Universidad Autónoma de Barcelona
Escuela Técnica Superior de Ingeniería
Departamento de Informática

Algoritmos de asignación basados en un nuevo modelo de representación de programas paralelos

Memoria presentada por Concepció Roig Mateu para optar al grado de Doctora en Informática

Barcelona, Mayo de 2002

Algoritmos de asignación basados en un nuevo modelo de representación de programas paralelos

Memoria presentada por Concepció Roig Mateu para optar al grado de Doctora en Informática por la Universidad Autónoma de Barcelona. Trabajo realizado en el Departamento de Informática de la Escuela Técnica Superior de Ingeniería de la Universidad Autónoma de Barcelona, dentro del programa de doctorado "Arquitectura de Computadores y Procesamiento Paralelo", bajo la dirección de la Dra. Ana Ripoll Aracil.

Barcelona, Mayo de 2002

Vo. Bo. Directora de Tesis

Fdo. Ana Ripoll Aracil

A mi familia

Agradecimientos

En primer lugar quiero expresar mi más sincero agradecimiento a la Dra. Ana Ripoll por su esmerada dedicación a la dirección de esta tesis. Por sus constantes sugerencias para mejorar la calidad de los trabajos que hemos ido presentando, y por saberme transmitir en todo momento la ilusión necesaria para seguir adelante.

Al Dr. Emilio Luque, junto con todas las personas de la unidad de Arquitectura de Computadores y Sistemas Operativos de la UAB, por habernos integrado, tanto a mi como a mis compañeros de área de la Universitat de Lleida, en su grupo de investigación. Por habernos hecho sentir como uno más del grupo tanto a nivel de trabajo como de trato personal.

A las personas que han colaborado de forma más directa en el desarrollo de trabajos relacionados con esta tesis, Fernando Guirado, Miquel A. Senar y Xiao Yuan. A todos ellos les quiero agradecer su optimismo, su sentido del humor y al mismo tiempo su seriedad y disciplina para hacer las cosas bien.

A Francesc Solsona por haber compartido conmigo inquietudes, al haber coincidido en el periodo de confección y presentación del trabajo de tesis. A Francesc Giné por su compañerismo y apoyo siempre sincero. Y en general a todos mis compañeros de la Universitat de Lleida que me han brindado su amistad.

Gracias también a mi familia, porque entre todos me han dado el soporte necesario para facilitarme la realización de este trabajo. De una manera muy especial quiero expresar mi reconocimiento y cariño a mi esposo Joan y mis hijos Joan y Imma, porque a pesar de mi importante dedicación en los últimos años a la tesis, siempre han sabido percibir que ellos para mi son lo más importante.

Propósito y desarrollo de la memoria

El paralelismo ha sido considerado desde siempre como una técnica importante para aumentar las prestaciones de los sistemas informáticos. En la última década el interés por el paralelismo ha crecido enormemente, debido a la aparición de las arquitecturas distribuidas que facilitan la disponibilidad de los sistemas paralelos para todo tipo de usuarios.

En la mayoría de los estándar de programación actuales, los algoritmos paralelos que resuelven una aplicación son definidos por completo por el programador, que es quién decide el conjunto de tareas que forman la aplicación y las interacciones que se dan entre las mismas.

En el momento de ejecutar una aplicación paralela, el programador (o el usuario) se enfrenta además a decisiones importantes que le permitan reducir el tiempo de ejecución global, tales como **cuántos** procesadores ha de usar para ejecutar la aplicación y, dado un número de procesadores, **cómo** distribuir las tareas de la aplicación aprovechando al máximo su capacidad de concurrencia. Al problema de resolver la distribución de las tareas de una manera global se le conoce como el *problema del mapping* (que debe ser diferenciado del problema del *scheduling*, que consiste en determinar el orden de ejecución de las tareas por parte del procesador local).

En la literatura existen dos formas distintas de abordar el problema del mapping en función del conocimiento que se tiene de la aplicación. Cuando el comportamiento de la aplicación es conocido (o predecible) a priori, la asignación se realiza de forma estática (antes de la ejecución), y las tareas se ejecutan en el procesador asignado hasta que finalizan. Por el contrario,

cuando el comportamiento de la aplicación no es predecible, la asignación se realiza de forma dinámica, y las tareas pueden cambiar de procesador durante la ejecución. Como principal objetivo de la asignación dinámica se contempla, además de reducir el tiempo de ejecución de las aplicaciones, minimizar también el *overhead* asociado al proceso de asignación, puesto que afectará al tiempo de ejecución final.

En el presente trabajo nos centramos en el proceso de mapping estático. Para la realización de este proceso, el programa paralelo se suele representar mediante un modelo de grafo de tareas ponderado, que resume las características más relevantes estimadas del comportamiento de la aplicación. En función del tipo de aplicación, en la literatura se utilizan principalmente dos modelos de grafo. Para aplicaciones cuyas tareas se comunican únicamente por el principio y el final, el modelo, denominado TPG (Task Precedence Graph), refleja las comunicaciones y precedencias entre las tareas y el orden parcial de ejecución de las mismas. Cuando se trata de aplicaciones cuyas tareas tienen comunicaciones en cualquier punto, e incluso comunicaciones bidireccionales, en la literatura se utiliza un modelo simplificado, denominado TIG (Task Interaction Graph), en el que no se contemplan las precedencias y se asume que todas las tareas pueden ser simultáneas.

Ahora bien, en los entornos actuales de paso de mensajes, el programador no está sujeto a ninguna restricción en cuanto a la ubicación de las primitivas de comunicación dentro de las tareas. Además, debido al tipo de problemas que se resuelven computacionalmente, existe en los últimos años un creciente interés en el desarrollo de aplicaciones formadas por un conjunto de tareas que realizan distintas funciones y que coordinan su ejecución mediante intercambios de información en cualquier punto dentro de las mismas. Un ejemplo de ello son las aplicaciones de simulaciones multidisciplinares, que construyen un modelo complejo a partir de distintos módulos. En dichas aplicaciones, cada módulo consta de una o varias tareas distintas con un patrón de comunicaciones arbitrario y en consecuencia con distinta posibilidad de ejecución simultánea.

Para modelar el comportamiento de las aplicaciones con paralelismo de tareas, con un patrón de interacciones entre tareas arbitrario, se propone un

nuevo modelo de grafo, denominado Temporal Task Interaction Graph (TTIG). Dicho modelo incluye un nuevo parámetro, denominado grado de paralelismo, que indica la máxima capacidad de concurrencia de las tareas que se comunican, en función de las dependencias provocadas por dichas comunicaciones. Este parámetro será de gran utilidad, como mostraremos en las estrategias de asignación.

A partir del comportamiento obtenido de la aplicación, se propone un mecanismo para determinar las cotas teóricas mínima y máxima sobre el número de procesadores necesario para realizar su ejecución en un tiempo mínimo.

A partir del modelo TTIG se definen nuevas políticas de mapping de distintas complejidades que realizan las asignaciones de tareas teniendo en cuenta la posibilidad de concurrencia entre las mismas que indica el grado de paralelismo.

En los entornos actuales de paso de mensajes PVM y MPI, la política de mapping que se usa por defecto es una distribución de las tareas basada en el orden de activación de las mismas. Dada la simplicidad de este mecanismo, dichos entornos se mejoran integrando un proceso automático para la extracción del grafo TTIG y para aplicar una política de mapping basada en dicho modelo.

La presente memoria se organiza en los siguientes capítulos.

- Capítulo 1. Se enmarca el problema del mapping y se describen las alternativas clásicas. Se exponen los tipos de grafo utilizados para modelar la aplicación y se estudia la adecuación de cada una de ellos según las características de la aplicación modelada. Se resumen los algoritmos de mapping más representativos desarrollados para cada uno de los modelos clásicos, y las herramientas de mapping existentes en la literatura.

- Capítulo 2. En este capítulo se define formalmente el nuevo modelo de grafo TTIG propuesto en el presente trabajo, y se desarrolla el conjunto de pasos a seguir para generar dicho modelo a partir de una aplicación de paso de mensajes.

- Capítulo 3. Se analizan las características inherentes al nuevo modelo TTIG, que facilitan la toma de decisiones más eficientes en el proceso de mapping. A partir del modelo de comportamiento se desarrolla un procedimiento para calcular un valor de las cotas teóricas mínima y máxima del número de procesadores, que permiten ejecutar la aplicación en un tiempo mínimo.

- Capítulo 4. Se presentan dos estrategias de asignación de tareas, TASC y MATE, para el modelo TTIG basadas en distintos objetivos. Se analiza la robustez de la política para la estrategia MATE, en lo que concierne a la sensibilidad de las asignaciones a pequeñas variaciones en la estimación de los parámetros asociados al modelo de entrada.

- Capítulo 5. Para tener una valoración de la bondad de las políticas TASC y MATE, se realiza la comparación del tiempo de ejecución de sus asignaciones respecto el tiempo de ejecución de la asignación óptima. Puesto que la generación de la asignación óptima requiere una elevada complejidad, dicho estudio se realiza para un conjunto de grafos TTIG de dimensiones reducidas.

- Capítulo 6. La efectividad de las políticas desarrolladas para el modelo TTIG se analiza para un conjunto de programas de paso de mensajes en un cluster de PCs. En este caso se realiza la comparación de las políticas TASC y MATE respecto a los algoritmos de asignación basados en los modelos clásicos de la literatura, puesto que el número de tareas de las aplicaciones no hace viable la comparación con el óptimo.

- Finalmente se presentan las principales conclusiones que se derivan del presente trabajo, junto con las contribuciones a que a dado lugar el desarrollo del mismo. Así mismo, se indicarán las lineas que quedan abiertas a partir de los resultados obtenidos.

Estos capítulos se complementan con los siguientes apéndices.

- Apéndice A. Se muestra el procedimiento desarrollado para extraer el comportamiento temporal de las aplicaciones con paso de mensajes, que

conduce a la generación del modelo TTIG. Además se da la relación de las primitivas especificadas en el lenguaje intermedio utilizado que se usan para definir dicho comportamiento.

- Apéndice B. Se justifican las razones que nos han llevado a incluir en el modelo TTIG el valor de grado de paralelismo sólo entre tareas adyacentes y no incluirlo también entre tareas no adyacentes.

- Apéndice C. Se resumen las principales facilidades que proporcionan las herramientas desarrolladas en nuestro grupo, y que han sido utilizadas en la tesis para llevar a cabo distintos aspectos de la experimentación.

- Apéndice D. Se muestran las tablas con los tiempos de ejecución obtenidos en la realización de las fases experimentales del trabajo.

- Apéndice E. Se realiza un estudio comparativo de las soluciones de una y dos fases para los algoritmos de scheduling de grafos TPG. En concreto se analiza la mejora que proporcionan los algoritmos basados en el modelo TTIG frente los algoritmos basados en el TIG, al ser aplicados en la fase de cluster-mapping (segunda fase) de grafos TPG.

Índice General

1 Modelando el comportamiento de las aplicaciones paralelas para la asignación estática **1**

1.1 El proceso de paralelización 3

1.2 Modelos estáticos de comportamiento 6

1.2.1 Grafo de Precedencia de Tareas 8

1.2.2 Grafo de Interacción de Tareas 9

1.2.3 Aplicabilidad de los modelos clásicos 10

1.2.4 Propuesta de un nuevo modelo 13

1.3 El problema del mapping 16

1.3.1 Algoritmos de mapping para Grafos de Precedencia de Tareas (TPG) 17

1.3.2 Algoritmos de mapping para Grafos de Interacción de Tareas (TIG) 20

1.3.3 Algoritmos de mapping basados en el modelo TTIG . . . 23

1.3.4 Herramientas de mapping 23

1.4 Objetivos del trabajo 25

2 El modelo TTIG (Temporal Task Interaction Graph) **28**

2.1 Definición del modelo TTIG 28

2.2 Proceso de generación del modelo TTIG 30

2.2.1 Generación del fichero de traza 31

2.2.2 Extracción del Grafo de Flujo Temporal 32

2.2.3 Cálculo de los parámetros asociados al modelo TTIG . . 36

2.2.4 Verificación del grado de paralelismo de los lazos 42

2.3 Complejidad computacional asociada al modelo TTIG 44

3 Influencia del modelo de comportamiento en el proceso de mapping **46**
3.1 Bondad y sensibilidad de la asignación 46
3.1.1 Influencia del grado de paralelismo 47
3.1.2 Sensibilidad de la asignación a las variaciones en la estimación de los costes asociados al modelo 48
3.1.3 Bondad de las asignaciones basadas en la evaluación de las tareas más dependientes 50
3.2 Utilización del modelo de comportamiento para determinar una cota sobre el número de procesadores 52
3.2.1 Cálculo del valor de cota mínima 56
3.2.2 Cálculo del valor de cota máxima 60

4 Algoritmos de mapping basados en el modelo TTIG **62**
4.1 Fundamentos de la estrategia de asignación 64
4.1.1 Valor de nivel de los nodos 65
4.1.2 Evaluación de los tiempos de ejecución de las tareas adyacentes según sus dependencias 67
4.2 Estrategia de asignación basada en explotar la concurrencia de las tareas adyacentes (TASC) 69
4.2.1 Fase 1. Definición de la lista de prioridades 70
4.2.2 Fase 2: Asignación de tareas a procesadores 71
4.2.3 Análisis de la complejidad 76
4.3 Estrategia de asignación priorizando las adyacencias más dependientes (MATE) 77
4.3.1 Algoritmo de asignación 79
4.3.2 Análisis de la complejidad 83
4.4 Robustez de la política de mapping 85
4.4.1 Valoración global del grafo TTIG 86
4.4.2 Sensibilidad de la asignación frente a variaciones en los parámetros de entrada 90

5 Efectividad de las políticas de mapping respecto del óptimo **94**
5.1 Generación de la asignación óptima 95

5.2 Marco para la experimentación 97
5.3 Desviación respecto a la asignación óptima 100
5.3.1 Grafos con comportamiento arbitrario 101
5.3.2 Grafos con comportamiento homogéneo 107
5.4 Adecuación de las cotas teóricas en relación a la asignación óptima 109
5.4.1 Validación de las cotas teóricas 110
5.4.2 Grado de ocupación de los procesadores 112
5.4.3 Desviación de las políticas de mapping en el valor de cota 115

6 Estudio comparativo de las estrategias de asignación en un entorno cluster 119
6.1 Descripción de las aplicaciones utilizadas 120
6.1.1 Aplicaciones sintéticas 120
6.1.2 Aplicación real 122
6.2 Descripción de las políticas de mapping 124
6.3 Resultados experimentales 127
6.3.1 Aplicaciones con comportamiento arbitrario 129
6.3.2 Aplicaciones con comportamiento homogéneo 132
6.3.3 Aplicación BASIZ 133
6.4 Estudio de la escalabilidad 136
6.4.1 Resultados experimentales 138

Conclusiones y principales contribuciones 145

A Ejemplo de generación del comportamiento temporal a partir de la traza 151

B Estudio de porqué el grado de paralelismo se calcula sólo entre tareas adyacentes 156
B.0.2 Grado de paralelismo entre tareas no adyacentes considerando el subgrafo aislado 156
B.0.3 Grado de paralelismo a partir de la simulación global del grafo 158

C Herramientas utilizadas en el proceso automático de mapping 161
C.1 Herramienta CASOS 162
C.2 Herramienta AMEEDA 163
C.3 Herramienta ESPPADA 165

D Tablas de tiempos de ejecución de las aplicaciones 168
D.1 Aplicaciones ejecutadas en el cluster 168
D.1.1 Aplicaciones sintéticas 169
D.1.2 Aplicación real 171
D.2 Ejecución en el entorno de simulación ESPPADA 173

E Aplicación del modelo TTIG en las estrategias de asignación basadas en el modelo TPG 175
E.1 Algoritmos de scheduling y cluster-mapping 176
E.2 Comparación de las estrategias de mapping de una y dos fases para grafos TPG 180
E.3 Tablas de tiempos de ejecución 186

Bibliografía 190

Capítulo 1

Modelando el comportamiento de las aplicaciones paralelas para la asignación estática

Tradicionalmente, la computación paralela se ha considerado como un campo de estudio importante para aumentar la velocidad de cómputo de los sistemas informáticos. En la última década su interés ha crecido enormemente debido a la disponibilidad de los sistemas paralelos comerciales que ponen a nuestro alcance la posibilidad de disponer de computadores de altas prestaciones a un precio razonable.

El paralelismo puede ser explotado a distintos niveles en los sistemas computadores, tal como se deduce de la siguiente definición de Almasi y Gottlieb [AG89].

"A parallel computer is a collection of processing elements that communicate and cooperate to solve large problems fast."

Esta sencilla definición plantea las siguientes cuestiones importantes a resolver [CS99]:

(a) Qué número de elementos de procesamiento (EP), debe de contener el sistema paralelo.

(b) Qué prestaciones debe ofrecer cada EP individual.

(c) De qué forma deben comunicarse y cooperar los distintos EPs.

(d) Cuáles han de ser las herramientas software a utilizar para explotar la capacidad de paralelismo del sistema.

Es evidente pues, que en el rendimiento final de un sistema paralelo inciden diversos factores que afectan a distintos niveles de diseño. Por un lado, el uso de tecnología más avanzada junto con soluciones de diseño más eficientes, permiten la construcción de elementos de procesamiento más potentes y redes de interconexión más rápidas. Por otro lado, las técnicas software facilitan el desarrollo de las aplicaciones paralelas, de tal forma que se explote la capacidad de paralelismo del sistema.

Dado un sistema paralelo con unas características concretas, existen aspectos clave a resolver, referentes al desarrollo de aplicaciones, que son de vital importancia para que el paralelismo del sistema tenga un impacto positivo en el rendimiento. Tales aspectos hacen referencia a la descomposición del cómputo y los datos en procesos o tareas[1] más pequeños y a la posterior asignación de dichas tareas en los EPs del sistema. La solución de estos aspectos tiene inherente una gran complejidad, y es difícil estimar el rendimiento de una determinada solución previo a su implementación real.

Generalmente la definición de las tareas que forman una aplicación así como la división de los datos a procesar entre las mismas, las decide el programador. El proceso de asignación de tareas a procesadores puede ser también decidido por el programador o bien puede realizarse mediante el uso de algoritmos automáticos diseñados para este fin.

A continuación se desarrollan los pasos involucrados en el proceso de paralelización de las aplicaciones, y se resumen las características de los modelos de aplicación y políticas de asignación existentes en la literatura, para ubicar la aportación de nuestro trabajo dentro de este ámbito.

[1] En este contexto los términos tarea y proceso se utilizan como sinónimos

1.1 El proceso de paralelización

Independientemente de las características del sistema disponible, la solución de un problema en paralelo requiere del desarrollo de las siguientes fases: detección del paralelismo, definición del grafo de tareas y asignación de tareas a procesadores. Estas fases se ilustran en la Figura 1.1 y el cometido de cada una de ellas se desarrolla a continuación [Fos94].

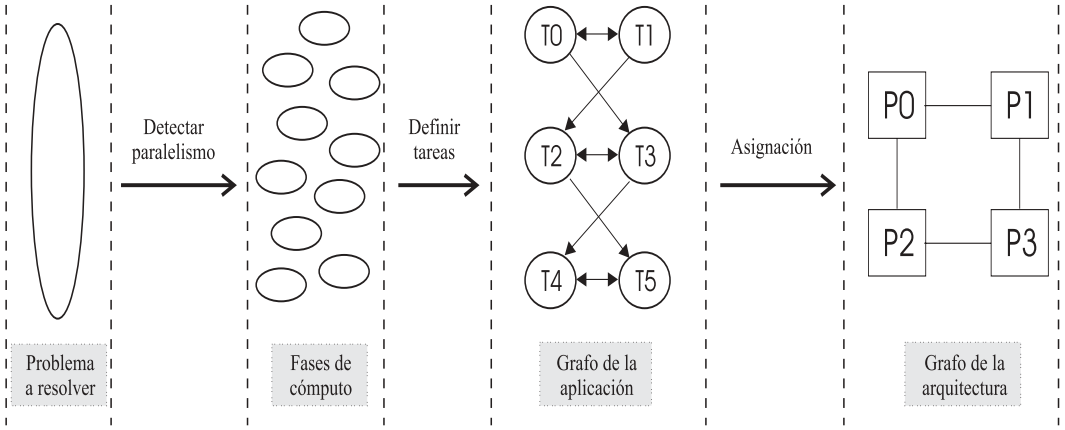

Figura 1.1: Etapas del proceso de paralelización.

(a) *Detección del paralelismo.*

Detección y descomposición de las actividades secuenciales que pueden ser realizadas en paralelo, con sus interdependencias. Denotamos estas actividades como *fases de cómputo*, siendo éstas 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 máximo número de fases de cómputo disponibles a lo largo del tiempo proporciona la cota superior sobre el número de procesadores a utilizar para ejecutar una aplicación paralela.

Así pues, el principal objetivo que se persigue en la etapa de descomposición, no es tan solo la detección de las actividades que pueden hacerse en paralelo, si no realizar 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. Estos objetivos son difíciles de conseguir y, tal como pasa en el cómputo secuencial, el diseño de un buen algoritmo paralelo está básicamente supeditado a la experiencia del programador, y diferentes decisiones en este punto pueden llevar a distintas soluciones de rendimiento.

(b) *Definición del grafo de la aplicación.*

Las fases de cómputo se agrupan en tareas o procesos. Cada tarea realiza una parte de las actividades del algoritmo paralelo y constituye en sí misma un programa. Se definen también las interacciones entre tareas, que se llevan a cabo mediante un mecanismo de comunicación. En el proceso de definición del grafo deben resolverse problemas importantes tales como:

1. Granularidad de las tareas. Cantidad de trabajo que realiza cada una de las tareas que forma la aplicación. La granularidad es de tipo fino cuando el grafo está formado por gran cantidad de tareas que ejecutan una pequeña parte del cómputo global. En caso contrario, nos encontramos con aplicaciones de granularidad gruesa.
2. Descomposición de los datos a tratar por cada una de las tareas. Este es un aspecto fundamental a tratar en los modelos de programación paralela con memoria distribuida donde las comunicaciones suponen un coste de tiempo importante.
3. Organización de las comunicaciones entre tareas, determinando 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. Esto nos lleva a distinguir entre: (a) aplicaciones con paralelismo de datos (SPMD: Single Program

Multiple Data), donde cada tarea ejecuta la misma operación en paralelo sobre distintos elementos del conjunto de datos, (b) aplicaciones con paralelismo de tareas (MPMD: Multiple Program Multiple Data), donde las tareas realizan funciones distintas sobre distintos datos y, (c) aplicaciones con paralelismo mixto que integran tanto paralelismo de tareas como de datos [BH98].

(c) *Asignación de tareas a procesadores.*

Las tareas de la aplicación se asignan a los procesadores del sistema con el objetivo de minimizar el tiempo de ejecución final. Dicha asignación se lleva a cabo mediante un proceso denominado de *scheduling*, que determina *donde* y *cuando* 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 decidir *donde* se ejecutan las tareas se denomina *mapping*.

En ambos casos, la asignación puede hacerse de forma dinámica durante la ejecución, o bien estática previa a la ejecución. Para la asignación dinámica se realizan escasas o nulas estimaciones acerca del comportamiento de la aplicación, y las decisiones se basan en parámetros que reflejan la situación del sistema en cada instante de tiempo. Como principal objetivo de la asignación dinámica se contempla, además de minimizar el tiempo de ejecución de las aplicaciones, minimizar también el *overhead* asociado al proceso de asignación, puesto que puede afectar al rendimiento del sistema.

En este trabajo nos centramos en el proceso de asignación estática, a la que nos referiremos a lo largo del trabajo como mapping. Puesto que la ubicación de las tareas a los procesadores se realiza de forma previa a la ejecución, el *overhead* asociado al proceso de mapping no tiene efectos durante la ejecución, ya que cada procesador conoce de antemano las tareas que ha de ejecutar.

Para la realización del mapping, el programa paralelo se representa mediante un modelo de grafo ponderado, que resume las características más relevantes estimadas del comportamiento del programa, tales como el tiempo de

cómputo de las tareas, comunicaciones entre tareas, dependencias de datos y requerimientos de sincronización [CL95].

Es evidente que las estrategias de mapping aplicables generarán unos resultados esperados en el tiempo de ejecución siempre y cuando los parámetros estimados en el modelo guarden relación con los del programa real. Es decir, no será necesario que los costes estimados sean los mismos que los del programa real, pero sí que la relación entre los valores de cómputo y de comunicación de las tareas del grafo guarden la misma relación que la existente en el programa real. Cuando esto sea difícil de conseguir debido a que los pesos del programa sufren grandes variaciones dependiendo de los datos de entrada, el mapping estático no será adecuado y habrá que apostar por alguna técnica dinámica que haga la asignación en función de los valores que se dan en tiempo de ejecución [HKL00] [CKW01].

Existen distintos modelos de grafo para representar el comportamiento de las aplicaciones paralelas que clásicamente se han venido utilizando para la resolución del mapping estático. En la siguiente sección se estudia la forma de representar las aplicaciones mediante dichos modelos.

1.2 Modelos estáticos de comportamiento

Desde el punto de vista del programador, el desarrollo de las aplicaciones paralelas se puede llevar a cabo según dos modelos de cómputo [Pel98]:

(a) Modelo *implícito*, en el que el programador escribe sus programas mediante un lenguaje de programación secuencial, sin ocuparse de definir ninguna estrategia 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 de tareas que modela la aplicación y de realizar su ejecución de forma totalmente transparente al usuario.

(b) Modelo *explícito*, según el cual 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, dentro de la cual se distinguen los modelos paralelos explícitos de alto nivel

y de bajo nivel. En los primeros el programador define únicamente el conjunto de tareas que componen la aplicación junto con las actividades que realiza cada una de ellas. Ejemplos de lenguajes de programación que pertenecen a esta categoría son HPF (High Performance Fortran) [KLSZ94], Dataparallel C [HQ91] y FortranD [HKK+92]. Cuando el programador, además de definir las tareas, se encarga del manejo de sus requerimientos de sincronización y comunicación nos encontramos en un modelo de programación explícito de bajo nivel. Las librerías de paso de mensajes PVM (Parallel Virtual Machine) [GBD+94] y MPI (Message Passing Interface) [For94] pertenecen a esta categoría. En éstas, la comunicación y sincronización entre tareas se lleva a cabo mediante primitivas explícitas que se incluyen dentro del cómputo secuencial. El uso de los entornos de programación PVM y MPI, se ha extendido últimamente debido a las facilidades que proporcionan para la programación de aplicaciones paralelas en entornos distribuidos formados por redes de estaciones de trabajo o PCs.

En este trabajo nos centramos en modelar el comportamiento de las aplicaciones con paralelismo explícito definidas mediante paso de mensajes para el propósito del mapping estático. Puesto que los entornos actuales de paso de mensajes no imponen ninguna restricción respecto al número de tareas que forman la aplicación, ni a su granularidad ni a su estructura de interconexión, será necesario considerar que estos aspectos pueden ser definidos por el programador de forma arbitraria. Por lo tanto, *el grafo que modela la aplicación debe ser capaz de reflejar cualquier patrón de comunicación entre tareas y capturar el comportamiento temporal de las mismas.*

Clásicamente en la literatura se han utilizado dos modelos de grafo para representar estáticamente las aplicaciones paralelas. El Grafo de Precedencia de Tareas (TPG: Task Precedence Graph) y el Grafo de Interacción de Tareas (TIG: Task Interaction Graph). Estos dos grafos reflejan el comportamiento de la aplicación de forma distinta, y la idoneidad de utilizar uno u otro dependerá de la estructura de tareas de la aplicación a modelar, tal y como vemos en los siguientes apartados.

1.2.1 Grafo de Precedencia de Tareas

Mediante el Grafo de Precedencia de Tareas (TPG: Task Precedence Graph), la aplicación se representa como un grafo dirigido acíclico $G = (N, E)$, donde N es el conjunto de nodos del grafo, cada uno representando una tarea, y E es el conjunto de arcos entre tareas adyacentes. Dichos arcos representan tanto las comunicaciones como las relaciones de precedencia entre tareas, e indican un orden parcial en la ejecución de las mismas.

A cada tarea $Ti \in N$ se le asocia un valor no negativo $w(Ti)$, que representa su tiempo de cómputo estimado, y a cada arco $(Ti, Tj) \in E$ se le asocia un valor $c(Ti, Tj)$, que representa el volumen de comunicación que se transmite de Ti a Tj durante la ejecución. La Figura 1.2 ilustra un ejemplo de grafo TPG de seis tareas {T0,..,T5}. Como ejemplo de orden parcial en la ejecución marcado por las precedencias, puede verse en el grafo que las tareas T0 y T1 son tareas iniciales y pueden empezar su ejecución desde el primer momento, en cambio la tarea T3 debe esperar a que T0 y T1 hayan finalizado y le hayan transmitido todos los datos para empezarse a ejecutar.

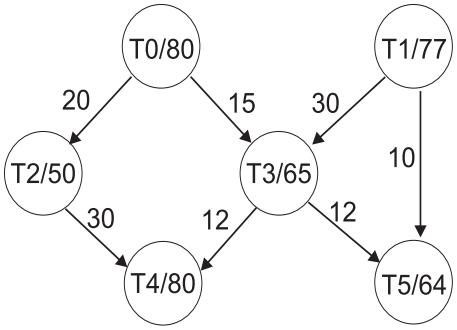

Figura 1.2: Grafo de Precedencia de Tareas.

Puesto que en el TPG se representan las relaciones de precedencia de forma explícita, este grafo 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 los tenemos en el ámbito de las aplicaciones numéricas como las de cálculo matricial y transformadas de Fourier.

1.2.2 Grafo de Interacción de Tareas

Mediante el Grafo de Interacción de Tareas (TIG: Task Interaction Graph), el programa paralelo se representa 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 a las aristas, representando los tiempos de cómputo de las tareas y los volúmenes de comunicación que se intercambian respectivamente. El TIG no incluye ninguna información relativa al orden de ejecución de las tareas y por lo tanto los autores de la literatura asumen que todas las tareas pueden ejecutarse concurrentemente.

El grafo de la Figura 1.3 es un ejemplo de TIG de cuatro tareas {T0,..,T3}, en el que, como puede verse, de cada tarea solo se conoce el tiempo de cómputo y cuales son sus adyacentes con el volumen de comunicación que se transmiten entre ellas, pero no el instante en que se efectúa esta transmisión.

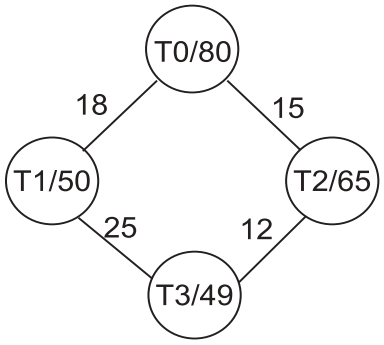

Figura 1.3: Grafo de Interacción de Tareas.

El modelo TIG es menos refinado que el TPG, puesto que no captura información temporal relativa al orden de ejecución de las 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 [KSK97].