Un modelo para el mapeo automático de tareas a procesadores en arquitecturas multicluster heterogéneas *

Laura De Giusti [1], Franco Chichizola [2], Marcelo Naiouf [3], Ana Ripoll [4], Armando De Giusti [5]

Instituto de Investigación en Informática (III-LIDI) – Facultad de Informática – UNLP
Dpto. de Arquitectura de Computadoras y Sistemas Operativos – Univ. Autónoma de Barcelona.

Abstract

This paper analyzes different techniques for automatic mapping of concurrent tasks to distributed processors, using a task interaction graph with information on processing and communication time between concurrent tasks.

The first model considers clusters with homogeneous processors and fixed communication time between them. Then a new model is presented, considering heterogeneous processors with different communication levels, as in multi-cluster architectures.

Experimental results are presented and future research lines are discussed, in particular the optimal required number of processors for obtaining a fixed efficiency.

Keywords: *Parallel Systems. Cluster and Multi-cluster Architectures.Perfomance prediction models. Tasks to processors mapping. Homogeneous and Non-Homogeneous Processors.*

Resumen

En este trabajo se discuten técnicas de mapeo automático de tareas concurrentes a procesadores, mediante el análisis del grafo de relación entre tareas, en el que se incorporan los tiempos de procesamiento y comunicación.

Partiendo de un primer análisis en el que los procesadores son homogéneos y los tiempos de trasmisión de datos no dependen de los procesadores que se están comunicando (caso clásico en clusters homogéneos), se avanza para extender al modelo a procesadores heterogéneos con posibilidad de diferentes niveles de comunicación, aplicable a multi-cluster.

Se presentan algunos resultados iniciales obtenidos con el modelo y se plantean las líneas de trabajo futuras, en particular la posibilidad de obtener el número óptimo de procesadores requeridos, manteniendo un nivel de eficiencia constante.

Palabras Clave: *Sistemas Paralelos. Arquitecturas de Cluster y multi-Cluster Modelos de predicción de perfomance. Mapping de tareas a procesadores. Procesadores homogéneos y no-homogéneos.*

VI Workshop de Procesamiento Distribuido y Paralelo.

---

[1] Becaria de Formación Superior UNLP. Jefe de Trabajos Prácticos Facultad de Informática UNLP. ldgiusti@lidi.info.unlp.edu.ar.
[2] Becario de Doctorado del CONICET. Jefe de Trabajos Prácticos Facultad de Informática UNLP. francoch@lidi.info.unlp.edu.ar.
[3] Profesor Titular D.E. Facultad de Informática UNLP. mnaiouf@lidi.info.unlp.edu.ar.
[4] Full Professor. Departamento de Arquitectura del Computador y Sistemas Operativos – Universidad Autónoma de Barcelona. ana.ripoll@uab.es
[5] Investigador Principal CONICET. Profesor Titular D.E. Facultad de Informática UNLP. degiusti@lidi.info.unlp.edu.ar.

* *Esta investigación es financiada por la CIC , la Fundación YPF y el proyecto Grid CyTED.*

1. INTRODUCCIÓN

En la Ciencia Informática, los modelos de computación son usados para describir entidades reales tales como arquitecturas de procesamiento y resultan una versión "abstracta" o simplificada de la máquina física, capturando características esenciales e ignorando detalles sin importancia de la implementación [1][2]. Un modelo no se relaciona necesariamente a ninguna computadora real, sino que su principal razón de ser es ayudar a comprender la computación. Provee un marco para estudiar problemas, obtener ideas sobre sus distintas estructuras, y desarrollar soluciones.
Una vez que un algoritmo fue diseñado para resolver un problema con un cierto modelo, éste permite dar una descripción significativa del algoritmo, derivar un análisis preciso e incluso predecir perfomance [3].

La computación monoprocesador se benefició por la existencia de un modelo teórico simple de computadora conocido como RAM [4]. Esto hizo posible desarrollar algoritmos para arquitecturas uniprocesador, estableciendo correctitud y performance esperada, en forma relativamente independiente de la máquina específica sobre la cual se ejecutaría el algoritmo. La optimización para características dependientes de la máquina tales como el reloj del procesador, capacidad de memoria, número de registros, niveles y velocidad de caché, etc, es manejada por el compilador y/o el monitor de ejecución. Generalmente, a nivel del algoritmo o del programador, estos elementos no son tenidos en cuenta. La simplicidad del modelo RAM y su precisión en la modelización de máquinas uniprocesador está largamente demostrada y permitió grandes avances para lograr eficiencia en el cómputo monoprocesador.

En el caso de computadoras paralelas, los requerimientos mínimos que debe cumplir un modelo siguen las ideas expresadas para el caso monoprocesador: ser conceptualmente simple de entender y usar, que la determinación de corrección de un algoritmo sobre el modelo sea válida independientemente de la arquitectura física, que *la performance real se corresponda con la predicha por el modelo*, y que se aproxime a las arquitecturas reales para minimizar la brecha conceptual entre modelo y arquitectura física.
En estos requerimientos queda claro que un objetivo central de los modelos de cómputo paralelo es la posibilidad de *predicción de performance* que brinden: el éxito o fracaso dependerán en gran parte de este punto [5].

Al tratar las máquinas paralelas se encuentran un gran número de modelos abstractos, pero ninguno con la simplicidad y precisión del RAM. Además, ninguno intenta servir como modelo para todas las clases de máquinas paralelas, y la tecnología de compiladores paralelos no puede manejar el "gap" entre los modelos y las computadoras comerciales. Las dificultades involucradas en formular un único modelo simple y preciso para computadoras paralelas pueden medirse examinando las variaciones en las máquinas comerciales y propuestas: hay sincrónicas, asincrónicas y semi-sincrónicas; algunas tienen memoria compartida, otras memoria distribuida, y otras combinan ambas; algunas operan en modo SIMD mientras otras en modo MIMD; difieren en la red de interconexión usada y/o en los métodos de ruteo; etc. [6] [7]

Cada modelo intenta proveer una abstracción para desarrollar algoritmos y programas en una clase de computadoras paralelas. La abstracción es generalmente más simple para trabajar que cualquier instancia de la clase modelizada. Naturalmente esta simplificación se obtiene a expensas de introducir imprecisiones en la modelización. Dada la incapacidad de los compiladores paralelos para compensar esta imprecisión, los algoritmos desarrollados para el modelo pueden resultar en

código ineficiente sobre la computadora de destino. Como resultado, la aplicabilidad del modelo está limitada.

Los problemas son paralelizables en distintos grados y no todos son adecuados para el procesamiento paralelo. Algunos problemas pueden ser completamente secuenciales. En otros, asignar "subproblemas" a diferentes procesadores podría significar mayor consumo de tiempo global. Por otra parte, un problema puede tener distintas formulaciones paralelas y la eficiencia de cada una de ellas dependerá de la arquitectura de soporte y el algoritmo específico que se desarrolle.

Dado que el paralelismo implica la existencia de varios procesos que interactúan en la resolución de un problema, es necesario considerar conceptos como la comunicación (para intercambiar datos entre procesos), la sincronización (para evitar interacciones indeseadas), la arquitectura sobre la cual se ejecutará el programa, el modelo de comunicación utilizado y la manera de dividir el problema y los datos, etc.

En nuestros días las arquitecturas más utilizadas por su relación costo/performance son los cluster y multiclusters de procesadores, razón por la cual resulta de importancia estudiar la predicción de performance, con el objetivo de determinar el grado de precisión de los modelos existentes y la necesidad de adecuación de los mismos a esta clase de plataformas [8]. Un elemento fundamental que aparece en estas arquitecturas es la potencial *heterogeneidad* de los procesadores, lo cual agrega un elemento a la complejidad intrínseca de la modelización.

Se han analizado diferentes modelos para caracterizar el comportamiento de aplicaciones paralelas entre los cuales se pueden mencionar PRAM, LOGP, BSP, TIG, TPG, TTIG. En particular la investigación se centra en los modelos que caracterizan el comportamiento de las aplicaciones paralelas en arquitecturas distribuidas, modelos TIG (Grafo de Interacción de Tareas), TPG (Grafo de Precedencias de Tareas) y el TTIG (Grafo de Interacción Temporal de Tareas) el cual es una combinación de los dos anteriores [9] [10] [11].

Una vez definido el grafo que modela la aplicación el problema de "mapping" se resuelve mediante algún algoritmo que establece un mecanismo automático para realizar la asignación de tareas a procesadores, y así obtener mejores resultados en la ejecución de la aplicación [12] [13] [14] [15].
Este es un problema NP-completo, debido a la existencia de gran cantidad de factores a tener en cuenta que directa o indirectamente influyen en el tiempo de ejecución del programa [16]. Los algoritmos de mapping estático pueden clasificarse en dos grandes grupos [17]:

- *óptimo:* se evalúan todas las posibles formas de asignar las tareas a los diferentes procesadores. Este tipo de soluciones solo puede abordarse cuando el número de configuraciones posibles es lo suficientemente bajo. En caso contrario la solución óptima no puede llevarse a cabo debido a la explosión combinatoria en el número de soluciones posibles.

- *heurístico:* se basan en técnicas de aproximación que utilizan suposiciones "realistas" del algoritmo y sistema paralelo. Dichos algoritmos producen soluciones subóptimas en tiempos de ejecución más razonables comparado con las estrategias óptimas.

1.1 Contribución de este trabajo

Este trabajo estudia el modelo TTIG el cual implementa un grafo TTIG que 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, en virtud de sus dependencias mutuas. El uso de este modelo permite representar aplicaciones paralelas en forma más realista y constituye una alternativa que unifica los dos modelos clásicos [18] [19].
El modelo TTIG está pensado para ser aplicado a arquitecturas homogéneas. En este trabajo se lo extiende para que pueda ser utilizado en arquitecturas heterogéneas. Esto implica considerar la heterogeneidad de los procesadores dentro de la arquitectura, como así también los distintos tipos de interconexión entre los procesadores y el costo de comunicaciones asociado; además debe redefinirse el algoritmo de mapping para que tenga en cuenta las nuevas características del modelo.

2. MODELOS Y ALGORITMOS DE MAPPING

En esta investigación se trabajó en base al modelo TTIG al que luego se le realizaron algunas modificaciones para considerar arquitecturas con procesadores heterogéneos los cuales se conectan por medio de una red, creando el modelo TTIGHe.
Luego se realizaron modificaciones al algoritmo de mapping existente MATE para adaptarlo al nuevo modelo.

2.1 Modelo TTIG (Temporal Task Interaction Graph) y Algoritmo de Mapping

2.1.1 Modelo TTIG

Es un nuevo modelo de grafo que representa las aplicaciones con la estructura de interconexión de tareas definidas por el programador.
El grafo del modelo TTIG *G(V,E)* está compuesto por dos elementos:

- *V*, el conjunto de nodos, donde cada uno representa una tarea del programa T_i.
- *E*, el conjunto de aristas que representan la comunicación entre los nodos o tareas de grafo.

Cada nodo contiene información relacionada con su tiempo de ejecución w(Ti). Las aristas del grafo mantienen la información que involucra el tiempo de comunicación c(Ti,Tj) y el grado de concurrencia entre dos tareas adyacentes p(Ti,Tj). El grado de concurrencia entre una tarea Ti y otra Tj adyacentes indica el máximo tiempo en que Ti y Tj pueden ejecutarse en forma paralela (0 si deben ejecutarse en forma secuencial y 1 si son totalmente independientes).

Cálculo de los elementos del grafo TTIG
Sea CP(Ti) el conjunto de subtareas (fases de cómputo) de la tarea Ti, $ct_m(Ti)$ el tiempo de la fase de cómputo m de la tarea i.
Sea CM(Ti,Tj) el conjunto de comunicaciones de Ti a Tj, y $cc_m(Ti,Tj)$ el tiempo de la comunicación m entre Ti y Tj.
El cálculo del grafo TTIG se hace por medio de la utilización de los siguientes valores:

- Tiempo de cómputo de la tarea i: es la suma del tiempo de cómputo de todas las subtareas que la componen, y está dado por la siguiente fórmula

$$w(Ti) = \sum_{m \in CP(Ti)} ct_m(Ti)$$

- Tiempo de comunicación entre las tareas adyacentes Ti y Tj: es la suma de los tiempos de todas aquellas comunicaciones desde Ti hacia Tj, y está dado por la siguiente formula

$$c(Ti,Tj) = \sum_{m \in CM(Ti,Tj)} cc_m(Ti,Tj)$$

- Grado de concurrencia entre las tareas adyacentes Ti y Tj: se calcula a partir de la simulación del subgrafo aislado de ambas tareas, en el cual se consideran las dependencias entre las subtareas de Ti y Tj, sin incluir las comunicaciones entre ellas. Se calcula a partir de la siguiente fórmula

$$p(Ti,Tj) = \frac{TP(Ti,Tj)}{w(Tj)}$$

  donde TP(Ti,Tj) es el máximo tiempo en que las dos tareas se pueden ejecutar en forma paralela.

2.1.2 Algoritmo de mapping para el modelo TTIG

Existen diferentes algoritmos de mapping para la asignación de tareas a procesadores. En este trabajo se analizaron algunos de ellos y optó por utilizar el algoritmo MATE (Mapping Algorithm based on Task dependencies).
La estrategia de este algoritmo es determinar para cada par de tareas adyacentes la diferencia de tiempo al ejecutarlas en un mismo procesador o en procesadores diferentes. Dichos valores de tiempo se calculan teniendo en cuenta, además del tiempo de cómputo, el volumen de comunicación entre las tareas y el grado de paralelismo.
Mediante la evaluación de estos tiempos se lleva a cabo una política en la que se asignan al mismo procesador aquellas tareas con mayor grado de dependencia relativa, mientras que se asignan a procesadores diferentes aquellas tareas que pueden ejecutarse en forma concurrente.
Esta heurística se define considerando un número N de procesadores homogéneos, y sin considerar la topología de interconexión de los nodos.

El algoritmo MATE puede dividirse en tres pasos, el primero determina el nivel de cada uno de los nodos del grafo, el segundo paso consiste en evaluar qué decisión se debe tomar para cada par de tareas adyacentes, es decir, se decide si dos tareas adyacentes deben ejecutarse en forma conjunta o separada. Para este paso se necesita calcular los tiempos de ejecución en forma conjunta y separada para cada par de tareas adyacentes. El último paso consiste en asignar cada una de las tareas a un procesador.

Un pseudocódigo del algoritmo sería el siguiente:

```
Procedure MATE ( G(V,E) )
{
    Calcular el nivel de cada nodo perteneciente a V del grafo G.
    Calcular los tiempos de ejecución junta y separada.
    Realizar el algoritmo de mapping.
    Asignar cada tarea a un procesador
 }
```

Procedure CalcularNivel (G)
{

Dado un grafo G el nivel de un nodo LN(T) se define como el mínimo número de tareas que tienen que haber comenzado su ejecución para que la tarea que corresponde al nodo T pueda también iniciarse. La siguiente fórmula expresa lo dicho anteriormente:

$$LN(T) = \min_{T_{in} \in S} d(T_{in}, T)$$

donde:

S es el conjunto de nodos iniciales (tareas que no dependen de ninguna otra para comenzar su ejecución).

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

}

Procedure CalcularTiempos(G)
{

Considerando cada par de tareas adyacentes Ti,Tj del grafo G en forma aislada, se puede calcular el mínimo tiempo necesario para ejecutar ambas tareas en el mismo procesador (t_junto), o bien separadas en distintos procesadores (t_sep). Los tiempos mencionados se calculan por medio de las siguientes fórmulas:

$$t\_sep(Ti,Tj) = w(Ti) + w(Tj) + c(Ti, Tj) + c(Tj, Ti) - [w(Tj) * p(Ti, Tj)]$$

$$t\_junto(Ti,Tj) = w(Ti) + w(Tj) + acum(Tj)$$

donde

acum(Tj) es el tiempo acumulado como consecuencia de asignar otras tareas adyacentes a Tj en el mismo procesador.

}

Procedure Mapping(G)
{

Este algoritmo parte de una lista inicial donde las tareas están ordenadas de menor a mayor valor de nivel LN(Ti) y realiza los siguientes pasos para realizar la asignación de las tareas en los procesadores:

a. Seleccionar el primer nivel n sin asignar.
b. Ordenar de forma decreciente según el valor de máxima ganancia max_gain(Ti) aquellas tareas pertenecientes al nivel n que aun no han sido asignadas. El valor de max_gain(Ti) indica la ganancia en tiempo que puede ser obtenida cuando la tarea Ti se ubica en el mismo procesador que una tarea adyacente Tj previamente asignada. La forma para obtener el valor de max_gain para una tarea Ti es:

$$max\_gain(Ti) = \max_{Tj \in P\_as(Ti)} (t\_sep (Ti, Tj) - t\_junto (Ti, Tj))$$

donde

P_as(Ti) es el conjunto de tareas adyacentes a Ti, que ya fueron asignadas.

c. Asignar la primer tarea Ti de la lista generada en b., a un procesador específico. Si el valor de max_gain(Ti) es positivo significa que existe una dependencia fuerte con otra tarea adyacente ya asignada Tj, por lo que Ti se asigna al mismo procesador que Tj. En caso contrario la tarea Ti se asigna al procesador menos cargado, es decir, al procesador con menor tiempo de cómputo acumulado hasta el momento.
d. Si aún existen tareas sin asignar de nivel n, se continúa con el paso b.
e. Si aún existen niveles sin procesar se vuelve al paso a.

}

2.2 Modelo TTIGHe y Algoritmo de Mapping MATEHe

El incremento en la utilización de las arquitecturas de clusters heterogéneos conlleva a analizar la posibilidad de aplicar el modelo anteriormente mencionado en estas arquitecturas. Como se explicó en el punto 2.1.1, el modelo TTIG no es aplicable a arquitecturas heterogéneas debido a que no considera los diferentes tipos de procesadores y comunicaciones que pueden formar una arquitectura heterogénea.
Por lo tanto, en este trabajo se propone una adaptación al modelo TTIG, denominado TTIGHe para ser utilizado en arquitecturas heterogéneas. Este nuevo modelo requiere también la adaptación del algoritmo de mapping utilizado, para lo cual se propone un nuevo algoritmo de mapping llamado MATEHe.

2.2.1 Modelo TTIGHe

En este nuevo modelo se debe agregar información relacionada a las diferentes características de los procesadores y las comunicaciones entre los mismos. El grafo del modelo TTIGHe G(V,E,Tp,Tc) está compuesto por cuatro elementos:

- V, el conjunto de nodos, donde cada uno representa una tarea del programa Ti.
- E, el conjunto de aristas que representan la comunicación entre los nodos o tareas de grafo.
- Tp es el conjunto de procesadores, donde para cada uno se indica a qué tipo de procesador pertenece.
- Tc es el conjunto de tipos diferentes de comunicaciones, donde para cada tipo de comunicación se indica el tiempo de startup y el tiempo de transferencia de un byte.

A diferencia del modelo TTIG donde cada nodo sólo tenía la información en cuanto a su tiempo de ejecución, ahora el modelo almacena para cada nodo el tiempo de ejecución en cada uno de los diferentes procesadores existentes en la arquitectura $w_p(Ti)$ (representa el tiempo de ejecutar la tarea i en el procesador p).
El conjunto de aristas también amplia su información, ya que ahora mantiene para una arista A entre Ti y Tj una matriz C_{sd} de dimensión #m x #m (#m cantidad de procesadores totales de la arquitectura), donde $C_{sd}(Ti,Tj)$ es el tiempo de comunicación entre la tarea Ti en el procesador s y la tarea Tj en el procesador d.
Además se cuenta con una matriz P_{sd} de dimensión #m x #m donde $p_{sd}(Ti,Tj)$ representa el grado de concurrencia entre la tarea Ti en el procesador s y la tarea Tj en el procesador d.
Cada uno de los elementos de las matrices c y p se calculan de la misma manera que se hace en el modelo TTIG.

2.2.2 Algoritmo de Mapping MATEHe

Un pseudocódigo del algoritmo podría expresarse de la siguiente manera:

```
Procedure MATEHe ( G(V,E) )
{
    Calcular el nivel de cada nodo perteneciente a V del grafo G.
    Calcular los tiempos de ejecución junta y separada.
    Realizar el algoritmo de mapping.
    Asignar cada tarea a un procesador.
}
```

Procedure CalcularNivel (G)
{
Dado un grafo G el nivel de un nodo LN(T) se define como el mínimo número de tareas que deben haber comenzado su ejecución para que la tarea que corresponde al nodo T pueda también iniciarse. La siguiente fórmula expresa lo dicho anteriormente:

$$LN(T) = \min_{T_{in} \in S} d(T_{in}, T)$$

donde:
S es el conjunto de nodos iniciales (tareas que no dependen de ninguna otra para comenzar su ejecución).
$d(T_{in},T)$ corresponde al mínimo número de arcos que hay que recorrer desde T_{in} a T.
}

Procedure CalcularTiempos(G)
{
Este procedimiento difiere del procedimiento CalcularTiempos para una arquitectura homogénea, ya que el tiempo de ejecutar en forma separada dos tareas Ti,Tj es diferente de acuerdo al procesador en que sean ejecutadas, por lo tanto no obtendremos un único tiempo sino un vector de #p elementos donde cada posición del vector representa el tiempo de ejecutar en un procesador determinado la tarea Ti.
Otra modificación se realiza sobre el cálculo del tiempo de ejecución conjunta, dado que ahora el tiempo de ejecutar la tarea Ti, es el tiempo que requiere la ejecución de dicha tarea en el procesador donde Tj ha sido asignado. Por lo tanto las fórmulas para calcular los tiempos de ejecución junta y separada se reescriben de la siguiente manera:

$$t\_sep_{sd}(Ti,Tj) = w_d(Ti) + w_s(Tj) + c_{sd}(Ti, Tj) + c_{ds}(Tj, Ti) - [w_s(Tj) * p_{sd}(Ti, Tj)]$$

$$t\_junto(Ti,Tj) = w_d(Ti) + w_d(Tj) + acum_d(Tj)$$

donde
d es el procesador al que ha sido asignado Tj.
$acum_d(Tj)$ es el tiempo acumulado como consecuencia de asignar otras tareas adyacentes a Tj en el mismo procesador.
}

Procedure MappingHe(G)
{
Este algoritmo parte de una lista inicial donde las tareas están ordenadas de menor a mayor valor de nivel LN(Ti) y realiza los siguientes pasos para realizar la asignación de las tareas en los procesadores:

a. Seleccionar el primer nivel n sin asignar.
b. Ordenar de forma decreciente según el valor de máxima ganancia max_gain(Ti) aquellas tareas pertenecientes al nivel n que aun no han sido asignadas. El valor de max_gain(Ti) indica la ganancia en tiempo que puede ser obtenida cuando la tarea Ti se ubica en el mismo procesador que una tarea adyacente Tj previamente asignada. La forma para obtener el valor de max_gain para una tarea Ti es:

$$max\_gain(Ti) = \max_{Tj \in P\_as(Ti)} (t\_sep_{min}(Ti, Tj) - t\_junto(Ti, Tj))$$

donde:
P_as(Ti) es el conjunto de tareas adyacentes a Ti, que ya fueron asignadas.
min representa el procesador donde se obtiene el menor tiempo de t_sep(Ti,Tj).

c. Asignar la primer tarea Ti de la lista generada en b., a un procesador específico. Si el valor de max_gain(Ti) es positivo significa que existe una dependencia fuerte con otra tarea adyacente ya asignada Tj, por lo que Ti se asigna al mismo procesador que Tj. En caso contrario la tarea Ti se asigna al procesador al cual agregando esa tarea su tiempo es mínimo. El tiempo del procesador está dado por el tiempo que tiene hasta el momento más el tiempo de ejecutar Ti en ese procesador más el tiempo de comunicación.
d. Si aún existen tareas sin asignar de nivel n, se continúa con el paso b.
e. Si aún existen niveles sin procesar se vuelve al paso a.

}

3. RESULTADOS EXPERIMENTALES

Para la realización de las pruebas se desarrolló un sistema, el cual permite dos operaciones básicas. La primera es la instanciación del modelo para lo cual el sistema requiere que se especifique:

- cantidad de tipos diferentes de procesadores.
- cantidad de máquinas de cada tipo.
- cantidad de tipos diferentes de comunicación.
- para cada tipo de comunicación su tiempo de startup y transferencia de un byte
- para cada par de procesadores el tipo de comunicación que utilizan.
- cantidad de subtareas de la aplicación.
- tiempo de cada una de las subtareas para cada uno de los tipos de procesadores
- información referida sobre qué subtarea pertenece a cada tarea.
- para cada par de subtareas el volumen de información que intercambian.

Con toda esta información ingresada se crea el modelo TTIGHe y se permite la segunda acción del sistema la cual calcula el mapping de tareas a procesadores utilizando el algoritmo MATEHe. Este sistema permite realizar el mapping de manera automática o manual. Esta última da la posibilidad de agrupar las tareas en algún procesador en particular y de seleccionar un orden de ejecución diferente para las subtareas de cada tarea.

Una vez que se realiza el mapping el sistema permite simular la ejecución del programa paralelo de acuerdo a la asignación calculada. Cuando finaliza esta simulación se visualiza la información obtenida en el proceso de mapping. Esta información incluye:

- para cada tarea en qué procesador quedó asignada
- orden de ejecución en cada procesador
- un gráfico que muestra la ocupación y espera de cada uno de los procesadores

Con el sistema descrito anteriormente se realizaron diferentes pruebas para verificar la adaptación del modelo TTIGHe a una arquitectura heterogénea [20]. Estas pruebas consisten en ejecutar un algoritmo con paralelismo funcional en diferentes arquitecturas simuladas (variando la cantidad de tipos de procesadores, de procesadores por tipo, y de tipos de comunicaciones).

En todas las pruebas realizadas la asignación generada a partir del algoritmo MATEHe sobre el modelo TTIGHe obtuvo mejores resultados en cuanto a tiempo final que las realizadas de forma manual. En la figura 1 y 2 se muestran los gráficos obtenidos para un ejemplo con 2 tipos de procesadores con dos máquinas cada uno de ellos, y 3 tipos de comunicaciones, la primera para el mapping automático y la segunda para el manual. El tiempo requerido por cada procesador se

representa por medio de una línea de tiempo donde en oscuro se representan los momentos de cómputo, y en blanco los de espera.

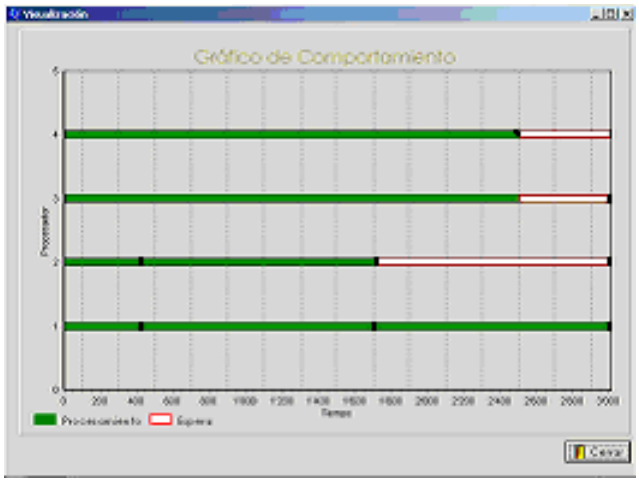

Figura 1: Mapping Automático

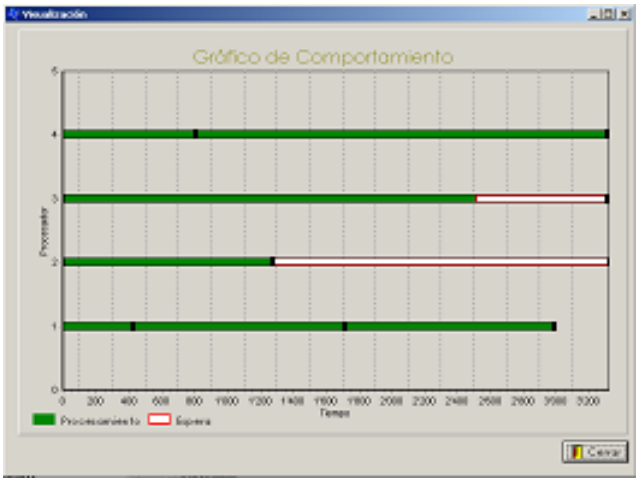

Figura 2: Mapping Manual

4. CONCLUSIONES

El objetivo de este trabajo era analizar el modelo TTIG para ver su comportamiento en una arquitectura heterogénea. Esto generó la definición de una ampliación al modelo TTIG llamada TTIGHe la cual considera la heterogeneidad de los procesadores y de la arquitectura.
Una vez realizada la ampliación se analizaron e implementaron las modificaciones necesarias al algoritmo de mapping para que contemple las características del nuevo modelo.
Se verificó el funcionamiento del modelo a través de las diferentes pruebas realizadas. Con la ejecución de estas pruebas se pudo comprobar que el algoritmo de mapping automático MATEHe logra una asignación de modo tal que minimiza el tiempo final de la aplicación modelada, frente al tiempo requerido por la misma aplicación ejecutada a partir de la asignación manual.
Una vez que la simulación se ejecuta se obtiene como resultado la asignación de cada una de las tareas a los procesadores y la ocupación de los mismos. Por lo tanto se puede estimar la arquitectura óptima requerida (cantidad de máquinas de cada tipo de procesadores) para la ejecución de la aplicación, lo cual lleva a un mejoramiento de la eficiencia.

5. TRABAJO FUTURO

Se continuará el estudio del algoritmo de mapping MATEHe para tratar de obtener una optimización del speed-up y balance de carga alcanzables.

Se realizarán modificaciones en el algoritmo MATEHe para poder determinar la arquitectura óptima de manera automática, y a partir de este dato, lograr una asignación que sin aumentar el tiempo final de la aplicación, incremente la eficiencia de la misma.

Se corroborarán los resultados de las pruebas frente a ejecución en arquitecturas reales.

REFERENCIAS

[1] Grama A., Gupta A., Karypis G., Kumar V., "An Introduction to Parallel Computing. Design and Analysis of Algorithms", Pearson Addison Wesley, 2nd Edition, 2003

[2] Hagit Attiya, Jennifer Welch, Distributed Computing: Fundamentals, Simulations, and Advanced Topics (Wiley Series on Parallel and Distributed Computing). Wiley-Interscience; 2 edition (March 12, 2004).

[3] Leopold C., "Parallel and Distributed Computing. A survey of Models, Paradigms, and Approaches", Wiley Series on Parallel and Distributed Computing. Albert Zomaya Series Editor, 2001

[4] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. Thee Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Massachusetts, 1974

[5] S. Akl, "Parallel Computation. Models and Methods", Prentice-Hall, Inc., 1997.

[6] M. J. Flynn, Computer Architecture: Pipelined and Parallel Processor Design. Jones and Bartlett, 1995

[7] Baker M., R. Buyya. "Cluster Computing at a Glance". R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA,pp.3-47,1999.

[8] Zoltan Juhasz (Editor), Peter Kacsuk (Editor), Dieter Kranzlmuller (Editor), Distributed and Parallel Systems : Cluster and Grid Computing (The International Series in Engineering and Computer Science). Springer; 1 edition (September 21, 2004)

[9] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Suramonian and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation}", SIGPLAN Notices (USA), vol 28 N° 7, pp 1-12, 1993

[10] Valiant L.G.. *A Bridging Model for Parallel Computation*. Communications of the ACM, 33(8): 103-111, August 1990.

[11] C. Roig, A. Ripoll, M.A. Senar, F. Guirado, and E. Luque. Modelling Message-Passing Programs for Static Mapping. In Euromicro Workshop on Parallel and Distributed Processing (PDP'00). IEEE CS Press. USA, pp 229-236, 1999

[12] A. Kalinov, S. Klimov. Optimal Mapping of a Parallel Application Processes onto Heterogeneous Platform. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), April 2005

[13] A. Kalinov and S. Klimov, Multidimensional Static Block Data Decomposition for Heterogeneous Clusters, *in Proceedings of 5th PPAM,* Chenstohova, Poland, September 2003, LNCS 3019 , Springer, pp.907-914

[14] Y. Kishimoto and S. Ichikawa, "An Execution-Time Estimation Model for Heterogeneous Clusters", *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, 26-30 April 2004, Santa Fe, New Mexico, USA, CDROM/Abstracts Proceedings, IEEE Computer Society 2004.

[15] J. Cuenca, D. Gimenez, and J. Martinez, "Heuristics for Work Distribution of a Homogeneous Parallel Dynamic Programming Scheme on Heterogeneous Systems", *Procss of the 3rd International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (HeteroPar'04)*, July 5-8, 2004 Cork, Ireland, IEEE CS Press.

[16] M. Garey and D. Johnson. Computers and Intractability. W.H. Freeman and Co. S. Francisco, 1979

[17] J.C. Cunha, P. Kacksuk, and S.C. Winter. Parallel Program development for cluster computing. Nova Science Pub. Inc., Huntington, New York, 2001

[18] C. Roig, "Algoritmos de asignación basados en un nuevo modelo de representación de programas paralelos", Tesis Doctoral, Universidad Autónoma de Barcelona, 2002.

[19] C. Roig, A. Ripoll, J. Borrás, E. Luque. "Efficient Mapping for Message-Passing Applications Using the TTIG Model: A Case Study in Image Processing", Lecture Notes In Computer Science; Vol. 2131, Proceedings of the 8th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Pages: 370 – 377, 2001, ISBN:3-540-42609-4

[20] L. De Giusti, F. Chichizola, M. Naiouf, A. De Giusti. "Modelo TTIGHe. Resultados experimentales". Informe Técnico III-LIDI. 2006.