12
  - dif2: tsep – tjunto = -7.88

- Con respecto a T5
  - Tjunto: 110 (tiempo ejecución de la tarea T7 en P1) + 165 (tiempo acumulado de la tarea T5 en P1) = 275
  - Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 25 (tiempo T5) + 122 (tiempo de T7 en P2) – 25 (grado concurrencia entre T5 y T7* tiempo de T7 en P2) + 0.42 (0.0002 * 2000 + 0.02) = 122.42
  - dif: tsep – tjunto = -152.58

- Todas las diferencias son menores a 0 ⇒ganancia = 0.

Dado que la ganancia es 0 para ambas tareas se decide la ejecución de T_6 en el procesador P_1, la tabla 5.31 muestra dicha asignación:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_1 [5..25] ST_9 [30.62...65.62] ST_{10} [65.62...85.62] ST_2 [160.70..260.70] ST_{13} [281.53..401.53]	
P_1	ST_3 [5.51..30.51] ST_3 [30.51..45.51] ST_6 [45.51..60.51] ST_7 [60.51..160.51] ST_8 [160.51..185.51] ST_5 [261.51..271.51] ST_{11} [271.51..281.51] ST_{11} [281.51..296.51]	
P_2		

Tabla 5.31- Asignación de la tarea T_6 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 5.32 muestra el acumulado de cada tarea asignada:

Tarea	Tiempo Acumulado
T_0	5
T_1	120
T_2	75
T_3	140
T_4	55
T_5	165
T_6	120
T_7	

Tabla 5.32- Acumulado en cada tarea

Una vez que T_6 fue ubicada se debe recalcular la ganancia de T_7:

- ✓ Ganancia de T7: para cada adyacente (ya asignada) se calcula el "tiempo junto" y el "tiempo separado". En esta caso se tienen cuatro adyacentes asignadas T3 T4 ,T5 yT6
  - Con respecto a T3

- Tjunto: 110 (tiempo ejecución de la tarea T7 en P1) + 140 (tiempo acumulado de la tarea T3 en P1) = 250
- Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 140 (tiempo T3) + 122 (tiempo de T7 en P2) – 85 (grado concurrencia entre T3 y T7 * tiempo de T7 en P2) + 0.23 (0.0002 * 1100 + 0.01) = 177.23
- dif1: tsep – tjunto = -72.77

o Con respecto a T4
- Tjunto: 110 (tiempo ejecución de la tarea T7 en P0) + 55 (tiempo acumulado de la tarea T4 en P0) = 165
- Tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 55 (tiempo T4) + 110 (tiempo de T7 en P1) – 20 (grado concurrencia entre T4 y T7* tiempo de T7 en P1) + 0.06 (0.0001 * 500 + 0.01) = 145.06
- dif2: tsep – tjunto = -19.94

o Con respecto a T5
- Tjunto: 110 (tiempo ejecución de la tarea T7 en P1) + 165 (tiempo acumulado de la tarea T5 en P1) = 275
- Tsep: de acuerdo a los cálculos se elige el procesador P2, entonces = 25 (tiempo T5) + 122 (tiempo de T7 en P2) – 25 (grado concurrencia entre T5 y T7* tiempo de T7 en P2) + 0.41 (0.0002 * 2000 + 0.01) = 122.41
- dif: tsep – tjunto = -152.59

o Con respecto a T6
- Tjunto: 110 (tiempo ejecución de la tarea T7 en P0) + 120 (tiempo acumulado de la tarea T6 en P0) = 230
- Tsep: de acuerdo a los cálculos se elige el procesador P1, entonces = 120 (tiempo T6) + 110 (tiempo de T7 en P1) – 100 (grado concurrencia entre T6 y T7* tiempo de T7 en P1) + 0.26 (0.0001 * 2500 + 0.01) = 130.26
- Dif: tsep – tjunto= 99.74

- Todaslas diferencias son menores a 0 ⇒ganancia = 0.

Dado que la ganancia es 0 se asigna T_7 al procesador P_1. La tabla 5.33 muestra esta asignación:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_1 [5..25] ST_9 [30.62...65.62] ST_{10} [65.62...85.62] ST_2 [160.70..260.70] ST_{13} [281.53..401.53]	
P_1	ST_3 [5.51..30.51] ST_3 [30.51..45.51] ST_6 [45.51..60.51] ST_7 [60.51..160.51] ST_8 [160.51..185.51] ST_5 [261.51..271.51] ST_{11} [271.51..281.51] ST_{14} [296.51..376.51] ST_{15} [376.51..396.51] ST_{16} [401.79..411.79]	
P_2		

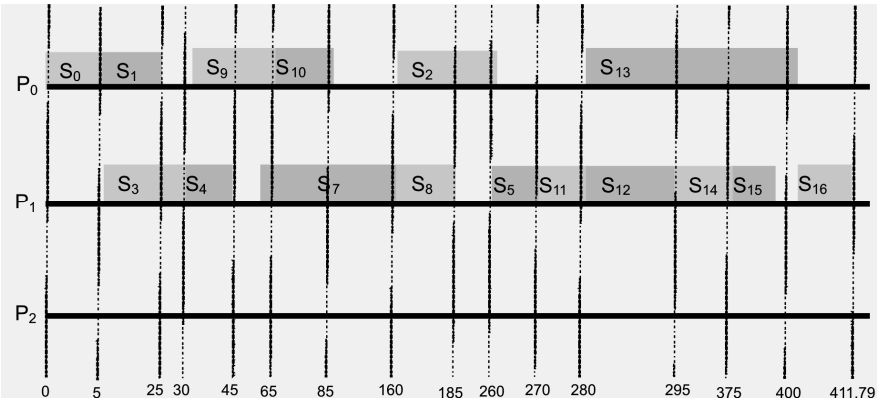
Tabla 5.33- Asignación de la tarea T_7 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 5.34 muestra el acumulado de cada tarea asignada:

Tarea	Tiempo Acumulado
T_0	5
T_1	120
T_2	75
T_3	140
T_4	55
T_5	165
T_6	120
T_7	110

Tabla 5.34- Acumulado en cada tarea

A continuación la figura 5.5 muestra de manera gráfica la ejecución de la aplicación:



Después de la implementación del algoritmo MATEHAIB y viendo la figura 5.5 se puede ver que en el procesador P_0 quedan asignadas las tareas T_0, T_1, T_4 y T_6, en el procesador P_1 quedaron asignadas T_2, T_3, T_5 y T_7. Además se pueden notar dos situaciones:

- El procesador P_2 no tiene ninguna tarea asignada.
- En el procesador P_0 se realizó un aprovechamiento de un hueco existente para asignar las subtareas T_9 yT_{10}. El aprovechamiento de huecos genera que:
  - La complejidad del algoritmo MATEHAIB es mayor a la del algoritmo MATEHA.
  - El tiempo final de la aplicación utilizando el algoritmo MATEHAIB es de 411.79, mientras que el obtenido utilizando el algoritmo MATEHA era de 418.16.

Resumen del capítulo

En este capítulo se presenta la definición de dos nuevos algoritmos de mapping (MATEHA y MATEHAIB) los cuales se aplican a grafos del modelo TTIGHA para determinar cómo deben asignarse las tareas a los procesadores tratando de minimizar el tiempo de ejecución del algoritmo en la arquitectura. Estos algoritmos están diseñados para poder funcionar en arquitecturas las cuales pueden ser heterogéneas tanto para los procesadores como las comunicaciones. Esta característica presenta una ventaja frente a otros algoritmos existentes en la actualidad como se ha analizado en capítulos anteriores.
Además para finalizar el capítulo se presenta un ejemplo para mostrar el funcionamiento de ambos algoritmos. Del ejemplo se puede ver que el algoritmo MATEHA obtiene peor tiempo de respuesta para la aplicación. Además se puede notar que el algoritmo MATEHAIB requiere menos tiempo (debido al aprovechamiento de huecos), logrando un mayor aprovechamiento de la arquitectura. Sin embargo, el costo del algoritmo MATEHAIB es mayor al de MATEHA.
El conjunto de pruebas completas realizadas a los algoritmos se describe en el capítulo VIII.

CAPÍTULO VI

Modelo MPAHA y algoritmo de mapping AMTHA

6.1 Introducción

En el capítulo II, se han descripto un conjunto de modelos los cuales intentan representar las características más salientes de las aplicaciones paralelas, junto a un conjunto de problemas que presentan cada uno de ellos (TIG, TPG, TTIG). Además en el capítulo IV se describió un nuevo modelo TTIGHA el cual soluciona las dificultades de los modelos anteriores. Este modelo permite representar aplicaciones paralelas con alto nivel de abstracción, y considerando la heterogeneidad de la arquitectura, tanto en lo que respecta a los procesadores como a las comunicaciones.
Basados en el modelo TTIGHA, en el capítulo V se definieron dos algoritmos de mapping MATEHA y MATEHAIB. De acuerdo al ejemplo mostrado en el capítulo V se vio que el algoritmo MATEHA al momento de elegir el procesador en el cual una tarea debe ejecutarse solo tiene en cuenta el tiempo acumulado en cada tarea sin considerar el instante en el cual cada una puede ser ejecutada. Este hecho es considerado por el algoritmo MATEHAIB ya que el mismo realiza un aprovechamiento de los huecos libres en cada procesador al momento de ejecutar una tarea; sin embargo la complejidad de este último algoritmo es mayor al de MATEHA.
Teniendo en cuenta las características mencionadas para los algoritmos anteriores se implementa un nuevo algoritmo de mapping llamado AMTHA (Automatic Mapping Task on Heterogeneous Architectures). Este nuevo algoritmo no utiliza toda la información incluida en el modelo TTIGHA y por lo tanto también se implementa un nuevo modelo llamado MPAHA (Model for Parallel Algorithms on Heterogeneous Architectures) en el cual se basa dicho algoritmo. Este nuevo modelo es una variante del modelo TTIGHA.

6.2 Definición del modelo MPAHA

El modelo MPAHA se basa en la construcción de un grafo G (V,E) donde:

- V, es el conjunto de nodos que representan cada una de las tareas T_i del programa paralelo.
- E, es el conjunto de aristas que representan cada una de las comunicaciones entre los nodos del grafo.

Detalle de los parámetros del modelo
En el primer parámetro del grafo (V) cada nodo representa una tarea T_i del programa paralelo. Cada una de las tareas T_i está compuesta por una o más subtareas indicando la relación de precedencia estricta entre ellas.
Dada la posibilidad de contar con una arquitectura heterogénea se deben tener en cuenta los tiempos de cómputo en cada uno de los tipos de procesadores que la componen. Es decir, el nodo *i* (V_i □ V) almacena el tiempo de cómputo en cada uno de los tipos diferentes de procesador para cada una de las subtareas que componen a la tarea T_i. Por lo tanto,

V_i (s,p) = tiempo de ejecución de la subtarea s en el tipo de procesador p.

En el segundo parámetro del grafo (E), las aristas representan las comunicaciones que existen entre cada par de tareas. En este conjunto una arista A entre dos tareas T_i y T_j contiene el volumen de comunicación (en bytes), la subtarea origen (□ T_i) y una subtarea destino (□ T_j). Es decir,

$E_{i,j}$ (o,d) = volumen de comunicación entre la subtarea origen (o □ T_i) y la subtarea destino (d □ T_j).

Es importante notar que dado a la heterogeneidad de la red de interconexión solo se mantiene el volumen de comunicación entre dos subtareas, en lugar del tiempo requerido para la misma.

6.3 Creación del modelo MPAHA

Al igual que en la generación del modelo TTIGHA para generar el modelo MPAHA se utilizó la técnica de obtener la estimación de los parámetros del modelo por medio del análisis del archivo de trazas, con lo cual, los pasos aplicados para obtener la estimación de los parámetros del modelo son:

1. Ejecución de la aplicación para generar el archivo de trazas en cada uno de los tipos de procesadores.
2. Análisis del archivo de trazas para la generación de datos intermedios.
3. Generación del modelo MPAHA a partir de los datos obtenidos en el paso anterior.

Los pasos 1 y 2 se llevan a cabo de la misma manera que para el modelo TTIGHA, por lo tanto después de la ejecución de los mismos se obtienen los valores de M, Ts y Cs. A continuación se describe el paso 3 en el cual se genera el modelo MPAHA.

Generación del modelo MPAHA

Luego de que se han obtenidos los valores antes mencionados se procede a construir el grafo G que representará la aplicación en el modelo MPAHA. Esta construcción se realiza en dos etapas, la primera es la creación del conjunto de nodos V, mientras que la segunda es la formación del conjunto de aristas E.

- Creación de V: para cada nodo que representa una tarea T_i, se obtienen las subtareas que la componen y el orden de precedencia en que deben ejecutarse. Para esto debe tenerse en cuenta el tiempo de ejecución de cada subtarea en cada uno de los diferentes tipos de procesadores.
- Creación de E: se agrega una nueva arista entre la tarea T_i y la tarea T_j por cada comunicación que existe entre dos subtareas S_o y S_d pertenecientes a T_i y T_i respectivamente.

Al igual que en capítulo IV se mostrará un ejemplo para construir el modelo MPAHA considerando la misma aplicación que se mostró en dicho capítulo.

6.3.1. Ejemplo de la generación del modelo MPAHA

Para la mejor comprensión del proceso de generación del grafo G, se desarrolla un ejemplo, el cual muestra el funcionamiento en cada paso de la creación de dicho grafo.

Para el ejemplo se describe una aplicación paralela y la arquitectura sobre la cual será ejecutada.

La aplicación está formada por 17 “funciones” ($ST_0..ST_{16}$). Cada una de estas funciones realiza una tarea con datos propios o datos resultantes (en caso de ser necesario) de otra “función”.

En la figura 6.1 se muestra las funciones de la aplicación y la interacción que existe entre las mismas.

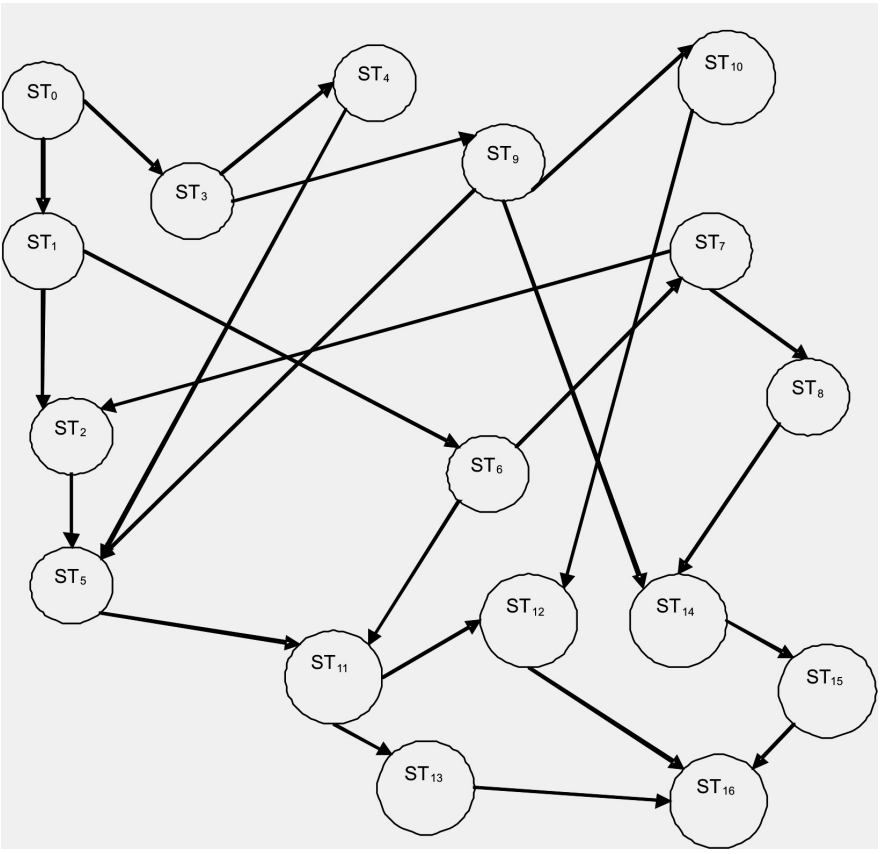


Figura 6.1 Subtareas de la aplicación

Supongamos que del análisis del comportamiento de las funciones y la interacción entre las mismas se tiene como resultado el siguiente agrupamiento de funciones:

- ST_0
- ST_1 y ST_2
- ST_3, ST_4 y ST_5
- ST_6, ST_7 y ST_8
- ST_9 y ST_{10}
- ST_{11} y ST_{12}

Con estos, se obtiene, el nuevo gráfico de la aplicación que se muestra en la figura 6.2. La figura no incluye los volúmenes de comunicación entre subtareas para que el gráfico tenga mayor claridad.

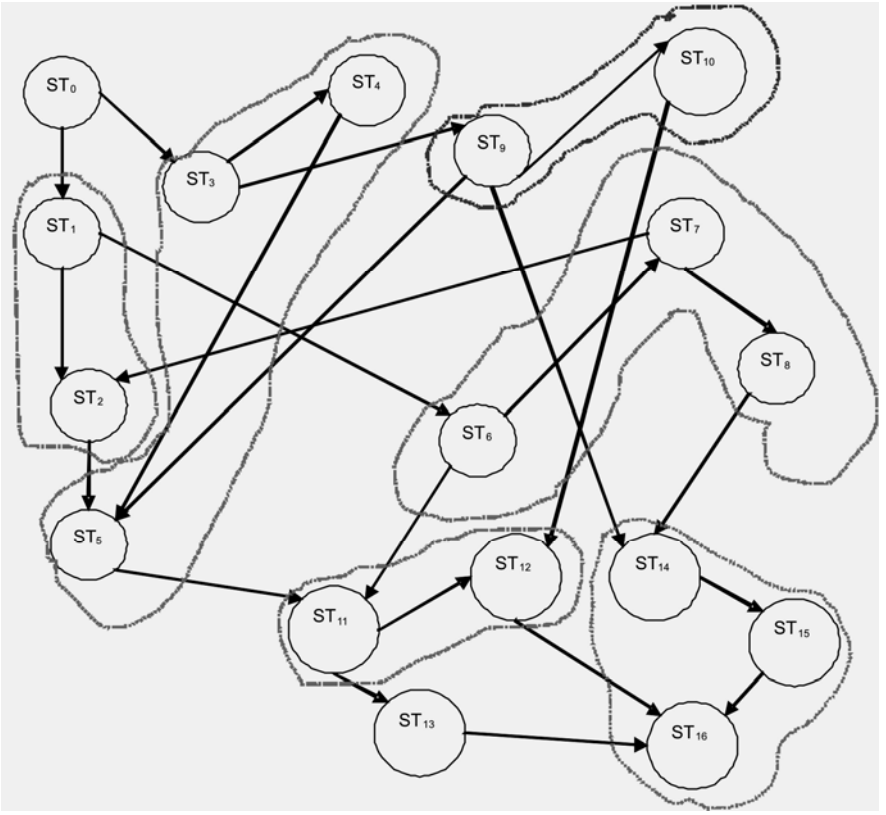


Figura 6.2 Agrupamiento de las funciones de la aplicación

Las tablas 6.1 y 6.2 muestran los tiempos de ejecución en cada tipo de procesador y los volúmenes de comunicación de las subtareas que las componen respectivamente.

Subtarea (M)	Tiempo Ejecución en el tipo de procesador 0 (Ts)	Tiempo Ejecución en el tipo de procesador 1 (Ts)
ST_0	5	7
ST_1	20	35
ST_2	100	145
ST_3	25	40
ST_4	15	25
ST_5	10	15
ST_6	15	20
ST_7	100	115
ST_8	25	40
ST_9	35	60
ST_{10}	20	35
ST_{11}	10	15
ST_{12}	15	20
ST_{13}	120	160
ST_{14}	80	85
ST_{15}	20	25
ST_{16}	10	12

Tabla 6.1 – Tiempos de cómputo de cada función

	ST_0	ST_1	ST_2	ST_3	ST_4	ST_5	ST_6	ST_7	ST_8	ST_9	ST_{10}	ST_{11}	ST_{12}	ST_{13}	ST_{14}	ST_{15}	ST_{16}
ST_0	0	10.000	0	5.000	0	0	0	0	0	0	0	0	0	0	0	0	0
ST_1	0	0	100000	0	0	0	1.500	0	0	0	0	0	0	0	0	0	0
ST_2	0	0	0	0	0	8.000	0	0	0	0	0	0	0	0	0	0	0
ST_3	0	0	0	0	10000	0	0	0	0	1.000	0	0	0	0	0	0	0
ST_4	0	0	0	0	0	50000	0	0	0	0	0	0	0	0	0	0	0
ST_5	0	0	0	0	0	0	0	0	0	0	0	6.700	0	0	0	0	0
ST_6	0	0	0	0	0	0	0	15000	0	0	0	1.500	0	0	0	0	0
ST_7	0	0	1.800	0	0	0	0	0	300000	0	0	0	0	0	0	0	0
ST_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.100	0
ST_9	0	0	0	0	0	0	0	0	0	0	20000	0	0	0	500	0	0
ST_{10}	0	0	0	0	0	0	0	0	0	0	0	0	2.000	0	0	0	0
ST_{11}	0	0	0	0	0	0	0	0	0	0	0	0	10000	100	0	0	0
ST_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
ST_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2500
ST_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18000	0
ST_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35000
ST_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 6.2 – Comunicación entre subtareas

Cada una de las agrupaciones consideradas anteriormente forma lo que se denomina tarea de la aplicación, formada por cada una de las funciones que la integran. Por lo tanto, ahora la aplicación estará formada por un conjunto de tareas las cuales interactúan entre sí, y a su vez cada tarea puede estar formada por un conjunto de subtareas relacionadas.
En nuestro ejemplo, la aplicación estará formada por 8 tareas.
Considerando esta distribución el grafo G de la aplicación se obtiene por medio de los siguientes pasos:

1. cada tarea de la aplicación forma un nodo del grafo. Cada uno de estos nodos está compuesto por todas las subtareas que forman la tarea que representa ordenadas por su relación de precedencia dentro del nodo. Además en cada nodo se almacena el tiempo de ejecución de cada subtarea en cada uno de los tipos diferentes de procesadores que componen la arquitectura en donde va a ser ejecutada la aplicación.
2. Cada comunicación existente entre un par de subtareas es representada como una arista en el grafo desde el nodo que incluye a la subtarea origen hacia el nodo que incluye la subtarea destino. Esta arista mantiene el volumen de comunicación entre los nodos.

Después de que se realizan los dos pasos anteriores el gráfico de la aplicación queda como se muestra en la figura 6.3

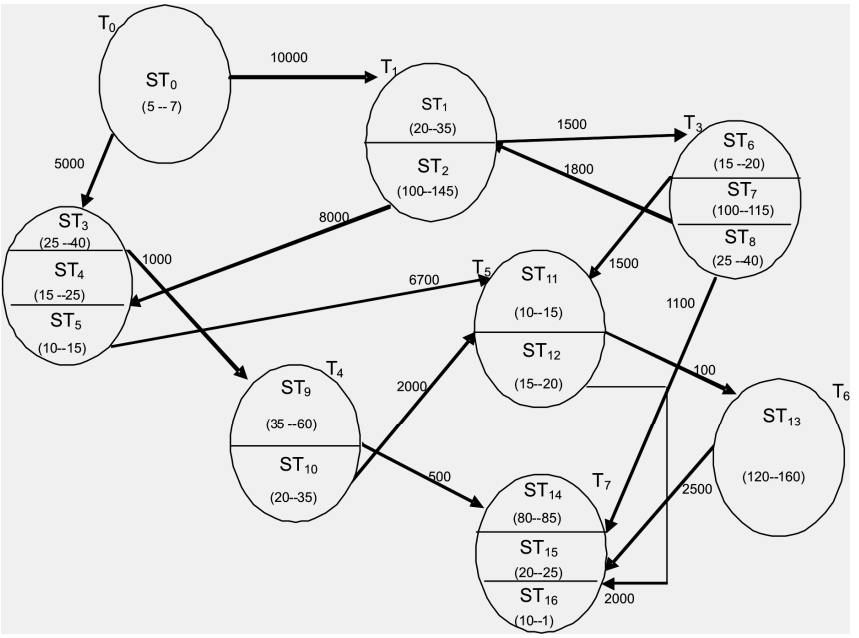


Figura 6.3 – Modelo MPAHA

Suponiendo que la aplicación será ejecutada en una arquitectura compuesta por 2 tipos diferentes de procesadores en cada nodo se almacena la información referente a los tiempos de ejecución de las subtareas en cada uno de los 2 tipos de procesadores.

6.4 Algoritmo de mapping AMTHA

El algoritmo AMTHA se aplica al modelo MPAHA. El mismo permite determinar la asignación de tareas a los procesadores de la arquitectura a utilizar, buscando minimizar los tiempos de ejecución de la aplicación en dicha arquitectura. Además este algoritmo debe indicar el orden en el cual deben ejecutarse las subtareas (que componen las tareas) asignadas en cada procesador.
AMTHA considera una arquitectura con un número acotado de procesadores heterogéneos en cuanto a su potencia de cálculo. Con respecto a la red de interconexión el algoritmo también considera que pueda ser heterogénea en cuanto al ancho de banda y a la velocidad de transmisión.

6.4.1 Descripción del algoritmo AMTHA

El algoritmo AMTHA utiliza los valores del grafo G generado por el modelo MPAHA; estos valores son el tiempo de cómputo de una subtarea en cada tipo de procesador, volumen de comunicación con sus adyacentes y la relación de pertenencia de subtareas a tareas.
El algoritmo AMTHA asigna una tarea por vez, hasta que todas han sido asignadas. El siguiente pseudocódigo describe los pasos fundamentales de este algoritmo:

Calcular el *rank* de cada tarea.

Mientras (no se asignaron todas las tareas)

Seleccionar la próxima tarea t a asignar.

Elegir el procesador p al cual debe asignarse la tarea t.

Asignar la tarea t (elegida en el paso 2) al procesador p (elegido en el paso 3).

Actualizar el rank de las tareas involucradas.

Al finalizar el algoritmo todas las tareas han sido asignadas a algún procesador y además se ha determinado el orden en que las subtareas que componen las tareas asignadas al procesador deben ser ejecutadas dentro del mismo.
A continuación se describen cada uno de estos tres pasos mencionados para desarrollar el algoritmo AMTHA.

6.4.1.1 Cálculo del *rank* de una tarea

Dado un grafo G, el rank de una tarea Rk(T) se define como la suma de los tiempos promedios de las subtareas que la componen y que se encuentran listas para su ejecución (todas sus predecesoras ya han sido asignadas y ubicadas en un procesador). La fórmula 6.1 expresa la definición antes mencionada:

$$Rk\ (T) = \sum_{i \in L(T)} W_{prom}\ (St_i) \qquad \text{Fórmula 6.1}$$

donde:
L(T) es el conjunto de subtareas listas para la tarea T.
W_{prom} (St) es el tiempo promedio de la subtarea St. El tiempo promedio se calcula como se muestra en la fórmula 6.2

$$W_{prom}\ (ST_i) = \frac{\sum_{p \in P} V_{st_i}\ (tipo\ de\ procesador\ de\ p)}{\#\ p} \qquad \text{Fórmula 6.2}$$

donde:
P es el conjunto de procesadores que componen la arquitectura.
#p es la cardinalidad de p

6.4.1.2 Selección de la tarea a ejecutar

Una vez obtenido el rank de cada tarea que compone la aplicación se elige la tarea de máximo rank. En caso que ocurriese que dos o más tareas poseen el mismo valor máximo el algoritmo rompe este empate eligiendo aquella tarea que minimice el promedio del tiempo total de ejecución de la tarea. La fórmula 6.3 muestra este cálculo:

$$Tprom\ (T) = \sum_{i \in T} W_{prom}(St_i)$$ Fórmula 6.3

6.4.1.3 Elección del procesador

La selección del procesador consiste en elegir aquel procesador, perteneciente a la arquitectura, que minimice el tiempo de ejecución al asignar la tarea elegida al procesador.
Para poder comprender cómo se calcula el tiempo de un procesador p, debe tenerse en cuenta que cada procesador mantiene una lista de subtareas que ya le fueron asignadas LU_p y que pueden ejecutarse (todas sus predecesoras ya fueron ubicadas) y otra lista que contiene aquellas subtareas que fueron asignadas a p pero aún no pueden ejecutarse LNU_p (alguna de sus predecesoras aún no ha sido ubicada).
Por lo tanto para calcular el procesador p que será elegido se tiene en cuenta dos casos:

1. todas las subtareas de la tarea t pueden ser ubicadas en p (es decir todas sus predecesoras han sido ubicadas).
2. Alguna de las subtareas de t no puede ser ubicada en p (esto ocurre cuando alguna predecesora de alguna subtarea de t no ha sido ubicada).

Para el primer caso el tiempo Tp del procesador p, está dado por el instante en p en el cual termina la ejecución de la última subtarea de t. En cambio, para el segundo caso, el tiempo Tp del procesador p, está dado por el tiempo en que la última subtarea de LU_p finalizará, más la suma de los tiempos de ejecución en p para cada una de las subtareas de LNU_p.

6.4.1.4 Asignación de la tarea elegida al procesador elegido

Al asignar una tarea t a un procesador p, cada subtarea St_k perteneciente a t se intenta ubicar en el procesador en un instante de tiempo en el cual todas las subtareas adyacentes a ella han finalizado (inclusive su predecesora dentro de t en caso de que posea). Esto puede ser en un intervalo libre entre dos subtareas ya ubicadas en p, o bien en un intervalo al final de estas. Si la subtarea St_k no puede ser ubicada se agrega a la lista LNU de p. Cada vez que una subtarea St_k es agregada a la lista LU de algún procesador, se intenta ubicar todas sus subtareas predecesoras que pertenezcan a tareas ya asignadas.

6.4.1.5 Actualización del valor de rank en las tareas involucradas

Como primera acción en este paso se le asigna -1 como valor de rank a la tarea t que fue asignada al procesador p. Esto se realiza para que la tarea t no vuelva a ser elegida para asignarse.
Además en este paso se considera la siguiente situación: para cada subtarea St_k, ubicada en el paso 6.4.1.4 se analiza la necesidad de actualizar el rank de las tareas a las cuales pertenecen las sucesoras St_{succ} de St_k, es decir, si St_{succ} tiene todas sus predecesoras ubicadas entonces el rank de la tarea a la cual pertenece St_{succ} se actualiza incrementándolo con $W_{prom}(St_{succ})$.

6.5. Ejemplo del algoritmo de mapping AMTHA

Basado en el grafo generado por el modelo MPAHA se ilustra el funcionamiento del algoritmo AMTHA. En este caso el grafo G está formado por 8 tareas (T_0..T_7) como se ve en la figura 6.3.
En la figura 6.4 se visualiza cómo es la arquitectura. La misma está formada por dos tipos diferentes de procesadores con dos procesadores correspondientes al primer tipo (P_0 y P_1) y el segundo conformado por el procesador P_2. Además la red de interconexión está formada por dos grupos de comunicación, uno que conecta a los dos procesadores del mismo grupo (P_0 y P_1); y otra para conectar con el procesador del grupo restante (P_2).

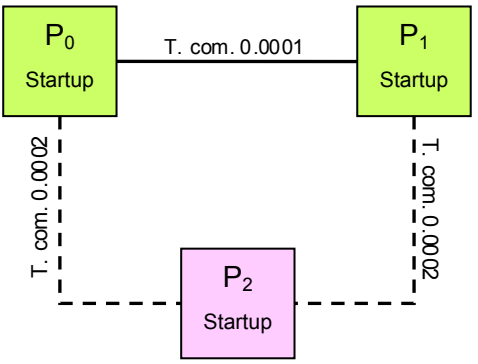


Figura 6.4- Esquema de la arquitectura

Teniendo en cuenta los datos descriptos anteriormente comienza la ejecución del algoritmo.
El primer paso de este algoritmo es calcular el rank de cada una de las tareas que componen la aplicación. La tabla 6.4 muestra este valor calculado:

Tarea	Rank
T_0	5.7
T_1	0
T_2	0
T_3	0
T_4	0
T_5	0
T_6	0
T_7	0

Tabla 6.4- Rank de cada tarea

Luego, se elige la tarea de máximo valor de rank. La tarea T_0 es elegida para comenzar la ejecución.

Asignación de la tarea T_0
La tarea T_0 es asignada al procesador que minimice su tiempo de ejecución, es decir, P_0.
La tabla 6.5 visualiza como se va actualizando la asignación de las tareas, en cada procesador. A su vez, se muestra los intervalos en los cuales se asigna cada tarea:

Procesador	LU	LNU
P_0	ST_0 [0..5]	
P_1		
P_2		

Tabla 6.5- Asignación de la tarea T_0 a un procesador, manteniendo las listas de c/ procesador

Esta asignación genera que el rank de las tareas T_0 ,T_1 y T_2 sea actualizado. La tabla 6.6 muestra los valores de rank actualizados para cada tarea:

Tarea	Rank
T_0	-1
T_1	25
T_2	48.3
T_3	0
T_4	0
T_5	0
T_6	0
T_7	0

Tabla 6.6- Rank de cada tarea

Asignación de la tarea T_2

La tarea T_2 es asignada al procesador que minimice su tiempo de ejecución:

- Tiempo de ejecución en P_0 = 45(LU) + 10(LNU) = 55.
- Tiempo de ejecución en P_1 = 45.51(LU) + 10(LNU) = 55.51
- Tiempo de ejecución en P_2 = 71.01(LU) + 15(LNU) = 86.01

Se elige el procesador P_0.

La tabla 6.7 muestra cómo se va actualizando la asignación de las tareas, en cada procesador. A su vez, se muestra los intervalos en los cuales se asigna cada tarea:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_3 [5..30] ST_4 [30..45]	ST_5
P_1		
P_2		

Tabla 6.7- Asignación de la tarea T_2 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 6.9 presenta los valores de rank actualizados después de asignar ST_2, y ST_4.

Tarea	Rank
T_0	-1
T_1	25
T_2	-1
T_3	0
T_4	68.3
T_5	0
T_6	0
T_7	0

Tabla 6.9- Rank de cada tarea

Asignación de la tarea T_4

La tarea T_4 es asignada al procesador que minimice su tiempo de ejecución:

- Tiempo de ejecución en P_0 = 100(LU) + 0(LNU) = 100.
- Tiempo de ejecución en P_1 = 85.11 (LU) + 0(LNU) = 85.11
- Tiempo de ejecución en P_2 = 95.21(LU) + 0(LNU) = 95.21

Se elige el procesador P_1.

En la tabla 6.10 se observa cómo se va actualizando la asignación de las tareas, en cada procesador. A su vez, se muestra los intervalos en los cuales se asigna cada tarea:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_3 [5..30] ST_4 [30..45]	ST_5
P_1	ST_9 [30.11..65.11] ST_{10} [65.11..85.11]	
P_2		

Tabla 6.10- Asignación de la tarea T_4 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 6.11 muestra los valores de rank actualizados después de asignar ST_9, y ST_{10}.

Tarea	Rank
T_0	-1
T_1	25
T_2	-1
T_3	0
T_4	-1
T_5	0
T_6	0
T_7	81.6

Tabla 6.11- Rank de cada tarea

Asignación de la tarea T_7

La tarea T_7 es asignada al procesador que minimice su tiempo de ejecución:

- Tiempo de ejecución en P_0 = 145.17(LU) + 40(LNU) = 185.17.
- Tiempo de ejecución en P_1 = 165.11 (LU) + 30(LNU) = 195.11
- Tiempo de ejecución en P_2 = 150.22(LU) + 37(LNU) = 187.22

Se elige el procesador P_0.

La tabla 6.12 muestra cómo se va actualizando la asignación de las tareas, en cada procesador. A su vez, se muestra los intervalos en los cuales se asigna cada tarea:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_3 [5..30] ST_4 [30..45] ST_{14} [65.17..145.17]	ST_5 ST_{15} ST_{16}
P_1	ST_9 [30.11..65.11] ST_{10} [65.11..85.11]	
P_2		

Tabla 6.12- Asignación de la tarea T_7 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 6.13 muestra los valores de rank actualizados después de asignar ST_{14}.

Tarea	Rank
T_0	-1
T_1	25
T_2	-1
T_3	0
T_4	-1
T_5	0
T_6	0
T_7	-1

Tabla 6.13- Rank de cada tarea

Asignación de la tarea T_1

La tarea T_1 es asignada al procesador que minimice su tiempo de ejecución:

- Tiempo de ejecución en P_0 = 145.17(LU) + 140(LNU) = 285.17.
- Tiempo de ejecución en P_1 = 85.11 (LU) + 100(LNU) = 185.11
- Tiempo de ejecución en P_2 = 42.01(LU) + 145(LNU) = 187.01

Se elige el procesador P_1.

En la tabla 6.14 se muestra cómo se va actualizando la asignación de las tareas, en cada procesador. A su vez, se muestra los intervalos en los cuales se asigna cada tarea:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_3 [5..30] ST_4 [30..45] ST_{14} [65.17..145.17]	ST_5 ST_{15} ST_{16}
P_1	ST_1 [6.01..26.01] ST_9 [30.11..65.11] ST_{10} [65.11..85.11]	ST_2
P_2		

Tabla 6.14- Asignación de la tarea T_1 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 6.15 muestra los valores de rank actualizados después de asignar ST_1.

Tarea	Rank
T_0	-1
T_1	-1
T_2	-1
T_3	151.6
T_4	-1
T_5	0
T_6	0
T_7	-1

Tabla 6.15- Rank de cada tarea

Asignación de la tarea T_3

La tarea T_3 es asignada al procesador que minimice su tiempo de ejecución:

- Tiempo de ejecución en P_0 = 270(LU) + 0(LNU) = 270.
- Tiempo de ejecución en P_1 = 225.11 (LU) + 0(LNU) = 225.11
- Tiempo de ejecución en P_2 = 201.32(LU) + 0(LNU) = 201.32

Se elige el procesador P_2.

La tabla 6.16 visualiza como se va actualizando la asignación de las tareas, en cada procesador. A su vez, se muestra los intervalos en los cuales se asigna cada tarea:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_3 [5..30] ST_4 [30..45] ST_{14} [65.17..145.17]	ST_5 ST_{15} ST_{16}
P_1	ST_1 [6.01..26.01] ST_9 [30.11..65.11] ST_{10} [65.11..85.11]	ST_2
P_2	ST_6 [26.32..46.32] ST_7 [46.32..161.32] ST_8 [161.32..201.32]	

Tabla 6.16- Asignación de la tarea T_3 a un procesador, manteniendo las listas de c/ procesador

Al realizar la ubicación de ST_7 y ST_8 se permite la ubicación de ST_2, ST_5 y ST_{15}. La tabla 6.17 muestra dicha ubicación.

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_3 [5..30] ST_4 [30..45] ST_{14} [65.17..145.17] ST_{15} [201.56..221.56] ST_5 [262.51..272.51]	ST_{16}
P_1	ST_1 [6.01..26.01] ST_9 [30.11..65.11] ST_{10} [65.11..85.11] ST_2 [161.7..261.7]	
P_2	ST_6 [26.32..46.32] ST_7 [46.32..161.32] ST_8 [161.32..201.32]	

Tabla 6.17- Asignación de la tarea T_3 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 6.18 muestra los valores de rank actualizados después de asignar ST_2., ST_5.y ST_{15}.

Tarea	Rank
T_0	-1
T_1	-1
T_2	-1
T_3	-1
T_4	-1
T_5	28.2
T_6	0
T_7	-1

Tabla 6.18- Rank de cada tarea

Asignación de la tarea T_5

La tarea T_5 es asignada al procesador que minimice su tiempo de ejecución:

- Tiempo de ejecución en P_0 = 297.51(LU) + 0(LNU) = 270.
- Tiempo de ejecución en P_1 = 298.19 (LU) + 0(LNU) = 298.19
- Tiempo de ejecución en P_2 = 308.86 (LU) + 0(LNU) = 308.86

Se elige el procesador P_0.

La tabla 6.19 visualiza como se va actualizando la asignación de las tareas, en cada procesador. A su vez, se muestra los intervalos en los cuales se asigna cada tarea:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_3 [5..30] ST_4 [30..45] ST_{14} [65.17..145.17] ST_{15} [201.56..221.56] ST_5 [262.51..272.51] ST_{11} [272.51..282.51] ST_{12} [272.51..297.51]	ST_{16}
P_1	ST_1 [6.01..26.01] ST_9 [30.11..65.11] ST_{10} [65.11..85.11] ST_2 [161.7..261.7]	
P_2	ST_6 [26.32..46.32] ST_7 [46.32..161.32] ST_8 [161.32..201.32]	

Tabla 6.19- Asignación de la tarea T_5 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 6.20 muestra los valores de rank actualizados después de asignar ST_{11}. y ST_{12}.

Tarea	Rank
T_0	-1
T_1	-1
T_2	-1
T_3	-1
T_4	-1
T_5	-1
T_6	133.3
T_7	-1

Tabla 6.20- Rank de cada tarea

Asignación de la tarea T_6

La tarea T_6 es asignada al procesador que minimice su tiempo de ejecución:

- Tiempo de ejecución en P_0 = 417.51(LU) + 0(LNU) = 417.51.
- Tiempo de ejecución en P_1 = 402.53 (LU) + 0(LNU) = 402.53
- Tiempo de ejecución en P_2 = 442.54 (LU) + 0(LNU) = 442.54

Se elige el procesador P_1.

La tabla 6.21 visualiza como se va actualizando la asignación de las tareas, en cada procesador. A su vez, se muestra los intervalos en los cuales se asigna cada tarea:

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_3 [5..30] ST_4 [30..45] ST_{14} [65.17..145.17] ST_{15} [201.56..221.56] ST_5 [262.51..272.51] ST_{11} [272.51..282.51] ST_{12} [272.51..297.51]	ST_{16}
P_1	ST_1 [6.01..26.01] ST_9 [30.11..65.11] ST_{10} [65.11..85.11] ST_2 [161.7..261.7] ST_{13} [282.53..402.53]	
P_2	ST_6 [26.32..46.32] ST_7 [46.32..161.32] ST_8 [161.32..201.32]	

Tabla 6.21- Asignación de la tarea T_6 a un procesador, manteniendo las listas de c/ procesador

Al realizar la ubicación de ST_{13} se permite la ubicación de ST_{16}. La tabla 6.22 muestra dicha ubicación.

Procesador	LU	LNU
P_0	ST_0 [0..5] ST_3 [5..30] ST_4 [30..45] ST_{14} [65.17..145.17] ST_{15} [201.56..221.56] ST_5 [262.51..272.51] ST_{11} [272.51..282.51] ST_{12} [272.51..297.51] ST_{16} [402.79..412.79]	
P_1	ST_1 [6.01..26.01] ST_9 [30.11..65.11] ST_{10} [65.11..85.11] ST_2 [161.7..261.7] ST_{13} [282.53..402.53]	
P_2	ST_6 [26.32..46.32] ST_7 [46.32..161.32] ST_8 [161.32..201.32]	

Tabla 6.22- Asignación de la tarea T_6 a un procesador, manteniendo las listas de c/ procesador

Además la tabla 6.23 muestra los valores de rank actualizados después de asignar ST_{13}.

Tarea	Rank
T_0	-1
T_1	-1
T_2	-1
T_3	-1
T_4	-1
T_5	-1
T_6	-1
T_7	-1

Tabla 6.23- Rank de cada tarea

Dado que todas las tareas han sido asignadas el algoritmo de mapping termina, quedando asignadas las tareas T_0, T_2, T_5 y T_7 al procesador P_0; T_1, T_4, y T_6 al procesador P_1; y por último T_3 al procesador P_2.
El tiempo final de la aplicación generado por el algoritmo de mapping es 412.79.

A continuación la figura 6.5 muestra de manera gráfica la ejecución de la aplicación:

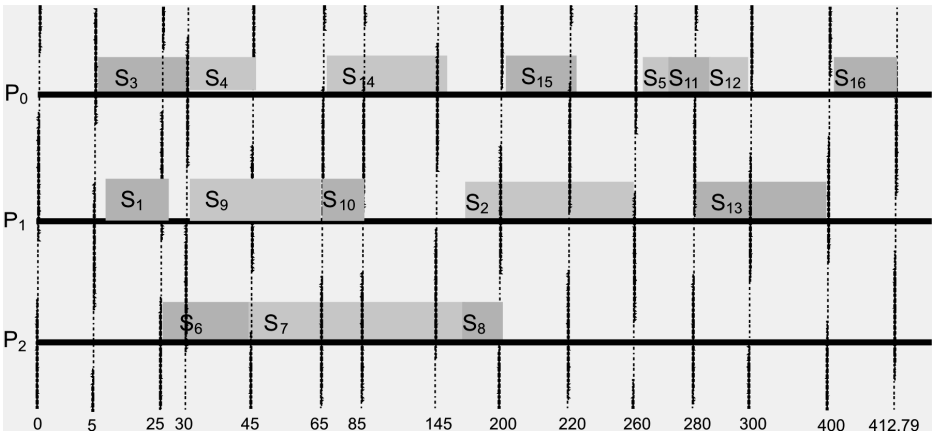


Figura 6.5 Simulación de la ejecución de la aplicación

Resumen del capítulo

En este capítulo se presenta la definición de un nuevo modelo llamado MPAHA y un algoritmo de mapping AMTHA el cual se basa en el modelo propuesto.
Tanto el modelo como el algoritmo de mapping están diseñados para poder funcionar en arquitecturas las cuales pueden ser heterogéneas tanto para los procesadores como las comunicaciones.
Además en este capítulo se presentan un ejemplo que muestra cómo se genera, a partir de una aplicación dada, el grafo G por medio del modelo MPAHA. Luego a partir de este grafo se muestra por medio de un ejemplo el funcionamiento del algoritmo de mapping AMTHA.
Un aspecto importante a considerar en cuanto a los algoritmos de mapping MATEHAIB y AMTHA es que el primer algoritmo tiene mayor complejidad debido a que:

- cada vez que debe elegir la tarea a asignar debe recalcular la ganancia para todas las tareas aun no asignadas pertenecientes al nivel que está procesando. En cambio el algoritmo AMTHA calcula el rank de cada tarea una única vez y luego lo va actualizando.
- cada vez que debe calcular la ganancia de una tarea realiza una simulación para obtener los tiempos de asignar la tarea a cada uno de los procesadores de la arquitectura. En cambio, el algoritmo AMTHA realiza esta simulación una única vez para cada tarea en el momento que ha sido elegida para ser asignada. Esta es un ventaja importante del algoritmo AMTHA ya que el tiempo que requiere la simulación es muy significativo respecto al tiempo del algoritmo de mapping, ya que debe "ubicar" a las subtareas que componen una tarea en el instante de tiempo más conveniente dentro del procesador.

CAPÍTULO VII

Resultados obtenidos

7.1 Introducción

En este capítulo se describe el conjunto de pruebas realizadas para analizar el comportamiento de los algoritmos de mapping MATEHA, MATEHAIB, AMTHA desarrollados en los capítulos V y VI de esta Tesis. En este análisis los algoritmos son comparados con el algoritmo MATE, utilizado en arquitecturas homogéneas, y el algoritmo HEFT el cual es uno de los algoritmos más utilizados para el mapping en arquitecturas heterogéneas por su eficacia y simplicidad.
Previamente a realizar las comparaciones entre los algoritmos se detalla el conjunto de pruebas realizadas para las mismas.

7.2 Descripción de las pruebas

Para analizar el comportamiento de los algoritmos propuestos se han generado un conjunto de aplicaciones. Las mismas varían en términos de cantidad de tareas que las componen; cantidad de subtareas que forman las tareas; tamaño de las subtareas y el volumen de comunicación entre las subtareas. En todas las aplicaciones, el tiempo de cómputo supera al de comunicaciones (aplicaciones de grano grueso).
Además en cada una de las pruebas realizadas debe ser indicada previamente la configuraciónde la arquitectura a utilizar. Por lo tanto se debe especificar la cantidad de tipos diferentes de procesadores, la cantidad de máquinas de cada tipo, y las características de comunicación entre los procesadores (tiempo de startup de cada uno, y el tiempo de transferencia entre cada par de ellos).
Con cada una de estas pruebas creadas (aplicación y configuración de la arquitectura), se generaron los grafos correspondientes a los modelos TTIGHA y MPAHA respectivamente, y a partir de estos grafos se realizó el mapping por medio de los algoritmos MATEHA, MATEHAIB y AMTHA. Al mismo tiempo, se realizó el mapping con los dos algoritmos conocidos MATE (para arquitecturas homogéneas) y HEFT (para arquitecturas heterogéneas).

7.2.1 Selección del conjunto de pruebas a evaluar

En esta tesis las pruebas elegidas se clasificaron en 32 grupos. Cada grupo posee un conjunto de características de la aplicación en sí misma y de la arquitectura utilizada. A su vez, cada grupo está constituido por 10 pruebas donde cada una de ellas varía dentro de los parámetros del grupo. El detalle de cada uno de estos grupos se encuentra en la tabla 7.1.

Grupo	# Tareas	Máx. Nro de Sub. por Tarea	Tiempo subtareas (seg)	Tipos de Procesador	CantProc
1	25	10	1000-7500	3	3 - 6
2	25	10	100-750	3	3 - 6
3	25	3	1000-7500	3	3 - 6
4	25	3	100-750	3	3 - 6
5	10	10	1000-7500	3	3 - 6
6	10	10	100-750	3	3 - 6
7	10	3	1000-7500	3	3 - 6
8	10	3	100-750	3	3 - 6
9	25	10	1000-7500	5	10-20
10	25	10	100-750	5	10-20
11	25	3	1000-7500	5	10-20
12	25	3	100-750	5	10-20
13	10	10	1000-7500	5	10-20
14	10	10	100-750	5	10-20
15	10	3	1000-7500	5	10-20
16	10	3	100-750	5	10-20
17	75	10	1500-6000	3	3 - 6
18	75	10	150-600	3	3 - 6
19	75	4	1500-6000	3	3 - 6
20	75	4	150-600	3	3 - 6
21	50	10	1500-6000	3	3 - 6
22	50	10	150-600	3	3 - 6
23	50	4	1500-6000	3	3 - 6
24	50	4	150-600	3	3 - 6
25	75	10	1500-6000	5	10-20
26	75	10	150-600	5	10-20
27	75	4	1500-6000	5	10-20
28	75	4	150-600	5	10-20
29	50	10	1500-6000	5	10-20
30	50	10	150-600	5	10-20
31	50	4	1500-6000	5	10-20
32	50	4	150-600	5	10-20

Tabla 7.1 Detalle de los grupos de aplicaciones para las pruebas

7.3 Experimentación realizada

La experimentación realizada se aplica a cada una de las pruebas descriptas en la sección 7.2.1. El conjunto completo de los resultados de cada una de las pruebas se encuentra en el Anexo I.

7.3.1 Análisis de comportamiento de MATEHA

En esta sección se estudia el comportamiento del algoritmo MATEHA descripto en esta Tesis. Este estudio consiste en analizar los resultados obtenidos en relación al tiempo final de cada una de las aplicaciones, involucradas en cada prueba, de acuerdo al mapping generado por el algoritmo MATEHA y al mapping generado por MATE y HEFT. La primera comparación se realiza entre los algoritmos MATEHA y MATE. La figura 7.1 muestra el porcentaje de variación del MATEHA respecto al algoritmo MATE. Este porcentaje se encuentra expresado en la fórmula 7.1:

$$\Delta = \frac{(tiempoMATE - tiempoMATE\,HA)}{tiempoMATE} * 100 \qquad \text{Fórmula 7.1}$$

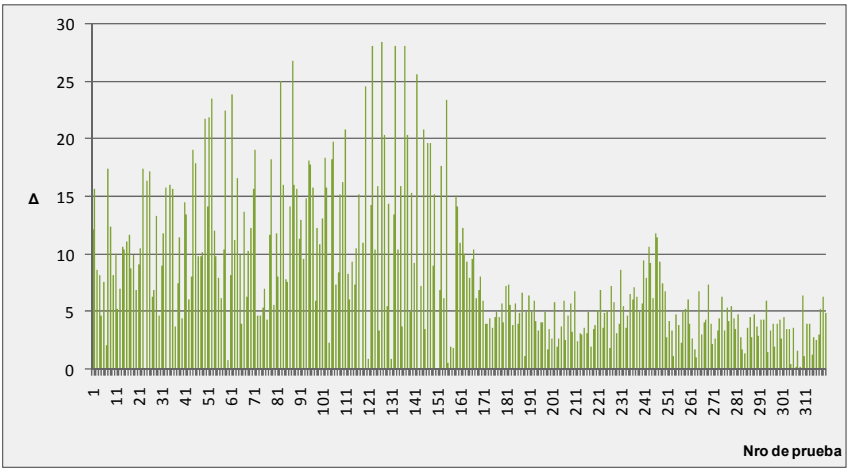


Figura 7.1 Mejora del algoritmo MATEHA con respecto a MATE

De la figura 7.1 se ve que el algoritmo de mapping MATEHA mejora para todas las pruebas los resultados obtenidos por el algoritmo de

mapping MATE. Este hecho era esperable debido que el algoritmo MATE fue diseñado para arquitecturas homogéneas.
La figura 7.2 muestra el porcentaje de mejora entre MATEHA y HEFT. La fórmula 7.2 muestra cómo se calcula este porcentaje.

$$\Delta = \frac{(tiempoHEFT - tiempoMATE\,HA)}{tiempoHEFT} * 100 \qquad \text{Fórmula 7.2}$$

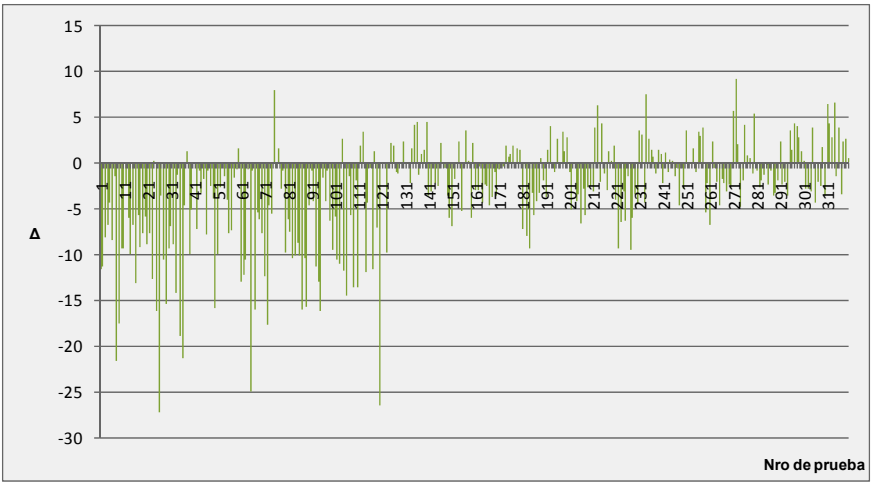


Figura 7.2 Mejora del algoritmo MATEHA con respecto al HEFT

Observando la figura 7.2 se puede ver que el algoritmo de mapping MATEHA obtiene algunas mejoras con respecto al HEFT en aplicaciones de gran tamaño y en arquitecturas con gran cantidad de procesadores.

7.3.2 Análisis de comportamiento de MATEHAIB

En esta sección se estudia el comportamiento del algoritmo MATEHAIB descripto en esta tesis. Este estudio consiste en analizar los resultados obtenidos en relación al tiempo final de cada una de las aplicaciones, involucradas en cada prueba, de acuerdo al mapping generado por el algoritmo MATEHAIB y al mapping generado por MATE y HEFT. La primera comparación se realiza entre los algoritmos MATEHAIB y

MATE. La figura 7.3 muestra el porcentaje de variación del MATEHAIB respecto al algoritmo MATE. Este porcentaje se encuentra expresado en la fórmula 7.3.

$$\Delta = \frac{(tiempoMATE - tiempoMATE\,HAIB)}{tiempoMATE} * 100 \qquad \text{Fórmula 7.3}$$

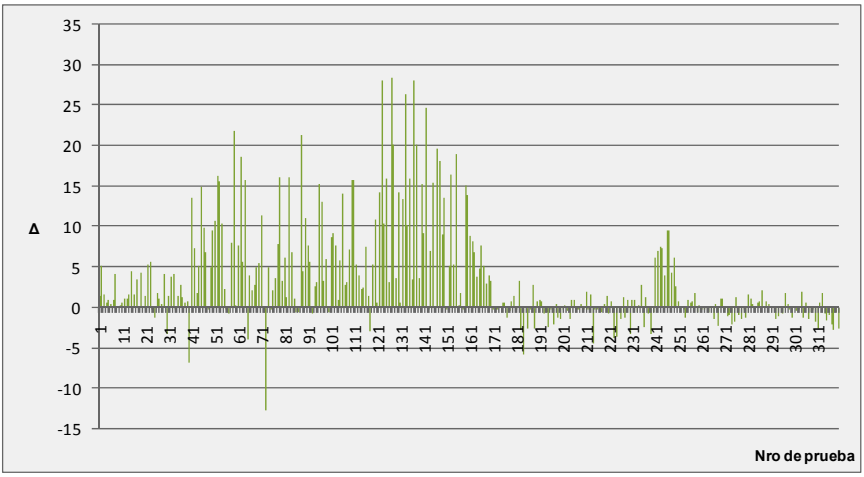


Figura 7.3 Mejora del algoritmo MATEHAIB con respecto a MATE

De la figura anterior se puede notar que el algoritmo de mapping MATEHAIB mejora en la mayoría de las pruebas los resultados obtenidos por el algoritmo MATE.
La figura 7.4 muestra el porcentaje de mejora entre MATEHAIB y HEFT. La fórmula 7.4 muestra cómo se calcula este porcentaje.

$$\Delta = \frac{(tiempoHEFT - tiempoMATE\,HAIB)}{tiempoHEFT} * 100 \qquad \text{Fórmula 7.4}$$

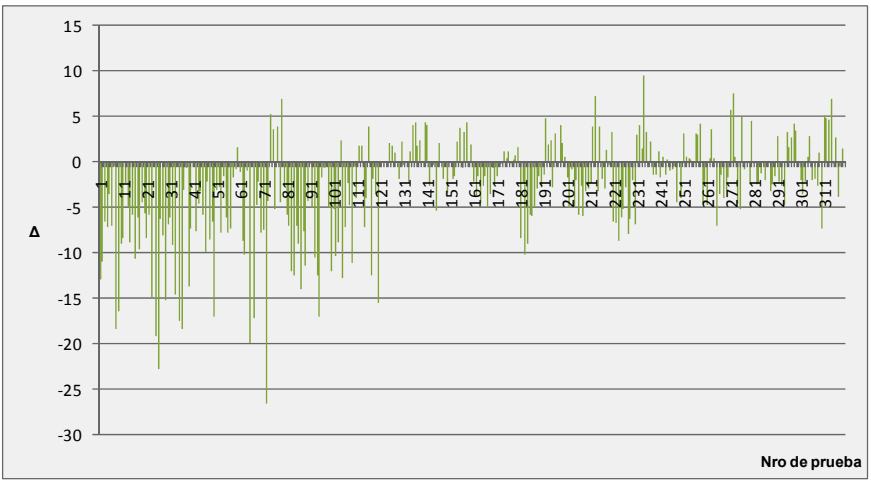


Figura 7.4 Mejora del algoritmo MATEHAIB con respecto al HEFT

La figura 7.4 muestra que el algoritmo MATEHHAIB obtiene en general peores resultados a los obtenidos por HEFT, sobre todo para aquellas aplicaciones con poca cantidad de tareas y arquitecturas compuestas por pocos procesadores.

7.3.3 Análisis de comportamiento de AMTHA

En esta sección se estudia el comportamiento del algoritmo AMTHA descripto en esta Tesis. Este estudio consiste en analizar los resultados obtenidos en relación al tiempo final de cada una de las aplicaciones, involucradas en cada prueba, de acuerdo al mapping generado por el algoritmo AMTHA y al mapping generado por MATE y HEFT. La primera comparación se realiza entre los algoritmos AMTHA y MATE. La figura 7.5 muestra el porcentaje de variación del AMTHA respecto al algoritmo MATE. Este porcentaje se encuentra expresado en la fórmula 7.5.

$$\Delta = \frac{(tiempoMATE - tiempoAMTHA)}{tiempoMATE} * 100 \qquad \text{Fórmula 7.5}$$

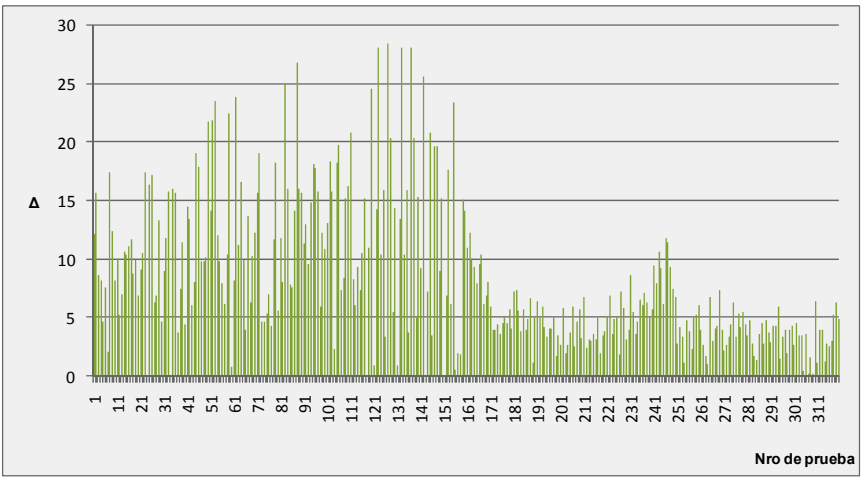


Figura 7. 5 Mejora del algoritmo AMTHA con respecto al MATE

De la figura 7.5 se ve que el algoritmo de mapping AMTHA al igual que el algoritmo MATEHA mejora, para todas las pruebas, los resultados obtenidos por el algoritmo de mapping MATE.
La figura 7.6 muestra el porcentaje entre AMTHA y HEFT. La fórmula 7.6 muestra cómo se calcula este porcentaje.

$$\Delta = \frac{(tiempoHEFT - tiempoAMTHA)}{tiempoHEFT} * 100$$

Fórmula 7.6

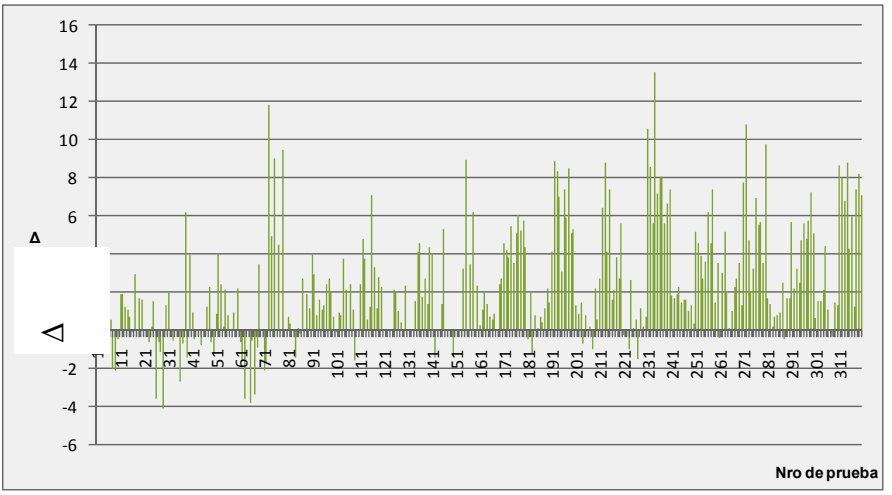


Figura 7.6 Mejora del algoritmo AMTHA con respecto al HEFT

En la figura 7.6 se puede notar que el algoritmo de mapping AMTHA mejora, para la mayoría de las pruebas, los resultados obtenidos por el algoritmo de mapping HEFT. Esta mejora es casi total en aquellas pruebas donde las aplicaciones están compuestas por gran cantidad de tareas y la arquitectura utilizada está formada por muchos procesadores.

7.3.4 Comparación entre los tres algoritmos

En esta sección se realiza una comparación entre los tres algoritmos presentados en esta tesis, en los capítulos V y VI (MATEHA, MATEHAIB y AMTHA). Para esto se hace una comparación de los resultados de los algoritmos mencionados frente al resultado obtenido por el algoritmo HEFT.
La figura 7.7 muestra el porcentaje de variación promedio por grupo de cada uno de los tres algoritmos MATEHA, MATEHAIB y AMTHA, respecto al HEFT. Estos porcentajes fueron obtenidos utilizando las fórmulas 7.2, 7.4 y 7.6 respectivamente.

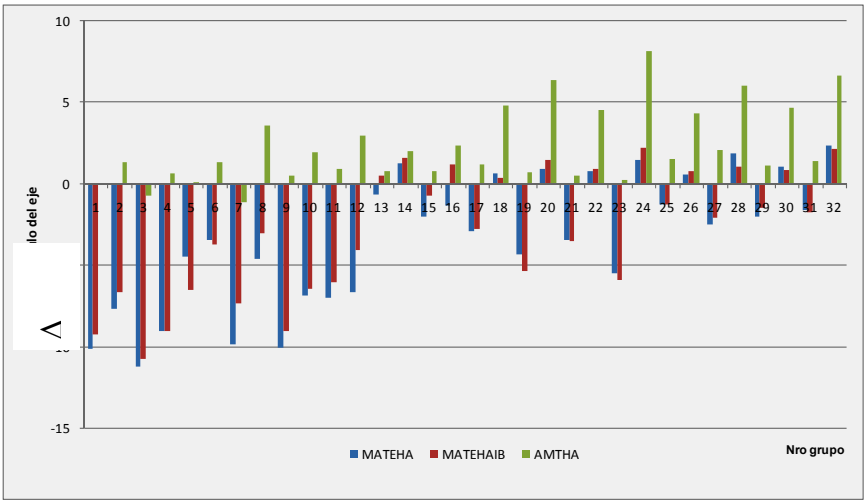


Figura 7.7 Comparación entre los algoritmos MATEHA, MATEHAIB y AMTHA con respecto al HEFT

En la figura 7.7 se puede ver que los algoritmos MATEHA y MATEHAIB mejoran los resultados obtenidos por el HEFT en 9 de los 32 grupos de pruebas.

Por su parte, el algoritmo AMTHA obtiene mejores resultados en 28 grupos de pruebas a los resultados obtenidos por el HEFT, en dos grupos obtiene los mismos resultados al HEFT y solamente para 2 grupos genera peores resultados que los obtenidos por HEFT.
Un aspecto importante a tener en cuenta es que el algoritmo AMTHA mejora en todos los casos los resultados obtenidos por los algoritmos MATEHA y MATEHAIB.

Resumen del capítulo

En este capítulo se presentaron las pruebas realizadas en esta tesis. Estas pruebas incluyen las comparaciones entre los resultados obtenidos por los algoritmos existentes como el MATE y el HEFT y los propuestos en este trabajo (MATEHA, MATEHAIB y AMTHA).
De los resultados se puede ver que el algoritmo AMTHA mejora en todas las pruebas a MATEHA y MATEHAIB. Además de estos resultados hay que tener en cuenta que la complejidad del algoritmo AMTHA es menor a la de los otros dos algoritmos propuestos.
A su vez, también se puede notar que el algoritmo AMTHA mejora o iguala al HEFT en el 94% de los grupos de pruebas. Desde el punto de vista de las pruebas realizadas sobre un total de 320 pruebas en 273 de ellas (85%) obtiene mejores o iguales resultados que el algoritmo HEFT.

CAPÍTULO VIII

Conclusiones

8.1. Descripción de las conclusiones

El objetivo de esta tesis ha sido definir un modelo que permita representar el comportamiento de aplicaciones paralelas con patrones de interacción arbitraria sobre arquitecturas heterogéneas, junto con un algoritmo de mapping que permita encontrar una asignación óptima de las tareas que componen una aplicación a los procesadores que forman la arquitectura sobre la cual ejecuta la misma. Dicho algoritmo de mapping debía basarse en un modelo de arquitectura que extraiga las características más importantes de la misma, por ejemplo, cantidad de procesadores que la componen, tipo de comunicaciones, tiempos de cómputo y comunicación, etcétera.

La tesis puede verse dividida en cinco etapas: la primera es una descripción de las características de las diferentes arquitecturas existentes: cluster, cluster homogéneos, cluster heterogéneos, multicluster y grid. Una vez estudiadas estas arquitecturas se pudo ver el gran uso en la actualidad del cluster heterogéneo.

La segunda etapa de la tesis, se centró en el estudio y análisis de ventajas y desventajas de los modelos de representación y predicción de performance existentes, entre los cuales podemos mencionar TIG, TPG y TTIG. Con respecto a este punto se encontró que el modelo TTIG, si bien soluciona numerosos problemas que tienen los modelos TIG y TPG, no puede ser aplicado en arquitecturas heterogéneas. Este hecho implica una valla en su utilización debido a que la arquitectura más utilizada actualmente es el cluster heterogéneos.

Por lo tanto, se definieron dos nuevos modelos TTIGHA y MPAHA, que permiten modelar y predecir la performance de aplicaciones paralelas que se ejecutan sobre arquitecturas heterogéneas. Para esto, ambos modelos extraen las características principales de la arquitectura e incluye nuevos parámetros para tener en cuenta la heterogeneidad de la misma.

La tercera etapa de la tesis, estuvo dedicada al desarrollo de tres nuevos algoritmos de mapping, dos de ellos MATEHA y MATEHAIB basados en el modelo TTIGHA; y un tercer algoritmo AMTHA que estaba basado en el modelo MPAHA.

En la cuarta etapa, se detalla el conjunto de pruebas (aplicaciones) utilizadas para le evaluación de los algoritmos propuestos. Las pruebas esta-

ban agrupadas dependiendo sus características, formándose 32 grupos con diez pruebas cada uno, es decir 320 aplicaciones diferentes en total.
A su vez, esta etapa puede subdividirse en dos partes. En la primera se evaluó y comparó los tiempos finales de la aplicación al utilizar la asignación obtenida por los algoritmos de mapping desarrollados en la Tesis frente a los obtenidos por dos de los algoritmos más utilizados en la actualidad MATE y HEFT.
De esta etapa, se obtuvo como resultado que los tres algoritmos de mapping propuestos MATEHA, MATEHAIB y AMTHA mejoraron los resultados obtenidos por el algoritmo MATE en más de un 98%. Con respecto a los resultados obtenidos en la comparación frente al HEFT se puede ver que el algoritmo MATEHA obtiene mejores resultados que el HEFT solo en aplicaciones de gran tamaño; mientras tanto el algoritmo MATEHAIB genera mejores resultados que el HEFT en un 50% de los casos; y por último se puede observar que los resultados del algoritmo AMTHA son mejores al HEFT en un 83% de las pruebas.
En la segunda parte se realizó una comparación entre los tres algoritmos MATEHA, MATEHAIB y AMTHA. Esta comparación midió la diferencia de los resultados de cada algoritmo frente a los obtenidos por el algoritmo más utilizado hasta el momento (HEFT). De esta comparación se vieron dos cosas: primero que el algoritmo AMTHA obtuvo mejores resultados en todos los casos a los obtenidos por los otros dos algoritmos propuestos. Segundo el algoritmo AMTHA obtuvo en 28 grupos de pruebas mejores resultados que el HEFT, solo en dos casos generó los mismos resultados y en los otros dos casos obtuvo peores resultados.

8.2 Contribuciones de este trabajo de tesis

Se definieron los modelos TTIGHA y MPAHA que permiten representar el comportamiento de aplicaciones paralelas con patrones de interacción arbitraria sobre arquitecturas heterogéneas. Estos modelos son utilizados por los algoritmos de mapping para realizar la asignación de los procesos que componen la aplicación a los diferentes procesadores que forman la arquitectura de manera de obtener óptimos tiempos de respuesta para la aplicación.
Se desarrollaron tres algoritmos de mapping MATEHA, MATEHAIB y AMTHA, y se evaluó la performance de estos algoritmos frente a dos algoritmos ya existentes y muy utilizados como MATE y HEFT. De esta evaluación se vio que el algoritmo AMTHA basado en el mo-

delo MATHA mejora en un 100% los resultados obtenidos por MATE y en un 85% los obtenidos por el algoritmo HEFT.

8.3. Líneas futuras

Se continuará el estudio del algoritmo de mapping AMTHA para tratar de obtener una optimización del speedup y balance de carga alcanzables.
Analizar y estudiar las modificaciones necesarias al algoritmo AMTHA para determinar la cantidad óptima de procesadores, necesarios para ejecutar la aplicación, de manera automática; y a partir de este dato, se logre una asignación que sin aumentar el tiempo final de la aplicación, aumentando la eficiencia de la misma.
Analizar y estudiar los cambios necesarios en el algoritmo AMTHA y en el modelo MATHA para poder ser ejecutados sobre arquitecturas GRID.

Bibliografía

[1] Tinetti F., De Giusti A. (1998). *Procesamiento Paralelo. Conceptos de Arquitecturas y Algoritmos*. La Plata: Editorial Exacta.

[2] Grama A., Gupta A., Karpyis G., Kumar V. (2002). *Introduction to Parallel Computing*. *2nd Edition*. Boston: Pearson – Addison Wesley.

[3] Naiouf M. (2004). *Procesamiento Paralelo. Balance de Carga Dinámico en Algoritmos de Sorting*. Tesis Doctoral. Argentina: Facultad de Ciencias Exactas. UNLP.

[4] Pancake Cherri M. (1996). "Is Parallelism for You?". *IEEE Computational Science and Engineering, 3*(2), 18-37.

[5] Flynn M.J. (1972). "Some Computer Organizations and Their Effectiveness". *IEEE Transactions on Computers, 21* (9), 948-960.

[6] Grama A., Gupta A., Karypis G., Kumar V. (2003). *An Introduction to Parallel Computing. Design and Analysis of Algorithms, 2nd Editio.,* Pearson Addison Wesley.

[7] Akl S., (1997). *Parallel Computation. Models and Methods*. Prentice-Hall, Inc.

[8] De Giusti A., Naiouf M., De Giusti L., Chichizola F., Rodríguez I., Pettoruti J., Pousa A., Ardenghi J., Bertogna L., Printista M. (2007). "1st Iberian Grid Infrastructure Conference Proceedings. IBERGRID". *Parallel Algorithm on Multi-Cluster Architectures using GRID Middleware. Experiences in Argentine Universities*. (pp. 322-332). España.

[9] Grid Computing and Distributed Systems (GRIDS) Laboratory - Department of Computer Science and Software Engineering (University of Melbourne). (2007). *Cluster and Grid Computing*. <www.cs.mu.oz.au/678>

[10] Joseph J., Fellenstein C. (2003). *Grid Computing. On Demand Series*. IBM Press.

[11] Berman F., Fox G., Hey A. (2003). *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley & Sons.

[12] Abbas A. (2005). *Grid Computing. A Practical Guide to Technology and Applications*. Firewall Media.

[13] Zoltan J., Kacsuk P., Kranzlmuller D., (2004). *Distributed and Parallel Systems: Cluster and Grid Computing. The International Series in Engineering and Computer Science.* Springer.

[14] Minoli D. (2004). *A Networking Approach to Grid Computing.* Wiley-Interscience.

[15] Kumar V., Gupta A. (1994). "Analyzing Scalability of Parallel Algorithms and Architectures". *Journal of Parallel and Distributed Computing, 22*(1), 60-79.
[16] Leopold C., (2001). *Parallel and Distributed Computing. A Survey of Models, Paradigms and Approaches.* Wiley.
[17] Andrews G. (2000). *Foundations of Multithreaded, Parallel and Distributed Programing.* Addison Wesley Higher Education.
[18] Roig Concepción. (2005). *Algoritmos de asignación basados en un nuevo modelo de representación de programas paralelos*. Tesis Doctoral. España: Universidad Autónoma de Barcelona.
[19] Chaudhary V. Aggarwal J. (1993). "A generalized scheme for Mapping Parallel Algorithms". *IEEE Transactions on Parallel and Distributed Systems, 4*(3), 328-346.
[20] Pelagatti S. (1998). *Structured Development of Parallel Programs.* USA-Londres: Taylor & Francis.
[21] Roig C., Ripoll A., Senar M.A., Guirado F., Luque E. (1999). "Modelling Message Passing Programs for Static Mapping". *In Euromicro Worshop on Parallel and Distributed Processing (PDP'00),* (pp. 229-236), USA: IEEE CS Press.
[22] Senar M. A., Ripoll A., Cortes A. Roig C. Luque E. (1998). "Clustering and Reassignmentbase Mapping Strategy for Message-Passing Architectures". *In International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP'98)*, (pp. 415-421). USA: IEEE CS Press.
[23] Hwang J., Chow Y., Anger F., Lee C. (1989). "Scheduling Precedence Graphs in Systems with Interprocecessor Communication Times". *SIAM Journal of Computing, 18*(2), 244-257.
[24] Karipys G., Schlogel K., Kumar V. (1997). *PARMETIS: Parallel graph partitioning and sparce matrix ordering library*. Reporte Técnico del Departamento de Ciencias de la Computación de la Universidad de Minessota.
[25] Lam l. and Suen C.Y. (1995). "An Evaluation of Parallel Thinning Algorithms for Character recognition". *IEEE Transactions on Pattern Recognition and Machine Intelligence*, *17*(9), 914-919.
[26] Censor Y. Gordon D. and Gordon R. (2001). "Component Averaging: An Efficient Iterative Parallel Algorithm for Large Scale and Sparce Unstructured Problems". *Parallel Computing*, *27,* 777-808.
[27] Ahmmad I. He Y. and Liou M.L. (2002). "Video Compression with Parallel Processing". *Parallel Computing, 25,* 1039-1078.
[28] Bishop B. Kelliher T.P. and Irwin M.J, (1999). "A Detailed Analysis of Mediabench". *In Procceding on IEEE Workshop Signal Processing Systems (SiPS'99)*, (pp. 449-455).

[29] Floyd R. (1962). "Algorithim 97: Shortest path". *Communications of the ACM, 5*, 345.
[30] Yu-Kwong Kwok and Ishfaq Ahmad. (1996). "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors". *IEEE Transactions on Parallel and Distributed Systems, 7*(5), 506-521.
[31] Al-Mouhamed M. (1990). "Lower Bound on the Number of Processors and Time for Scheduling Precedence Graphs with Communications Cost". *IEEE Transactions on Software Engineering, 16* (12), 1390-1410.
[32] Fernandez E., Busseli B. (1973). "Bounds on the Number of Processors and Time for Multiprocessors Optimal Schedules". *IEEE Transactions on Computers, 22*(8), 745-751.
[33] Gerasoulis A. and Yang T. (1992). "A Comparison of Clustering Heuristics for Scheduling DAG on Multiprocessors". *Journal of Parallel and Distributed Computing, 16*(4), 276-291.
[34] Po-Jen C., Chih-Ming W. (2000). "AN Eficient Recognition-Complet Processor Allocation Strategy for k-ary n-cube Multiprocessors". *IEEE Transactions on Parallel and Distributed Systems, 11*(5), 485-490.
[35] Uppaluri S., Izadi B. and Radhakrishnan D. (2003). "Low-Power Dynamic Scheduling in Heterogeneous Systems". *In Proceedings of the International Conference on Embedded Systems and Applications, (ESA '03)* (pp. 261-267).
[36] Topcuoglu H., Wu M. (2002). "Performance- Effective and Low-Compexity Task Scheduling for Heterogeneous Computing". *IEEE Transactions on Parallel and Distributed Systems, 13*(3).
[37] Roig C. Ripoll A. Guirado F. (2007), "A New Task Graph Model for Mapping Message Passing Applications". *IEEE Transactions on Parallel and Distributed Systems, 18*(12).
[38] De Giusti L., Chichizola F, Naiouf M., De Giusti A. (2007). "Análisis de la Robustez del Método de Asignación MATEHa". *XIII Congreso Argentino de Ciencias de la Computación (CACIC 2007).* Argentina: Corrientes – Resistencia.
[39] Ali, S., Maciejewski, A.A., Siegel, H.J., Kim, J.-K. (2003). "Definition of a robustness metric for resource allocation". *In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS),* (pp. 42).
[40] England D. Weissman J., and Sadagopan J. (2005). "A New Metric for Robustness with Application to Job Scheduling". *IEEE International Symposium on High Performance Distributed Computing 2005 (HPDC-14)*, (pp. 24-27). Research Triangle Park, NC.

[41] De Giusti L. Chichizola F. (2007). *Reporte Técnico sobre el orden de ejecución de los procesos dentro de cada procesador*. III-LIDI.
[42] Foster, Kesselman. (2004). *The GRID 2. Blueprint for a new computing infrastructure.* Morgan Kauffman.
[43] Baker M., Buyya R., and Laforenza D. (2002). "Grids and grid Technologies for widearea distributed computing". *Software, Practice. Experience. 32*(15), 1437-1466.
[44] Foster and Iamnitchi. (2003). "On death, taxes, and the convergence of peer-to-peer and grid computing". *In International Workshop on Peer-to-Peer Systems (IPTPS), LNCS, 2*.
[45] Foster I. (2001). "The anatomy of the grid: Enabling scalable virtual organizations". *In CCGRID, IEEE Computer Society*, (pp. 6-7).

ANEXO I

Pruebas realizadas

1. Detalle de los resultados

Como se mencionó en el capítulo VII, se realizaron un conjunto de pruebas para evaluar el comportamiento de los algoritmos de mapping propuestos en esta tesis (MATEHA, MATEHAIB y AMTHA), por medio de la comparación de dos algoritmos conocidos (MATE, HEFT).
La tabla 1 muestra el tiempo final obtenido (en segundos) por el mapping producido por los cinco algoritmos, para cada una de las pruebas generadas.

Grupo	Prueba	HEFT	MATE	MATEHA	MATEHAIB	AMTHA
1	1	422009	483129	471528	476284	424073
	2	447707	523302	498309	496276	441214
	3	475574	515073	514468	506666	470167
	4	396774	427295	423962	424819	392292
	5	433075	452174	451943	447982	431063
	6	411763	442800	446818	440640	409051
	7	366812	372409	372111	368399	364737
	8	442089	545843	537722	523042	450806
	9	462065	538881	543624	537691	471745
	10	451837	494318	494024	492518	453886
2	1	45787	49929	50091	49620	44895
	2	42919	44431	43940	43908	42084
	3	39545	41978	41947	41472	39037
	4	53431	59121	58773	58154	52824
	5	47371	52508	50613	50133	47032
	6	37329	42018	42240	41287	37354
	7	40689	44759	43022	43183	39491
	8	45782	50187	50012	50119	45782
	9	37129	40514	39978	38763	36496
	10	51611	54580	54624	54538	50776

Grupo	Prueba	HEFT	MATE	MATEHA	MATEHAIB	AMTHA
3	1	196101	215705	213561	212413	196101
	2	147530	165007	158924	156094	147530
	3	142358	173528	160409	163575	143209
	4	142850	142735	142713	142653	142543

	5	150265	176894	174733	178957	147920
	6	135196	169047	172071	165863	140003
	7	146877	157855	152126	156090	147797
	8	158272	171997	175010	171098	160004
	9	157135	188759	181538	180925	163508
	10	159351	164965	174325	170215	157187
4	1	18928	20384	20243	20088	18555
	2	15090	17113	16429	16452	15090
	3	14468	17281	16518	16567	14543
	4	14270	14287	14465	14293	14270
	5	13269	15814	15779	15575	13269
	6	13283	16182	16125	15716	13634
	7	16626	17378	17409	17141	16734
	8	16951	17178	16735	17061	15899
	9	15650	17918	17216	17774	15867
	10	17542	17628	18448	18822	16838
5	1	229517	265782	229517	229517	227279
	2	210223	243894	225533	226035	211166
	3	205925	219324	207687	215282	205925
	4	186440	202859	192649	192265	186440
	5	242688	302152	247200	256735	244450
	6	174947	213206	188783	192153	174947
	7	153519	168188	154952	156673	151628
	8	141928	153869	145511	153869	138696
	9	191908	215062	199471	204409	193110
	10	142500	184455	165094	166669	144226
6	1	17975	20751	19768	18530	17817
	2	20091	24669	20629	20629	19274
	3	14676	18732	14676	15806	14322
	4	19199	21772	19494	19494	19155
	5	16539	17959	17209	17550	16185
	6	23352	25169	25169	25169	23170
	7	20081	21404	21567	21553	20081
	8	22528	24929	22906	22906	22314
	9	15634	20174	15747	15747	15634
	10	27431	27051	27024	26969	26816
7	1	45141	49450	51021	45647	45393
	2	55420	74019	62255	60206	56287
	3	70178	81896	77637	77252	72661
	4	58418	70071	58836	58993	58418
	5	43035	49613	53808	51568	44661
	6	62410	65366	62973	62722	62745
	7	40641	48638	47190	47570	41997
	8	73047	78705	76989	76460	73715
	9	53052	57057	56368	54176	51197
	10	53910	61469	58059	58059	53910

Grupo	Prueba	HEFT	MATE	MATEHA	MATEHAIB	AMTHA
8	1	6736	8164	7576	7232	6878
	2	5582	7021	6575	7059	5676
	3	6166	5702	6457	6424	5434
	4	6417	6396	6778	6076	6099
	5	5389	5181	4962	5190	4902
	6	6189	6660	6223	6511	6189
	7	7404	7386	7291	7109	7068
	8	7481	8478	7700	7809	7481
	9	6124	6789	6182	5697	5545
	10	7951	8424	8737	8139	7951
9	1	391981	441523	416172	414318	389014
	2	424110	460019	456549	453734	422592
	3	434052	579761	479541	486118	434052
	4	460063	555799	507095	517389	466530
	5	500062	541582	543900	535157	499174
	6	462581	501632	509521	503895	463388
	7	501961	568992	582617	571649	488183
	8	369514	505087	408142	397328	369514
	9	443398	517727	513033	494007	434871
	10	370029	433563	387253	385499	365521
10	1	44640	48379	45072	44656	42851
	2	41443	46200	43175	43530	40214
	3	47832	52471	53243	52848	47450
	4	38407	44405	43388	43206	37783
	5	42245	51028	49111	49424	41779
	6	42892	51487	43590	43590	42326
	7	38291	44371	39908	38534	37347
	8	49787	51543	50252	49855	48436
	9	42404	47414	45091	44538	41556
	10	52401	58358	57420	58665	52020
11	1	172939	199252	183206	181755	173006
	2	175772	213442	194511	193827	174060
	3	129803	153050	144096	141156	128747
	4	172510	169914	168011	168295	166028
	5	142879	171149	159710	161024	139802
	6	148166	184789	162309	158631	148166
	7	121342	127882	123260	124164	118381
	8	177328	191526	187650	185555	175396
	9	146021	174802	165987	162069	148249
	10	129549	154735	132117	130183	129549
12	1	15338	18922	16095	15944	14968
	2	15041	15618	14764	14770	14320
	3	16144	16534	15616	15855	15533
	4	14705	16139	16464	15761	14620

	5	14293	15245	14911	14854	14115
	6	13525	14042	13613	12991	12559
	7	15538	17728	17355	17475	15022
	8	11858	11721	12334	12068	11721
	9	18067	19747	19366	18686	17562
	10	14940	19357	18892	17247	14594

Grupo	Prueba	HEFT	MATE	MATEHA	MATEHAIB	AMTHA
13	1	162511	164074	162743	163041	162511
	2	196222	228861	196222	196222	196222
	3	180899	251536	198622	180899	180899
	4	113826	127107	113826	113826	113826
	5	165171	192224	161631	161631	161631
	6	181018	183555	177809	177809	177297
	7	107826	149249	108934	106713	106713
	8	178603	223385	180787	178233	177865
	9	224573	237591	224573	228655	224573
	10	137470	156775	134260	134260	134260
14	1	14311	14449	14332	14357	14311
	2	18317	21169	18317	18317	18317
	3	17519	24360	17910	17910	17519
	4	13016	14316	12816	12852	12816
	5	15105	17235	14479	14479	14479
	6	19180	18998	18335	18335	18293
	7	11309	15445	11469	11105	11105
	8	17552	21438	17391	17121	17074
	9	22771	23641	22451	22771	22451
	10	15074	17030	14414	14414	14414
15	1	70139	74169	72736	67282	67282
	2	60946	83107	62910	62489	61815
	3	74811	74811	74811	74811	74811
	4	74853	80716	76540	75088	74853
	5	58356	72704	59847	61419	57535
	6	58564	57481	57323	57355	55429
	7	71211	88684	71211	71211	71211
	8	72716	90552	73172	74076	72716
	9	60195	66168	62342	60195	60195
	10	53182	63606	56421	54967	53898
16	1	3646	3646	3902	3646	3646
	2	5340	5735	5441	5441	5340
	3	5867	7125	5877	5956	5864
	4	4052	4182	3958	3958	3920
	5	6011	7146	6328	5786	5474
	6	7469	7532	7502	7510	7486
	7	6910	6803	6670	6679	6670

	8	6086	5819	6079	5819	5708
	9	5969	7032	6328	5969	5969
	10	4709	5362	4612	4616	4599
17	1	1236194	1385978	1273248	1262914	1232879
	2	1170803	1321115	1207141	1213265	1158043
	3	1111928	1211813	1116406	1128043	1089517
	4	1273637	1384723	1329073	1331309	1255471
	5	1257639	1356697	1288094	1290612	1248221
	6	1286851	1414568	1319314	1306341	1278788
	7	1339275	1482690	1402520	1404824	1327855
	8	1217607	1298307	1264324	1259681	1217044
	9	1312263	1375014	1326323	1320451	1279836
	10	1224269	1295647	1268629	1252640	1190573

Grupo	Prueba	HEFT	MATE	MATEHA	MATEHAIB	AMTHA
18	1	121594	123284	122566	123370	115979
	2	113516	113198	113720	113487	108738
	3	130880	130969	131503	130988	125806
	4	128131	126788	125792	126618	121144
	5	142521	142712	141657	141885	137469
	6	126176	125534	125114	124741	119726
	7	119581	118307	117385	119627	112333
	8	110528	109800	110583	110143	104745
	9	130525	130466	128634	129464	122958
	10	130643	130339	128931	128418	124959
19	1	507971	550320	545283	550344	510300
	2	475042	502852	477099	485943	465474
	3	540450	579244	583458	595005	546878
	4	480092	495253	525102	523301	476260
	5	574447	608623	593457	607561	573485
	6	580353	599742	613580	615014	576172
	7	574568	601701	599269	602115	571819
	8	568977	602456	588139	585761	562081
	9	645210	638630	642160	655056	630941
	10	616938	639605	628801	634207	607665
20	1	48920	50113	50807	49596	46905
	2	53119	50939	52418	50560	48382
	3	59302	57746	56997	58168	54319
	4	51006	49539	51317	49805	47423
	5	60046	60251	60642	61657	58188
	6	62776	60580	61190	60738	58129
	7	60242	59103	60366	60260	56694
	8	60127	57924	58158	57643	55027
	9	65364	63139	64582	63932	62019
	10	60936	59808	59263	60594	57697

21	1	840035	852020	849215	854034	829061
	2	810330	853279	851941	850602	803130
	3	894681	898729	894255	901798	881132
	4	879681	910352	917057	922995	885784
	5	857381	883279	886683	874033	850135
	6	758253	809049	808630	801604	760590
	7	909908	932029	935892	933274	907862
	8	800852	848564	846428	848305	808349
	9	885538	918060	909946	913176	865802
	10	795628	817665	815975	818850	791129
22	1	84012	87631	86682	85870	81701
	2	95151	91324	91544	91362	89048
	3	91944	86587	86168	85215	83811
	4	73329	72433	74853	75653	70262
	5	77716	74709	74456	74686	71976
	6	92700	94228	93765	94428	91202
	7	76795	79123	79110	79045	75136
	8	90843	89137	89736	89559	87361
	9	90696	91442	90594	90955	88198
	10	78355	76912	76974	75777	73958

Grupo	Prueba	HEFT	MATE	MATEHA	MATEHAIB	AMTHA
23	1	373820	395362	389102	398397	374751
	2	333490	358889	364868	355958	333956
	3	390814	409087	415989	424492	394436
	4	359850	368565	371390	381887	350305
	5	342930	361135	364791	360533	342308
	6	367707	372774	373345	377979	365643
	7	373737	408817	409220	403319	379236
	8	320819	336818	340324	340835	317063
	9	369147	380596	382083	376586	368397
	10	319218	330201	333637	340932	316971
24	1	43146	42225	41640	41837	38582
	2	36378	35196	35287	34874	33257
	3	36429	35676	38023	35878	34376
	4	42400	38477	39256	38384	36651
	5	37653	37419	36682	36370	34949
	6	41203	40396	40623	41356	37919
	7	38130	37755	37911	37240	35039
	8	40201	40503	40677	40765	37923
	9	39828	39123	39295	40353	37159
	10	32094	31530	31815	31692	29713
25	1	1368457	1484301	1399805	1391974	1343547
	2	1400026	1496537	1385986	1391084	1376640
	3	1366493	1500671	1380351	1386068	1340390

	4	1203078	1295031	1199802	1198818	1175740
	5	1283773	1348719	1281008	1294794	1264558
	6	1422323	1586348	1443950	1433552	1398822
	7	1150947	1279357	1158281	1156374	1132222
	8	1245688	1359410	1303731	1300634	1232415
	9	1500462	1600843	1510036	1500262	1480607
	10	1323040	1413582	1360184	1374587	1318302
26	1	131851	128553	127332	127561	124975
	2	123632	123123	123548	122954	117998
	3	124598	123863	124659	124041	119688
	4	112878	111117	111184	112464	109792
	5	119120	120502	120397	119358	114760
	6	155284	151382	150202	150305	145584
	7	139873	136688	135905	135606	133449
	8	131786	128462	126801	126198	122003
	9	124193	129079	131000	129794	122336
	10	127987	131377	130969	131042	123439
27	1	556756	582316	594585	582974	558914
	2	596768	594256	583527	594461	578478
	3	467562	451363	470157	450722	443269
	4	541964	536611	553814	539270	530828
	5	542349	580977	567872	580009	541718
	6	447523	456990	455929	463104	443017
	7	542521	552819	554830	550234	530053
	8	485748	493731	501373	504947	472386
	9	543106	565938	556247	559778	523845
	10	512067	526412	526092	520503	505405

Grupo	Prueba	HEFT	MATE	MATEHA	MATEHAIB	AMTHA
28	1	63919	60267	60329	60272	58960
	2	52066	47692	47369	48160	46418
	3	53638	52885	52583	53302	51085
	4	54869	56236	57528	57385	53715
	5	59918	61936	61110	62979	57982
	6	51505	49602	49417	48938	47900
	7	50390	50315	50044	50775	47581
	8	54818	54004	54580	54738	51697
	9	54228	55355	54866	55503	52295
	10	60036	56660	56884	57299	54169
29	1	957371	975211	966504	958343	940821
	2	922992	955862	947969	944909	910100
	3	926803	952143	945343	948213	924951
	4	901955	911883	914088	912817	895568
	5	794610	799694	797363	793774	788076
	6	854462	877503	875393	870872	846084

	7	867517	885890	874884	866145	845494
	8	860787	889473	892702	887724	864427
	9	796665	822554	823508	815418	783354
	10	769779	786158	784452	781832	756484
30	1	90669	88102	88677	87997	85497
	2	82925	84800	84988	84629	81079
	3	85391	86364	87214	87607	82598
	4	86814	90027	89987	90909	84608
	5	96537	93412	93130	93336	91976
	6	80972	79108	79882	79622	76390
	7	78385	77720	75057	76285	74619
	8	81957	78845	78755	78463	77230
	9	85770	82785	83480	82748	79529
	10	88919	88201	87801	89229	84404
31	1	381621	389830	381219	389044	379126
	2	380634	392745	391430	394161	374676
	3	350020	357033	360158	357635	344613
	4	368271	373758	379821	366292	360422
	5	356170	342012	342849	345961	340257
	6	361071	370643	376666	368398	357074
	7	340771	342254	348112	346724	341265
	8	335682	342451	344364	344512	336654
	9	370327	365593	364064	366488	364810
	10	299502	315999	310260	321500	295467
32	1	35666	32963	33396	33822	32570
	2	41908	40149	40145	39899	38570
	3	41779	40563	40664	39848	38941
	4	47433	43830	44356	44127	43251
	5	38544	37955	39143	38560	36895
	6	38968	37586	37489	37898	36620
	7	36459	37132	37757	37864	36008
	8	41825	40869	40891	41993	38714
	9	37157	36388	36217	36579	34104
	10	33262	32481	33136	33330	30898

Tabla 1 – Resultados para c/algoritmo en cada prueba

La tabla 2 muestra los resultados en cuanto al porcentaje de diferencia entre cada uno de los algoritmos propuestos (MATEHA, MATEHAIB y AMTHA), respcto a los algoritmos existentes (MATE y HEFT).

		% de diferencia de MATEHA		% de diferencia de MATE-HAIB		% de diferencia de AMTHA	
Grupo	Prueba	MATE	HEFT	MATE	HEFT	MATE	HEFT
1	1	2,40112734	-11,7341099	1,41671021	-12,8611001	12,2235652	-0,4890891
	2	4,77609805	-11,3024813	5,16459232	-10,8483897	15,6866148	1,45027887
	3	0,11740454	-8,17832766	1,63214211	-6,53778381	8,718326	1,13694188
	4	0,78000866	-6,85226351	0,57944461	-7,06825548	8,19175104	1,12961031
	5	0,05098982	-4,35675114	0,92698089	-3,44212896	4,66868571	0,46458466
	6	-0,90732195	-8,51339241	0,48788915	-7,01301477	7,62180362	0,6586313
	7	0,07991055	-1,44460923	1,07666521	-0,4326467	2,05999375	0,56568487
	8	1,48773405	-21,6320696	4,17715361	-18,3114712	17,4110031	-1,97177491
	9	-0,88016891	-17,6509798	0,2208164	-16,3669614	12,458399	-2,09494335
	10	0,05950116	-9,3367741	0,36416326	-9,00346806	8,17937335	-0,45348212
2	1	-0,32535553	-9,40004805	0,61799243	-8,37137179	10,0815149	1,94815122
	2	1,10496256	-2,37889979	1,17698444	-2,30434073	5,28223133	1,94552529
	3	0,07378312	-6,07409281	1,20532895	-4,87292957	7,00599021	1,28461247
	4	0,58841971	-9,99794127	1,63542715	-8,83943778	10,6508547	1,13604462
	5	3,60975549	-6,84384961	4,52389449	-5,83057145	10,4296133	0,7156277
	6	-0,52951322	-13,1559913	1,73858872	-10,6030164	11,0989717	-0,06697206
	7	3,88039787	-5,73373639	3,52069224	-6,12942073	11,7693457	2,9442847
	8	0,34917676	-9,23943908	0,13597517	-9,47315539	8,77761358	0
	9	1,32260938	-7,67324733	4,32158455	-4,40087263	9,91720326	1,70486682
	10	-0,08007841	-5,83790277	0,07748762	-5,67127163	6,97008529	1,61787216
3	1	0,99413653	-8,9035752	1,52634387	-8,31816258	9,08850946	0
	2	3,68645555	-7,72317495	5,40153528	-5,80492103	10,5916211	0
	3	7,56028805	-12,6800039	5,7358011	-14,9039745	17,4722197	-0,59778867
	4	0,01555547	0,0959048	0,05759135	0,1379069	0,13465713	0,21491075
	5	1,22157448	-16,283233	-1,1662977	-19,0942668	16,3792489	1,56057632
	6	-1,78864464	-27,2752152	1,88369937	-22,6833634	17,1811891	-3,55557857
	7	3,62953759	-3,57373857	1,11837899	-6,27259544	6,37192043	-0,62637445
	8	-1,75188566	-10,575465	0,52257509	-8,10377072	6,97269462	-1,09431864
	9	3,82562441	-15,5299583	4,15037676	-15,1398479	13,3774758	-4,05574824
	10	-5,67419107	-9,39686604	-3,18274736	-6,81765411	4,71470488	1,35800842

Grupo	Prueba	% de diferencia de MATEHA		% de diferencia de MATE-HAIB		% de diferencia de AMTHA	
		MATE	HEFT	MATE	HEFT	MATE	HEFT
4	1	0,6936601	-6,94737954	1,45404555	-6,1284869	8,97450294	1,97062553
	2	3,99663266	-8,87342611	3,86223146	-9,02584493	11,8211204	0
	3	4,41552382	-14,169201	4,13197622	-14,5078795	15,8442283	-0,5183854
	4	-1,24305747	-1,36650315	-0,03919948	-0,1611773	0,12178153	0
	5	0,22037398	-18,916271	1,51038245	-17,378853	16,0925371	0
	6	0,35255594	-21,395769	2,8800477	-18,3166453	15,7461549	-2,64247534
	7	-0,17583071	-4,70949116	1,36630971	-3,09755804	3,70829162	-0,64958499
	8	2,58031991	1,27426111	0,68257174	-0,64892927	7,44693793	6,20612353
	9	3,92027772	-10,0063898	0,80616962	-13,571885	11,4488294	-1,38658147
	10	-4,65061216	-5,16474746	-6,77221499	-7,29677346	4,48249092	4,0132254
5	1	13,6447122	0	13,6447122	0	14,486755	0,97509117
	2	7,52827048	-7,28274261	7,32244336	-7,52153665	13,4189443	-0,44857128
	3	5,30592309	-0,85565133	1,84301249	-4,54388734	6,10930011	0
	4	5,0328989	-3,33029393	5,22219325	-3,12432954	8,09365048	0
	5	18,1867374	-1,85917722	15,0310358	-5,78809006	19,0968769	-0,72603507
	6	11,4552029	-7,90868091	9,87457345	-9,83497859	17,9446951	0
	7	7,86991885	-0,93343495	6,84666088	-2,05446883	9,84627533	1,23176936
	8	5,43191257	-2,52451947	2,0309E-05	-8,41342089	9,86100395	2,27721098
	9	7,24932171	-3,94095087	4,95323431	-6,51405882	10,2070803	-0,62634179
	10	10,4961981	-15,8554386	9,6423301	-16,9607018	21,8095429	-1,21122807
6	1	4,73801772	-9,97496523	10,7039391	-3,0876217	14,1398858	0,87899861
	2	16,3777694	-2,67781594	16,3777694	-2,67781594	21,8704313	4,06649744
	3	21,6537261	0	15,6213406	-7,69964568	23,5435177	2,41210139
	4	10,4640161	-1,53653836	10,4640161	-1,53653836	12,0210438	0,2291786
	5	4,17647087	-4,0510309	2,27770724	-6,11282423	9,87833001	2,14039543
	6	0,00057424	-7,78091812	0,00057424	-7,78091812	7,94283862	0,7793765
	7	-0,75998605	-7,40002988	-0,69457872	-7,33031224	6,18253444	0
	8	8,11380129	-1,67791193	8,11380129	-1,67791193	10,4885778	0,94992898
	9	21,9448799	-0,72278368	21,9448799	-0,72278368	22,5050011	0
	10	0,0997177	1,4837228	0,30303754	1,68422588	0,86863639	2,24198899
7	1	-3,17760659	-13,0258523	7,69000592	-1,12093219	8,20365935	-0,55825081
	2	15,8932965	-12,3330927	18,661502	-8,63587153	23,9560835	-1,56441718
	3	5,19991941	-10,6286871	5,67003071	-10,0800821	11,2759553	-3,53814586

		% de diferencia de MATEHA		% de diferencia de MATE-HAIB		% de diferencia de AMTHA	
	Prueba	MATE	HEFT	MATE	HEFT	MATE	HEFT
	4	16,0339806	-0,71553288	15,8099228	-0,98428567	16,6305167	0
	5	-8,45589772	-25,0331126	-3,9409332	-19,8280469	9,98087927	-3,77831997
	6	3,66129886	-0,90209902	4,04528904	-0,49991988	4,01010269	-0,53677295
	7	2,97650389	-16,1142688	2,195217	-17,0492852	13,6534061	-3,33653207
	8	2,18074015	-5,39652552	2,85286719	-4,67233425	6,34055853	-0,91447972
	9	1,20822719	-6,25047124	5,04997368	-2,11867602	10,2710333	3,4965694
	10	5,54804595	-7,69616027	5,54804595	-7,69616027	12,2977516	0
8	1	7,2073966	-12,4703088	11,4207883	-7,36342043	15,7566623	-2,10807601
	2	6,35144663	-17,7893228	-0,54222635	-26,4600502	19,1560169	-1,68398424
	3	-13,2495023	-4,71942913	-12,6707144	-4,18423613	4,69292313	11,8715537
	4	-5,96794444	-5,62568178	5,00719528	5,31400966	4,64761093	4,95558672
	5	4,23258809	7,92354797	-0,16784922	3,69270737	5,39059791	9,03692707
	6	6,56300014	-0,54936177	2,23874239	-5,20277912	7,07350279	0
	7	1,29066109	1,52620205	3,75467147	3,98433279	4,30975073	4,53808752
	8	9,17361676	-2,92741612	7,88789264	-4,38444058	11,7568607	0
	9	8,9464414	-0,9470934	16,0899186	6,97256695	18,3286991	9,45460483
	10	-3,71182339	-9,88554899	3,38668529	-2,36448246	5,61832348	0
9	1	5,74176344	-6,1714726	6,16167341	-5,69849049	11,8927423	0,75692444
	2	0,75436386	-7,64872321	1,36629482	-6,98498031	8,13601198	0,35792601
	3	17,28638	-10,4800807	16,1519463	-11,995337	25,1325493	0
	4	8,76295192	-10,2229477	6,91084498	-12,4604674	16,061448	-1,40567705
	5	-0,42799377	-8,76651295	1,18635068	-7,01812975	7,83040382	0,17757798
	6	-1,57262885	-10,147412	-0,45108997	-8,93119259	7,62395007	-0,17445593
	7	-2,39452928	-16,0681806	-0,46691097	-13,8831503	14,20218	2,74483476
	8	19,1937081	-10,4537311	21,3347258	-7,52718436	26,8415009	0
	9	0,90665544	-15,7048521	4,58156519	-11,413899	16,0038012	1,92310295
	10	10,6812234	-4,6547703	11,0857784	-4,1807534	15,6936459	1,21828289
10	1	6,83510649	-0,96774194	7,69498836	-0,03584229	11,4259662	4,00761649
	2	6,54826696	-4,17923413	5,77987402	-5,03583235	12,9573134	2,96551891
	3	-1,47096408	-11,3125105	-0,7181697	-10,4867035	9,56938479	0,79862853
	4	2,28969814	-12,96899	2,69956434	-12,4951181	14,9122261	1,62470383
	5	3,75594319	-16,252811	3,14254925	-16,9937271	18,1246472	1,10308912
	6	15,3374219	-1,62734309	15,3374219	-1,62734309	17,792423	1,3195934

		% de diferencia de MATEHA		% de diferencia de MATE-HAIB		% de diferencia de AMTHA	
	Prueba	MATE	HEFT	MATE	HEFT	MATE	HEFT
	7	10,0593295	-4,22292445	13,1559137	-0,63461388	15,8310559	2,46533128
	8	2,5051038	-0,93397875	3,27533133	-0,13658184	6,02836122	2,71355976
	9	4,8995535	-6,33666635	6,06587376	-5,0325441	12,355145	1,99981134
	10	1,60727423	-9,57806149	-0,52611037	-11,9539703	10,8605086	0,72708536
11	1	8,05293841	-5,93677539	8,78116339	-5,09775123	13,172094	-0,03874198
	2	8,86924831	-10,6609699	9,18971057	-10,2718294	18,4507887	0,97398903
	3	5,85023152	-11,0113017	7,77117533	-8,74633098	15,8789956	0,81354052
	4	1,12009655	2,60796476	0,95295337	2,44333662	2,28715614	3,75746334
	5	6,68338582	-11,7798977	5,91563157	-12,699557	18,3153885	2,1535705
	6	12,1653451	-9,54537478	14,1557206	-7,06302391	19,8189289	0
	7	3,61411042	-1,58065633	2,90720758	-2,32565806	7,42935263	2,44021031
	8	2,02352232	-5,82085175	3,11737108	-4,63942525	8,42162388	1,08950645
	9	5,04298429	-13,6733757	7,2843742	-10,9902	15,1904509	-1,52580793
	10	14,6173178	-1,98226154	15,8671956	-0,48939011	16,2769281	0
12	1	14,941124	-4,93545443	15,739129	-3,95097144	20,897095	2,4123093
	2	5,46846344	1,84163287	5,4300464	1,80174191	8,31132461	4,79356426
	3	5,54977436	3,27056492	4,10423108	1,79013875	6,05178312	3,78468781
	4	-2,01614443	-11,9619177	2,33986562	-7,18123087	9,40986203	0,57803468
	5	2,18796873	-4,32379486	2,56187295	-3,92499825	7,40950833	1,24536486
	6	3,05189752	-0,65064695	7,48161321	3,94824399	10,5582003	7,14232902
	7	2,10555853	-11,6939117	1,42867389	-12,4662119	15,2653241	3,32089072
	8	-5,22918395	-4,01416765	-2,95976909	-1,77095632	0,00070819	1,15533817
	9	1,9284855	-7,18990425	5,3720789	-3,42613605	11,0641362	2,79515138
	10	2,40247794	-26,4524766	10,9006742	-15,4417671	24,6062758	2,31593039
13	1	0,81125711	-0,14275957	0,62963182	-0,32613177	0,95265667	0
	2	14,2613896	0	14,2613896	0	14,2613896	0
	3	21,0364066	-9,79717964	28,0823117	0	28,0823117	0
	4	10,4487487	0	10,4487487	0	10,4487487	0
	5	15,9152795	2,14323338	15,9152795	2,14323338	15,9152795	2,14323338
	6	3,13023196	1,77275188	3,13023196	1,77275188	3,40916791	2,05559668
	7	27,0118146	-1,02758147	28,4999336	1,03221857	28,4999336	1,03221857
	8	19,0693198	-1,2228238	20,2126374	0,20716337	20,3773754	0,41320695
	9	5,47905188	0	3,7609713	-1,81767176	5,47905188	0

		% de diferencia de MATEHA		% de diferencia de MATE-HAIB		% de diferencia de AMTHA	
	Prueba	MATE	HEFT	MATE	HEFT	MATE	HEFT
	10	14,3611325	2,33505492	14,3611325	2,33505492	14,3611325	2,33505492
14	1	0,81021389	-0,14674027	0,63719236	-0,32143107	0,95555198	0
	2	13,4713091	0	13,4713091	0	13,4713091	0
	3	26,4781803	-2,23186255	26,4781803	-2,23186255	28,083263	0
	4	10,4784161	1,53657037	10,226951	1,25998771	10,4784161	1,53657037
	5	15,992116	4,14432307	15,992116	4,14432307	15,992116	4,14432307
	6	3,48851148	4,40563087	3,48851148	4,40563087	3,70959043	4,62460897
	7	25,7431561	-1,41480237	28,0998996	1,80387302	28,0998996	1,80387302
	8	18,8782259	0,91727438	20,1376635	2,45556062	20,356899	2,72333637
	9	5,03268652	1,40529621	3,67909246	0	5,03268652	1,40529621
	10	15,3591471	4,37839989	15,3591471	4,37839989	15,3591471	4,37839989
15	1	1,93236328	-3,7026476	9,28581811	4,07334008	9,28581811	4,07334008
	2	24,3028109	-3,22252486	24,8093841	-2,53174942	25,6203824	-1,42585239
	3	0,00060569	0	0,00060569	0	0,00060569	0
	4	5,17355775	-2,25375068	6,97246021	-0,31394867	7,26360489	0
	5	17,6839985	-2,5550072	15,5218057	-5,2488176	20,8640175	1,4068819
	6	0,27488021	2,11904925	0,21920965	2,06440817	3,56988182	5,35311796
	7	19,7028223	0	19,7028223	0	19,7028223	0
	8	19,1933986	-0,6270972	18,1950773	-1,87028989	19,6969766	0
	9	5,78262947	-3,56674142	9,02738733	0	9,02738733	0
	10	11,2957228	-6,09040653	13,5816805	-3,35639878	15,2623468	-1,34632018
16	1	-7,01517334	-7,02139331	0,00581189	0	0,00581189	0
	2	5,12332292	-1,89138577	5,12332292	-1,89138577	6,8844963	0
	3	17,5211479	-0,17044486	16,412448	-1,51695926	17,7035922	0,05113346
	4	5,35084069	2,31984205	5,35084069	2,31984205	6,25954914	3,25765054
	5	11,4470965	-5,27366495	19,0317478	3,74313758	23,3978202	8,93362169
	6	0,39841035	-0,44182622	0,29219698	-0,5489356	0,6108371	-0,22760744
	7	1,960994	3,47322721	1,82870748	3,34298119	1,960994	3,47322721
	8	-4,46615811	0,11501807	0,00187958	4,38711798	1,90938798	6,21097601
	9	10,0085209	-6,01440777	15,1139161	0	15,1139161	0
	10	13,9898637	2,05988533	13,9152669	1,9749416	14,2323033	2,33595243
17	1	8,13360674	-2,99742597	8,87921742	-2,16147304	11,0462792	0,26816179
	2	8,62710665	-3,10368183	8,16355881	-3,62674165	12,3435129	1,0898503

		% de diferencia de MATEHA		% de diferencia de MATE-HAIB		% de diferencia de AMTHA	
	Prueba	MATE	HEFT	MATE	HEFT	MATE	HEFT
	3	7,87307943	-0,40272392	6,91278275	-1,44928449	10,0919861	2,0155082
	4	4,01885431	-4,35257456	3,85737797	-4,52813478	9,3341412	1,42630907
	5	5,05661913	-2,42160111	4,87102131	-2,62181755	7,99559518	0,74886355
	6	6,73378728	-2,52266968	7,65088706	-1,51454986	9,5986902	0,62656827
	7	5,40706419	-4,72233111	5,25167095	-4,89436449	10,4428438	0,85270015
	8	2,61748569	-3,83678806	2,97510527	-3,45546634	6,25915134	0,04623824
	9	3,54112758	-1,07143157	3,96817778	-0,62396029	6,92196589	2,47107478
	10	2,08529021	-3,62338669	3,31934547	-2,31738286	8,10977064	2,75233629
18	1	0,58241402	-0,79938155	-0,06973861	-1,46059838	5,925361	4,61782654
	2	-0,4615757	-0,17971035	-0,25574078	0,02554706	3,93958127	4,20909828
	3	-0,40751445	-0,47600856	-0,01429247	-0,08251834	3,94236053	3,87683374
	4	0,78568557	1,82547549	0,13420516	1,18082275	4,45164313	5,45301293
	5	0,73931629	0,60622645	0,57955407	0,44625003	3,67389589	3,54474078
	6	0,3342916	0,84168146	0,63142309	1,13730028	4,62636792	5,11190718
	7	0,77957078	1,83641214	-1,11549419	-0,03846765	5,04980641	6,06116356
	8	-0,71310759	-0,04976115	-0,31237902	0,34832803	4,60383192	5,23215837
	9	1,40416184	1,4487646	0,76798054	0,8128711	5,75472217	5,79735683
	10	1,08033109	1,31044143	1,47391983	1,70311459	4,12776673	4,35078803
19	1	0,91531941	-7,34530121	-0,00432703	-8,34161793	7,27216417	-0,45849074
	2	5,12146426	-0,43301434	3,36269769	-2,29474446	7,43327581	2,0141377
	3	-0,72749998	-7,95781293	-2,72096042	-10,0943658	5,58762801	-1,18937922
	4	-6,02712089	-9,3752864	-5,66346802	-9,00014997	3,83491856	0,79818035
	5	2,49187459	-3,30926961	0,17451276	-5,76450047	5,77337987	0,16746541
	6	-2,30736813	-5,72530856	-2,54647104	-5,97239956	3,9299832	0,7204236
	7	0,40414608	-4,29905599	-0,06884652	-4,79438465	4,96621452	0,47844642
	8	2,37638847	-3,3677987	2,7711063	-2,94985562	6,70168584	1,21199978
	9	-0,55281461	0,47271431	-2,5721386	-1,52601479	1,20391585	2,21152803
	10	1,68917714	-1,92288366	0,84396807	-2,79914675	4,99371634	1,50306838
20	1	-1,38569209	-3,85731807	1,03086612	-1,38184791	6,40077376	4,11896975
	2	-2,90411197	1,31967846	0,74341064	4,81748527	5,01913951	8,91771306
	3	1,29625831	3,88688408	-0,73160423	1,91224579	5,93384661	8,40275202
	4	-3,58827461	-0,60973219	-0,53615794	2,35462495	4,27213698	7,02466377
	5	-0,64890621	-0,99257236	-2,33352478	-2,68294308	3,42405339	3,09429437

		% de diferencia de MATEHA		% de diferencia de MATE-HAIB		% de diferencia de AMTHA	
	Prueba	MATE	HEFT	MATE	HEFT	MATE	HEFT
	6	-1,00658128	2,52644323	-0,26046304	3,24646362	4,04622384	7,40251051
	7	-2,13719713	-0,20583646	-1,95784877	-0,02987949	4,07570066	5,8895787
	8	-0,40355105	3,27473514	0,48554122	4,13125551	5,00178473	8,48204634
	9	-2,28592099	1,19637721	-1,25644144	2,1908084	1,77339608	5,11749587
	10	0,91183826	2,74550348	-1,3136033	0,56124458	3,53020151	5,31541289
21	1	0,32926149	-1,09281161	-0,23633531	-1,66647818	2,69469788	1,30637414
	2	0,15683611	-5,13506843	0,31376012	-4,96982711	5,87723773	0,8885269
	3	0,49777954	0,04761474	-0,34151714	-0,79547906	1,95795324	1,51439452
	4	-0,73652137	-4,24881292	-1,38879649	-4,92383034	2,69874299	-0,69377422
	5	-0,3853822	-3,41761714	1,04678137	-1,94219373	3,75238175	0,84513186
	6	0,05180464	-6,64382469	0,92023151	-5,71722103	5,98963938	-0,30820847
	7	-0,41451922	-2,85567332	-0,13362654	-2,56795192	2,59289935	0,2248579
	8	0,2516606	-5,69093915	0,03046325	-5,92531454	4,73912677	-0,93612802
	9	0,88379324	-2,75629053	0,53196429	-3,12104054	5,69219487	2,22870165
	10	0,206709	-2,55735092	-0,14490191	-2,9187007	3,24536105	0,56546527
22	1	1,08317938	-3,17811741	2,00978996	-2,21158882	6,76722778	2,75079751
	2	-0,24091768	3,7908167	-0,04162721	3,98209162	2,4922088	6,41401562
	3	0,4839333	6,28208475	1,58456011	7,31858523	3,2060502	8,84560167
	4	-3,34165393	-2,07830463	-4,44612968	-3,16927818	2,99665627	4,18251988
	5	0,33915795	4,19476041	0,03129836	3,89881106	3,65868745	7,38586649
	6	0,49157589	-1,14886731	-0,21203511	-1,86407767	3,21156833	1,61596548
	7	0,01636101	-3,01451917	0,09851164	-2,92987825	5,03892429	2,16029689
	8	-0,67186693	1,21858591	-0,47329645	1,41342756	1,99256746	3,83298658
	9	0,92738072	0,11246361	0,53259502	-0,28556937	3,54762043	2,75425598
	10	-0,08062177	1,76249123	1,4757025	3,29015379	3,8407433	5,61163933
23	1	1,5833046	-4,08806377	-0,76770667	-6,5745546	5,21314457	-0,24905035
	2	-1,6660898	-9,40897778	0,81657478	-6,7372335	6,94716806	-0,13973432
	3	-1,68721828	-6,44168326	-3,76575021	-8,61739856	3,58134559	-0,92678359
	4	-0,76654604	-3,20689176	-3,61462067	-6,12394053	4,95429357	2,65249409
	5	-1,01246869	-6,37477036	0,166593	-5,13311755	5,21318746	0,18137812
	6	-0,15320111	-1,53328601	-1,39631387	-2,79352854	1,91293063	0,56131648
	7	-0,0985159	-9,49410949	1,34491634	-7,9151917	7,23581258	-1,47135553
	8	-1,04081512	-6,07975213	-1,19252895	-6,23903198	5,86528142	1,1707536

		% de diferencia de MATEHA		% de diferencia de MATE-HAIB		% de diferencia de AMTHA	
	Prueba	MATE	HEFT	MATE	HEFT	MATE	HEFT
	9	-0,39080192	-3,50429504	1,05351316	-2,01518636	3,20514063	0,20317109
	10	-1,04049225	-4,51697586	-3,24975079	-6,80224799	4,00673226	0,70390767
24	1	1,38510674	3,4904742	0,91855694	3,03388495	8,62728598	10,5780374
	2	-0,25921975	2,99906537	0,91421686	4,13436693	5,50851954	8,57936115
	3	-6,57836058	-4,3756348	-0,56593169	1,51253123	3,64417002	5,63561997
	4	-2,02583941	7,41509434	0,24047738	9,47169811	4,7445221	13,5589623
	5	1,96911689	2,57881178	2,80292191	3,40743101	6,60047615	7,18136669
	6	-0,56282174	1,40766449	-2,3773738	-0,37133218	6,13096922	7,97029343
	7	-0,41396949	0,5743509	1,36329235	2,33412012	7,19302901	8,10647784
	8	-0,43049859	-1,18405015	-0,6477684	-1,40295018	6,36905873	5,66652571
	9	-0,44027088	1,33825449	-3,14457949	-1,31816812	5,01946747	6,70131566
	10	-0,90371978	0,86932137	-0,51361582	1,25257057	5,76293491	7,41883218
25	1	5,6926459	-2,29075521	6,22023431	-1,71850486	9,48284748	1,82029834
	2	7,38712107	1,00283852	7,04646795	0,63870242	8,01162952	1,67039755
	3	8,01774673	-1,01412887	7,63678381	-1,43249911	10,6806222	1,91021835
	4	7,35341471	0,27230155	7,42939744	0,35409175	9,21143973	2,27233812
	5	5,02039343	0,21538076	3,99823833	-0,85848511	6,24006928	1,49675994
	6	8,9764667	-1,52054069	9,63193448	-0,78948312	11,8212397	1,65229698
	7	9,46381659	-0,6372144	9,61287584	-0,47152475	11,5006992	1,62692114
	8	4,09582098	-4,65951346	4,3236404	-4,41089583	9,34192039	1,0655156
	9	5,67244883	-0,63807014	6,28300214	0,01332923	7,51079275	1,3232591
	10	3,77749575	-2,8074737	2,75859483	-3,89610291	6,74032352	0,35811465
26	1	0,94980281	3,4273536	0,77166616	3,25367271	2,78328783	5,21497751
	2	-0,34518327	0,06794357	0,13726111	0,54840171	4,16250416	4,5570726
	3	-0,6426455	-0,04895745	-0,14370716	0,44703767	3,37065952	3,94067321
	4	-0,0602968	1,50073531	-1,21223575	0,36676766	1,1924368	2,73392512
	5	0,08713548	-1,07202821	0,94936184	-0,19979852	4,76506614	3,66017461
	6	0,77948501	3,27271322	0,71144522	3,20638314	3,83004584	6,2466191
	7	0,57283741	2,83685915	0,79158375	3,0506245	2,36963011	4,5927377
	8	1,29298937	3,78264763	1,76238888	4,24020761	5,02794601	7,42339854
	9	-1,48823589	-5,48098524	-0,55392434	-4,50991602	5,22393263	1,49525336
	10	0,31055664	-2,32992413	0,25499136	-2,38696118	6,0421535	3,5534859
27	1	-2,10693163	-6,79453836	-0,11299707	-4,70906465	4,01878018	-0,38760247

		% de diferencia de MATEHA		% de diferencia de MATE-HAIB		% de diferencia de AMTHA	
	Prueba	MATE	HEFT	MATE	HEFT	MATE	HEFT
	2	1,80545085	2,21878519	-0,03449692	0,38658239	2,65508468	3,06484262
	3	-4,16383266	-0,55500661	0,1420143	3,60166138	1,79323516	5,19567458
	4	-3,20586048	-2,18649209	-0,49551724	0,49708099	1,07768942	2,05474902
	5	2,2556831	-4,70601034	0,16661589	-6,94386825	6,75741036	0,11634575
	6	0,23217138	-1,87833921	-1,33788486	-3,48160877	3,05761614	1,00687562
	7	-0,36377187	-2,26885227	0,46760332	-1,42169612	4,11816526	2,29815989
	8	-1,5478064	-3,21668849	-2,27168235	-3,95246095	4,32320434	2,75080906
	9	1,71237839	-2,41960133	1,08845845	-3,06975066	7,43774053	3,54645318
	10	0,06078889	-2,7388994	1,12250481	-1,64744067	3,99060052	1,30100163
28	1	-0,10287554	5,61648336	-0,00829641	5,70565872	2,1686827	7,75825654
	2	0,67726243	9,02124227	-0,98129665	7,50201667	2,67130756	10,8477701
	3	0,57105039	1,96688915	-0,78850336	0,62642157	3,40361161	4,7596853
	4	-2,2974607	-4,84608795	-2,04317519	-4,5854672	4,48289352	2,10319124
	5	1,33363472	-1,98938549	-1,68399638	-5,10864849	6,3840093	3,23108248
	6	0,37296883	4,05397534	1,3386557	4,98398214	3,43131325	6,99932045
	7	0,53860678	0,68664418	-0,91424029	-0,76404048	5,43376727	5,57451875
	8	-1,06658766	0,43416396	-1,35915858	0,14593747	4,27190578	5,69338538
	9	0,88338903	-1,17651398	-0,26736519	-2,35118389	5,52795592	3,56457918
	10	-0,39534063	5,25018322	-1,12777974	4,55893131	4,39639958	9,77246985
29	1	0,89283242	-0,95396664	1,72967696	-0,10152804	3,52641633	1,72869243
	2	0,82574681	-2,70609063	1,14587671	-2,37456013	4,78751117	1,39676184
	3	0,71417844	-2,00042512	0,41275313	-2,31009179	2,85587354	0,19982672
	4	-0,24180734	-1,34518906	-0,10242542	-1,20427294	1,78915497	0,70812845
	5	0,29148649	-0,34645927	0,74028316	0,10520884	1,4528057	0,82229018
	6	0,24045502	-2,44961157	0,75566693	-1,9205067	3,58050058	0,98050001
	7	1,24236643	-0,84920526	2,22883202	0,15815252	4,55993408	2,5386246
	8	-0,36302395	-3,70765358	0,19663329	-3,12934559	2,81582465	-0,42286884
	9	-0,11598023	-3,36942127	0,86754183	-2,35393798	4,76564457	1,67084032
	10	0,21700472	-1,9061315	0,55027107	-1,56577407	3,77455931	1,72711908
30	1	-0,65265261	2,19700228	0,11918004	2,94698298	2,95680007	5,70426496
	2	-0,22169811	-2,48779017	0,20165094	-2,05486886	4,3879717	2,22610793
	3	-0,98420638	-2,13488541	-1,4392571	-2,59512127	4,36061322	3,27083651
	4	0,04443112	-3,65494045	-0,97970609	-4,71698113	6,01930532	2,54106481

		% de diferencia de MATEHA		% de diferencia de MATE-HAIB		% de diferencia de AMTHA	
	Prueba	MATE	HEFT	MATE	HEFT	MATE	HEFT
	5	0,30188841	3,52921678	0,08136	3,31582709	1,53727572	4,72461336
	6	-0,97840926	1,34614435	-0,64974465	1,667243	3,43580927	5,65874623
	7	3,42640247	4,24571028	1,84637159	2,67908401	3,98996397	4,80449066
	8	0,11414801	3,9069268	0,4844949	4,2632112	2,04832266	5,76765865
	9	-0,83952407	2,66993121	0,04469409	3,52337647	3,93307966	7,27643698
	10	0,4535096	1,25732408	-1,16551967	-0,3486319	4,30493985	5,07765494
31	1	2,20891158	0,10534011	0,20162635	-1,94512357	2,74581228	0,65379002
	2	0,33482285	-2,83632045	-0,36053928	-3,55380759	4,60069511	1,56528319
	3	-0,87526923	-2,89640592	-0,16861187	-2,17558997	3,47867004	1,54476887
	4	-1,62217264	-3,13627736	1,99754922	0,53737601	3,56808416	2,13131091
	5	-0,24472826	3,74006795	-1,15463785	2,86632788	0,51313989	4,46781032
	6	-1,62501383	-4,31909514	0,60570414	-2,02924079	3,66093519	1,1069845
	7	-1,71159431	-2,15423261	-1,30604756	-1,74692095	0,28896667	-0,14496539
	8	-0,55862007	-2,58637639	-0,60183793	-2,63046574	1,69279693	-0,28955976
	9	0,41822464	1,69120804	-0,24480775	1,03665139	0,21417259	1,48976445
	10	1,81614499	-3,59196266	-1,74082829	-7,34485913	6,49748892	1,34723641
32	1	-1,31359403	6,36460495	-2,60595213	5,1701901	1,19224585	8,68053608
	2	0,00996289	4,20683402	0,62268052	4,79383411	3,93285013	7,96506634
	3	-0,24899539	2,6688049	1,76269014	4,62193925	3,99871804	6,79288638
	4	-1,20009126	6,48704488	-0,67761807	6,96983113	1,321013	8,81664664
	5	-3,13002239	-1,55406808	-1,59399289	-0,041511	2,79278092	4,27822748
	6	0,25807482	3,79542188	-0,83009631	2,74584274	2,57010589	6,02545679
	7	-1,68318432	-3,56016347	-1,97134547	-3,85364382	3,02703867	1,23700595
	8	-0,05383053	2,23311417	-2,7502508	-0,40167364	5,27294526	7,43813509
	9	0,46993514	2,52980596	-0,52489832	1,55556154	6,27679455	8,2164868
	10	-2,01656353	0,37881065	-2,61383578	-0,2044375	4,87361842	7,10720943

Esta publicación se terminó de imprimir
en el mes de mayo de 2011,
en la ciudad de La Plata,
Buenos Aires,
Argentina.