1.2.3 Aplicabilidad de los modelos clásicos

En los entornos de programación de paso de mensajes actuales, el programador define *qué* cómputo debe realizar cada una de las tareas de la aplicación, y *cuándo* y *dónde* una tarea intercambia información con sus adyacentes. En el caso general, las aplicaciones con paso de mensajes están formadas por un conjunto de tareas con granularidad gruesa, que realizan una determinada función sobre un conjunto de datos propio. Las transferencias de información entre tareas se llevan a cabo mediante primitivas de comunicación punto a punto de envío y de recepción. De los grafos clásicos estudiados, el adecuado para modelar estáticamente la aplicación, dependerá de la estructura de tareas definida por el usuario. Estudiamos con un ejemplo sencillo la aplicabilidad de cada uno de los modelos desde el punto de vista del patrón de interacciones entre tareas de cada uno.

El ejemplo que consideramos es el algoritmo de Floyd [Flo62], que calcula la distancia mínima entre cualquier par de nodos de un grafo. Dado un grafo de n nodos y una matriz de distancias A(nxn) 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 $\rightarrow$ Ak $\rightarrow$ Aj, donde Ak es un nodo intermedio en el camino Ai $\rightarrow$ Aj. La Figura 1.4 muestra el pseudocódigo para la versión secuencial del algoritmo de Floyd.

```
A(nxn)=matriz de distancia iniciales para cada par de nodos.
for k=0 to (n-1)
   for i=0 to (n-1)
      for j=0 to (n-1)
         if Aij>(Aik+Akj) then Aij=Aik+Akj;
```

Figura 1.4: Algoritmo de Floyd secuencial.

Una solución paralela del algoritmo de Floyd consiste en dividir la matriz inicial en m particiones de n/m columnas contiguas de A, y aplicar Floyd a

cada partición separadamente [JS87]. En este caso, el programa estará formado por una tarea que denominamos TR, que descompone la matriz en varias particiones y compone el resultado final, y diversas tareas que denominamos Si que calculan el algoritmo de Floyd para su partición de datos, tal como se ilustra en el pseudocódigo de la Figura 1.5.

Tarea TR

```
leer matriz A;
activar m tareas Si
for i=1 to m
   componer partición;
   enviar partición a la tarea Si;
end_for
for i=1 to m recibir resultados;
componer matriz de distancias mínimas;
```

Tarea Si

```
recibir partición;
for k=0 to n-1
   if k∈mi partición
      enviar columna k al resto de tareas Si;
      calcular Floyd;
   else
      recibir columna k de su propietaria;
      calcular Floyd;
   end_if
end_for;
enviar resultado a TR;
```

Figura 1.5: Versión paralela del algoritmo Floyd.

Para cada iteración de k, cada tarea Si procesa la parte de matriz que le corresponde, recorriéndola mediante los índices i y j. Como el cálculo de cada

Aij se realiza en base a los valores intermedios de Aik, a cada iteración de k la tarea que posee la columna k de la matriz debe enviarla a las demás. Al finalizar el cómputo, cada tarea Si envía los resultados a la tarea TR, quién compone la matriz de distancias mínimas final.

Supongamos como ejemplo el programa Floyd paralelo que procesa una matriz de distancias de 10 elementos en dos particiones. En este caso, el programa constará de tres tareas: la tarea TR para generar las dos particiones y componer el resultado, y las tareas S1 y S2 que procesarán cada una la mitad de las columnas de la matriz de distancias. El patrón de interacciones que corresponde a estas tres tareas es el que se indica en la Figura 1.6(a), donde se ve que las dos tareas S1 y S2 se comunican con la tarea TR para recibir su partición y enviarle el resultado, a la vez que S1 y S2 se comunicarán entre ellas para irse enviando las sucesivas columnas de la matriz.

De los modelos clásicos, la opción que se ajusta a las características de la aplicación sería utilizar el modelo TIG, ya que las tareas S1 y S2 tienen interacciones reiteradas en puntos internos de las mismas durante su ejecución. La Figura 1.6(b) muestra el patrón de interacción entre tareas para este ejemplo que corresponde al modelo TIG.

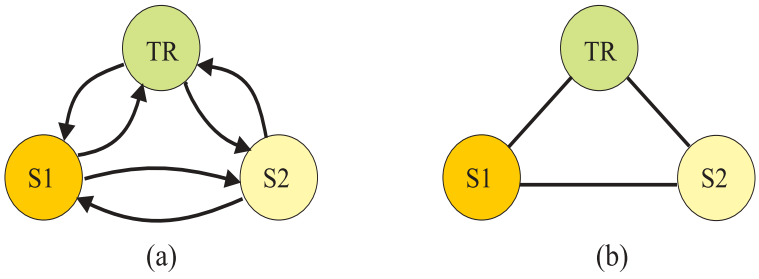

Figura 1.6: (a) Estructura de interacción de tareas para el algoritmo paralelo de Floyd. (b) Modelo TIG.

Si el mismo programa paralelo se quiere modelar usando un grafo TPG, habrá que abrir el bucle y subdividir las tareas S1 y S2 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 tareas por el principio y el final de las mismas, tal como se supone en este modelo. La Figura 1.7 muestra el correspondiente grafo TPG, donde cada subtarea S1.i y

S2.i corresponde a una iteración del bucle de las tareas S1 y S2 respectivamente. Como podemos observar, en este caso el hecho de reflejar explícitamente las precedencias hace que el número de tareas del grafo TPG dependa del número de nodos de la matriz de entrada, que para este ejemplo habíamos supuesto que era de 10 nodos.

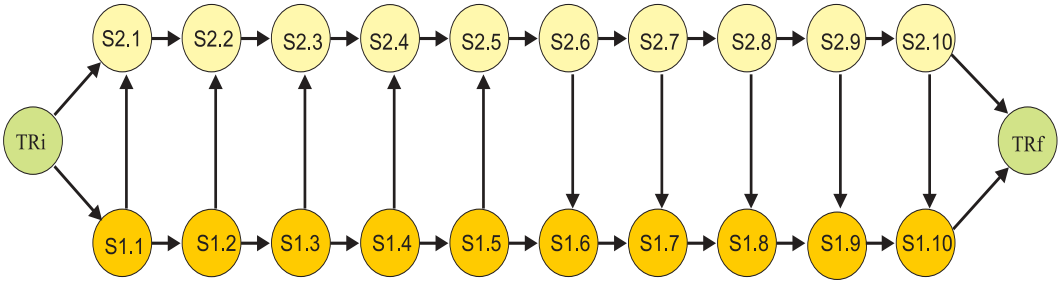

Figura 1.7: Estructura de interacción de tareas basada en el modelo TPG, para el algoritmo paralelo de Floyd.

Respecto a las dos posibles formas de modelar las interacciones entre tareas del mismo programa, usando los grafos TIG o TPG, es preciso hacer las siguientes observaciones. El TIG usa como modelo un grafo con las mismas tareas que las definidas por el programador, pero no refleja ninguna información acerca del orden de ejecución de las mismas. En el caso del TPG si que 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.

Vemos pues 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.

1.2.4 Propuesta de un nuevo modelo

En este trabajo se define un nuevo modelo de grafo de tareas denominado TTIG (Temporal Task Interaction Graph), que representa las aplicaciones con

la estructura de interconexión de tareas definida por el programador. Es decir, para el algoritmo de Floyd propuesto como ejemplo, el grafo TTIG representaría la aplicación con la estructura de tareas mostrada en la Figura 1.6(a). El grafo TTIG está formado por el mismo número de nodos que las tareas que componen el programa, e incluye además de los costes de cómputo y comunicación, el máximo grado de concurrencia entre las tareas adyacentes (i.e. entre las tareas que se comunican), en virtud de sus 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 clásicos.

Para ver la conveniencia de la inclusión de este nuevo parámetro, consideramos un programa con dos tareas T0 y T1 como el que se esquematiza en la Figura 1.8. El cómputo global estimado de cada tarea es de 80 unidades de tiempo, y existe una comunicación de T0 a T1 llevada a cabo mediante las primitivas de envío y recepción *send* y *rcv* respectivamente, donde la cantidad de datos está indicada como *vol*.

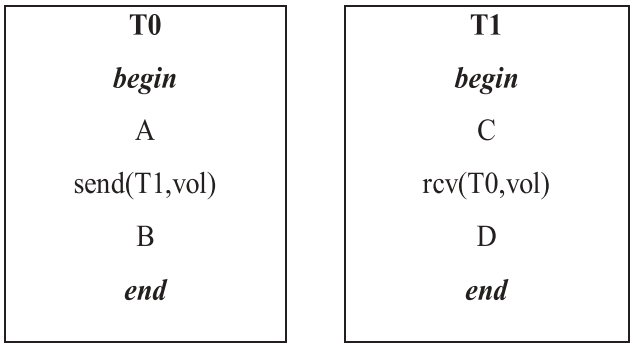

Figura 1.8: Programa de paso de mensajes con dos tareas T0 y T1.

En función de la posición relativa de las primitivas de envío y recepción dentro de las dos tareas, su comportamiento temporal será completamente distinto. En la Figura 1.9 se ilustran dos situaciones de diagramas de tiempo para distintos valores de tiempo de las fases de cómputo que se realizan antes y después de cada primitiva de comunicación. En dichos diagramas no se contempla el tiempo involucrado en la transmisión de datos, si no simplemente la dependencia temporal provocada por la existencia de las primitivas

de comunicación. En el caso (a) la tarea T1 debe parar su ejecución durante 10 unidades de tiempo, en espera de recibir los datos de la tarea T0, pero globalmente las dos tareas pueden llevar a cabo una ejecución prácticamente paralela. En la situación (b), el tiempo en que la tarea T1 debe estar esperando es de 70 unidades de tiempo, con lo cual, la posibilidad de paralelismo entre las dos tareas se reduce en gran manera, y sus dependencias les provocan una ejecución prácticamente secuencial.

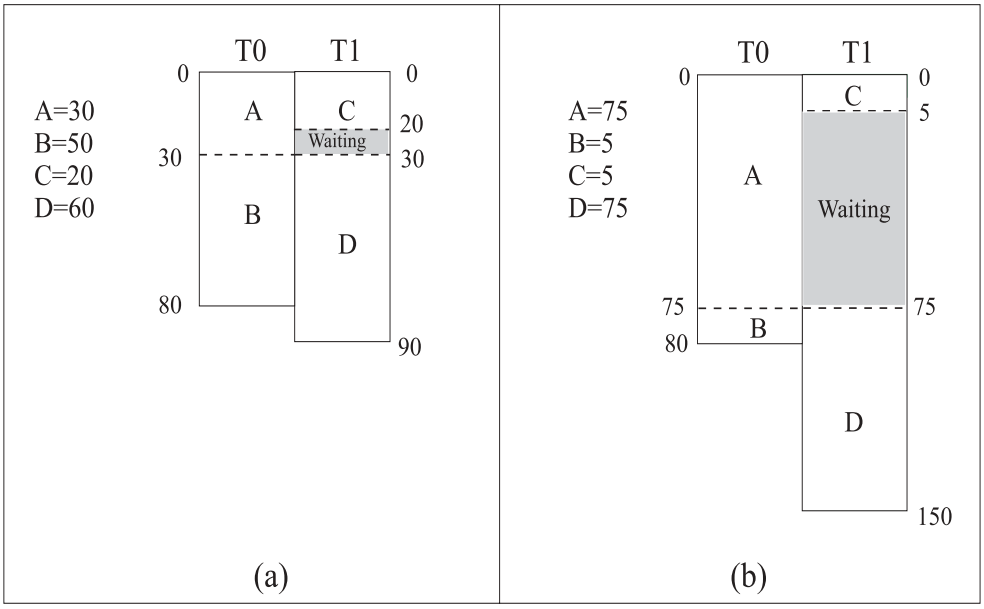

Figura 1.9: Situaciones distintas en el comportamiento temporal de las tareas T0 y T1.

Estas dos situaciones de comportamiento completamente distintas se modelan de la misma forma usando el modelo clásico TIG que corresponde al de la Figura 1.10; en cambio, la posibilidad de concurrencia entre las tareas adyacentes es una información fundamental a tener en cuenta en el proceso 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 posibilidad de concurrencia pueden ser asignadas a distintos procesadores.

Así, las dos situaciones reflejadas en la Figura 1.9 se modelarán de forma distinta, y en cada caso se incluirá en el TTIG un índice distinto de grado de

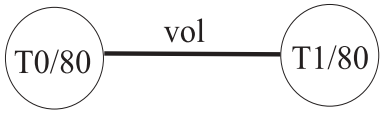

Figura 1.10: Modelo TIG para el esquema de programa de la Figura 1.8.

paralelismo, que para el caso (a) indicará que las dos tareas son potencialmente muy paralelas, mientras que en el caso (b) indicará que se trata de tareas poco paralelas debido a sus dependencias.

1.3 El problema del mapping

Una vez definidos los grafos que modelan la aplicación y la arquitectura, el problema del mapping se resuelve mediante algún algoritmo que establece un mecanismo automático para realizar la asignación de tareas a los procesadores. Este es un problema NP-completo, debido a la existencia de muchos factores a tener en cuenta que directa o indirectamente influyen en el tiempo de ejecución de un programa [GJ79].

En función de la estrategia seguida por las distintas políticas propuestas en la literatura, los algoritmos de mapping estático pueden clasificarse en los dos grupos siguientes [CKW01]:

- *Optimo.* En el caso que se consideren determinadas restricciones en la estructura de los programas y/o en el sistema paralelo, se han desarrollado algoritmos de mapping que proporcionan la solución óptima en tiempo polinomial [KA97] [Set76] [ST85]. Cuando no se consideran restricciones al problema inicial, la solución óptima sólo puede abordarse cuando el número de configuraciones posibles es lo suficientemente bajo, como por ejemplo que haya pocas tareas a ser asignadas a un número de procesadores pequeño. En caso contrario la solución óptima no puede llevarse a cabo debido a la explosión combinatoria en el número de posibles soluciones.

- *Heurístico.* La mayoría de las estrategias de mapping propuestas en la literatura se basan en técnicas heurísticas que utilizan suposiciones rea-

listas a priori del sistema y del algoritmo paralelo. Dichos algoritmos producen soluciones sub-óptimas en un tiempo de ejecución más razonable comparado con las estrategias óptimas. El tipo de estrategia utilizada depende a la vez del modelo usado [NT93]. Cuando las relaciones de precedencia entre tareas se suponen conocidas y se capturan en el modelo, como es el caso del TPG, el objetivo del algoritmo de mapping será el de minimizar el tiempo de ejecución final. En el caso que el modelo no capture dichas precedencias, como pasa en el TIG, el algoritmo de mapping basará su estrategia en la optimización de una determinada función de coste previamente definida. A continuación se resumen las principales características de los algoritmos de asignación basados en ambos modelos.

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

La aplicación se modela como un grafo TPG que tiene asociados tiempos de cómputo para las tareas y volúmenes de comunicación entre tareas adyacentes. La solución de este problema de mapping, (denominado como *scheduling*), consiste en determinar la colocación de cada tarea en un procesador, y en fijar un orden de ejecución dentro de cada procesador sujeto a dos restricciones: (a) ningún procesador ejecuta más de una tarea a la vez y, (b) las tareas que preceden a otra han de haber terminado para empezar la ejecución de ésta (es decir, se preservan las relaciones de precedencia explícitas en el grafo).

En este modelo aparece el concepto de *camino crítico* (CP: Critical Path), entendido como el camino más largo que va desde un nodo inicial hasta un nodo final. La Figura 1.11 representa un grafo TPG de nueve tareas donde el camino crítico está formado por las tareas {T1,T7,T9}, y cuyo valor de CP es de 23 (i.e. la suma de los tiempos de cómputo y las comunicaciones de las tareas que forman el CP).

Para aquellas aplicaciones en las que las tareas se intercambian un volumen de datos pequeño, algunos autores consideran que el tiempo asociado a las comunicaciones es insignificante en relación al cómputo y puede ser conside-

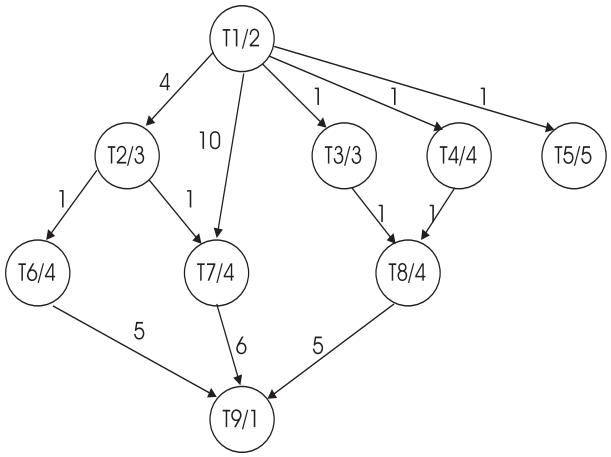

Figura 1.11: Ejemplo de grafo TPG.

rado como nulo [Her91] [KS84] [SWP90]. Esta aproximación da lugar a un grafo TPG sin volúmenes de comunicación. En este caso concreto de TPG, la suma de los tiempos de cómputo de las tareas que forman el CP representa el tiempo mínimo alcanzable en la ejecución del programa, independientemente de número de procesadores que se usen.

La mayoría de los algoritmos de scheduling para TPGs se basan en las políticas de lista. La idea básica consiste en asignar prioridades a las tareas del TPG e ir ordenándolas en una lista según un orden descendiente de prioridades. A partir de la lista generada, el algoritmo examina las tareas en un orden de prioridad de mayor a menor.

Los principales criterios que se usan para asignar prioridades a los nodos del grafo son las siguientes: (a) el *top-level*, que para cada nodo Ti es la longitud del camino más largo desde un nodo de entrada hasta Ti y, (b) el *bottom-level* que corresponde a la longitud del camino más largo desde un nodo Ti a un nodo de salida. Para grafos TPG con volúmenes de comunicación, estos atributos son dinámicos puesto que el coste de comunicación entre dos nodos adyacentes se considera 0 cuando estos se asignan al mismo procesador.

Algunos ejemplos representativos de algoritmos de scheduling que usan atributos de nivel en la asignación de prioridades a los nodos son los siguientes: ISH (Insertion Scheduling Heuristic) [KL87], DSC (Dominant Sequence Clustering) [YG94], DCP (Dynamic Critical Path) [KA96], LAST (Localized Allo-

cation of Static Tasks) [BP89], MCP (Modified Critical Path) [WG90], ETF (Earliest Task First) [HCAL89], EZ (Edge Zeroing) [Sar89b] y MD (Mobility Directed) [WG90].

En función del criterio seguido para asignar prioridades a las tareas, y también para seleccionar el procesador al cual se asigna cada tarea, puede establecerse una clasificación de estos algoritmos según los siguientes criterios.

- *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. Ejemplos de este caso son el MD, DSC, DCP y LAST.

- *Lista estática.* Cuando la lista de prioridades se construye antes de iniciar la asignación, y se mantiene la misma lista a lo largo de todo el proceso, se tratará de algoritmos de lista estática como son el ISH, ETF, LAST y EZ.

- *Lista dinámica.* En este caso la lista se va actualizando a medida que transcurre el proceso de scheduling. Algoritmos representativos de esta categoría son el DSC, DCP, 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 bastantes algoritmos de scheduling como el ETF, ISH, MCP, LAST, MD y EZ.

A parte de la técnica de asignación utilizada, los algoritmos de scheduling para TPGs pueden dividirse en dos grandes categorías, dependiendo de la disponibilidad que se considere del número de procesadores. En [KA99] se da una amplia clasificación y comparativa de los algoritmos según sean de los siguientes tipos: (a) BNP (Bounded Number of Processors), que consideran un número limitado de procesadores y basan su asignación en este número concreto y, (b) 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 en [KA99] se asume 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 1.1 se presenta, a modo de resumen, una clasificación de los algoritmos de scheduling más representativos, en función de sus características, agrupados en los tipos BNP y UNC.

Tabla 1.1: Clasificación de los algoritmos de scheduling para grafos TPG.

	Alg.	cam. crítico	lista estática	lista dinámica	greedy
BNP	ETF		x		x
	ISH		x		x
	MCP			x	x
	LAST	x	x		x
UNC	DSC	x		x	
	DCP	x		x	
	MD	x		x	x
	EZ		x		x

Cuando la arquitectura está formada por un número concreto de procesadores los algoritmos UNC pueden necesitar una segunda etapa, denominada *cluster-mapping*, que asigna los clusters de tareas a los procesadores del sistema. En este trabajo se estudia la efectividad de usar el modelo TTIG para modelar el grafo de clusters y realizar la fase de *cluster-mapping* basada en el mismo.

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

Los algoritmos de mapping que utilizan el grafo TIG, parten de un modelo de programa en el que no se incluye información acerca del comportamiento temporal de las tareas, tal como se ilustra en el grafo TIG con cuatro tareas representado en la Figura 1.12. Debido a ello, estos algoritmos basan su estrategia en la minimización de una determinada función de coste.

En general, las funciones de coste consideran los costes de cómputo y comunicación asociados a las tareas del grafo. Mediante la evaluación de estos

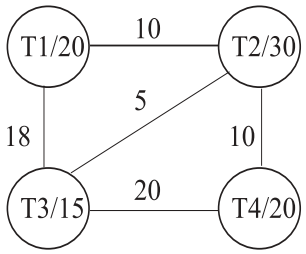

Figura 1.12: Ejemplo de grafo TIG.

parámetros, las estrategias de mapping desarrolladas para TIG suelen basarse en la consecución de los dos siguientes objetivos:

- Balanceo de la carga entre los procesadores del sistema.
- Minimizar el coste de comunicación entre procesadores.

Estos dos objetivos son en sí mismos contradictorios. Por un lado la distribución uniforme del cómputo contribuirá a que las comunicaciones sean mayores, mientras que por otro lado la minimización de las comunicaciones requerirá asignar muchas tareas al mismo procesador, desequilibrando de esta forma la carga. Podemos ver por ejemplo que si se asignan a dos procesadores las cuatro tareas del grafo TIG de la Figura 1.12, para repartir el cómputo de manera equilibrada habría que juntar las parejas (T2,T3) y (T1,T4) respectivamente. En cambio, para minimizar los costes de comunicación entre procesadores habría que agrupar T1, T3, y T4 en el mismo procesador, y dejar T2 sola en el otro.

Así pues, las funciones de coste con las que trabajan los algoritmos basados en TIG, calculan el coste de un determinado mapping m como $f(m) = W_m + U_m$ donde W_m y U_m son el coste global de cómputo y comunicación asociados a m respectivamente.

Mayoritariamente, las funciones de coste más utilizadas son la *minimax* y la *summed*. El objetivo de cada una de ellas es el siguiente:

- Función *minimax*. Estima el tiempo requerido por cada procesador (tiempo de cómputo + tiempo de comunicación) bajo un determinado mapping, y el mayor tiempo obtenido será el valor a minimizar.

- Función *summed*. Considera como asignación ideal la que tiene la carga computacional distribuida de manera uniforme entre los procesadores y no comporta un coste adicional debido a las comunicaciones. La bondad de un determinado mapping se mide en términos de desviación frente a la asignación ideal.

En general, las heurísticas desarrolladas para el modelo TIG basadas en ambas funciones de coste, pueden dividirse en los tres grupos siguientes:

- Métodos greedy. En el ámbito de las estrategias basadas en el modelo TIG se engloba dentro de esta categoría a todas aquellas estrategias que permiten encontrar una asignación de forma rápida mediante un mecanismo paso a paso que no reasigna una tarea ya colocada. Es decir, son estrategias sin vuelta atrás. Ejemplos de estrategias *greedy* las encontramos en [Lo88] y [AER93].

- Métodos iterativos. Se parte de una asignación inicial (no necesariamente buena), a partir de la cual se va a intentar mejorarla iterativamente. En la mayoría de los casos el proceso se basa en el intercambio de pares de tareas entre dos procesadores o bien en el movimiento individual de tareas. A cada nueva asignación obtenida se evalúa su bondad y se repite el proceso de forma reiterada. La asignación se acepta como definitiva cuando se alcanza un mínimo local en la función de coste que evalúa la calidad del mapping. En [LA87], [SM93], [BdKT95] y [HC97] se proponen distintas heurísticas basadas en métodos iterativos. La naturaleza de estas heurísticas hace que se obtengan soluciones de mejor calidad que con los métodos greedy, a costa de aumentar su complejidad temporal.

- Métodos mixtos. Este grupo de heurísticas incluye un conjunto de estrategias que buscan un compromiso entre la eficiencia computacional de los métodos greedy y los puramente iterativos. Para conseguir este objetivo, su diseño está condicionado a la existencia de dos fases: una primera obedece a la categoría tipo greedy y la segunda se amolda al tipo de estrategia iterativa. Propuestas importantes dentro de este grupo son los algoritmos expuestos en [SER90], [ERS90] y [SRCL98].

1.3.3 Algoritmos de mapping basados en el modelo TTIG

En este trabajo se proponen dos algoritmos de mapping de distinta complejidad basados en el nuevo modelo TTIG, que se estudiarán en el Capítulo 4. La estrategia básica de ambas heurísticas consiste en determinar para cada par de tareas adyacentes la diferencia de tiempo que comporta su ejecución en el mismo procesador o en procesadores distintos. 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, que da una indicación de su posibilidad de concurrencia.

Mediante la evaluación de estos tiempos se lleva a cabo una política en la que se asignan al mismo procesador aquellas tareas más dependientes, mientras que se asignan a procesadores distintos las tareas capaces de realizar una ejecución concurrente, intentando potenciar de esta forma la ejecución concurrente de las tareas.

1.3.4 Herramientas de mapping

A lo largo de los últimos años se han desarrollado herramientas de mapping que proporcionan entornos amigables para realizar la asignación de tareas a procesadores de forma automática. Dichas herramientas se basan principalmente en la simulación de la ejecución de un grafo de tareas para conocer la influencia de los diferentes algoritmos de mapping. De los dos modelos clásicos utilizados para el mapping, TPG y TIG, únicamente se han desarrollado hasta ahora, herramientas de mapping basadas en el modelo TPG. Esto es debido a que, de los dos modelos, el TPG es el único que permite simular la ejecución del programa una vez realizado el mapping ya que lleva asociada la información relativa al orden de ejecución de las tareas. Algunas de las herramientas de mapping más relevantes de este tipo, con sus características son las siguientes:

- PYRROS [YG92]. Genera código paralelo y el correspondiente scheduling en tiempo de compilación. La entrada a la herramienta la constituyen el grafo de precedencia de tareas y el código secuencial en lenguaje C. PYRROS genera como salida el código C paralelo, y el scheduling aso-

ciado. Proporciona además facilidades para la visualización de los grafos de tareas y los resultados de la ejecución mediante trazas de Gantt.

- Parallax [LER93]. Como en el caso anterior, esta herramienta se basa en programas modelados como grafos de precedencia de tareas. El usuario es quién da la especificación del grafo y la estimación de los pesos asociados al cómputo y la comunicación. Proporciona varias estrategias de asignación y diversos tipos de arquitecturas entre los cuales el usuario puede escoger. Los resultados de las ejecuciones se muestran mediante diversos diagramas que reflejan el uso de los procesadores, la ocurrencia de comunicaciones, etc.

- CASCH [AKWS00]. Constituye un entorno completo de programación paralela que incluye las etapas de paralelización, particionamiento, scheduling, comunicación, sincronización, generación de código y evaluación de rendimiento. El proceso de paralelización se lleva a cabo mediante un compilador que convierte automáticamente el código secuencial en paralelo con una estructura de grafo TPG. La herramienta CASCH se ejecuta en una SUN workstation y genera código paralelo para los sistemas Intel Paragon y IBM SP2. Los pesos asociados a los nodos y los arcos del grafo de tareas se computan usando una base de datos que contiene el tiempo que requieren varias operaciones de cómputo, comunicación y entrada/salida, para dichos sistemas.

  CASCH permite ejecutar diversos algoritmos de scheduling de TPGs, que pueden ser generados aleatoriamente, interactivamente o bien directamente de los programas reales. Esto facilita la evaluación de las diversas estrategias de asignación que incluye la herramienta.

- OREGAMI [LRG+91]. Este entorno lo constituyen un conjunto de herramientas, que incluyen un compilador de descripciones de grafos que permiten definir las aplicaciones paralelas en un modelo denominado grafo de comunicaciones temporal. Junto con el compilador existe una herramienta de mapping que incorpora un conjunto de bibliotecas, para la asignación de forma óptima de determinadas familias de grafos sobre

ciertas arquitecturas. Se incorporan además estrategias más generales que permiten la asignación de los grafos para los que no se dispone de una estrategia óptima. El entorno se completa con una herramienta de visualización y análisis de rendimiento de las asignaciones obtenidas.

Vemos pues que para el modelo TPG existe diversidad de herramientas para la realización y análisis del mapping, pero en los entornos distribuidos actuales de paso de mensajes, el TPG no se ajusta a la definición de tareas que normalmente realiza el programador; por lo tanto, las herramientas descritas no resultan apropiadas para entornos tipo cluster y aplicaciones paralelas de propósito general. Para estos entornos, en nuestro grupo se han desarrollado herramientas específicas de mapping y simulación que parten de la definición del comportamiento temporal del programa, respetando la definición de tareas del programador, y permiten realizar una evaluación detallada de las distintas alternativas de mapping, tanto en plataforma real como mediante simulación, tal como se verá en los capítulos de experimentación.

1.4 Objetivos del trabajo

Los entornos de paso de mensajes para los sistemas distribuidos actuales permiten la definición de programas paralelos con una estructura de tareas arbitraria. Las tareas que forman parte de una aplicación pueden exhibir distintos tipos de paralelismo: (a) paralelismo de tareas donde las tareas realizan distintas funciones, (b) paralelismo de datos, en el que se explota la ejecución concurrente de la misma tarea para distinto conjunto de datos y, (c) paralelismo mixto, que combina las anteriores y explota tanto la ejecución concurrente de distintas tareas como la de la misma tarea para distintos datos.

Para explotar la ejecución concurrente de las tareas de cada aplicación será necesario realizar una asignación que garantice que las tareas independientes, o con posibilidad de ejecutar concurrentemente una parte importante de su cómputo se asignen a distintos procesadores, mientras que las tareas con fuertes dependencias se ejecutan en el mismo procesador.

Las librerías actuales de paso de mensajes PVM y MPI proporcionan mecanismos de mapping automático basados en simples heurísticas que no

contemplan la relación de dependencia entre las tareas de la aplicación. Para el desarrollo de políticas de mapping más eficientes es preciso partir de un modelo que abstraiga las características de comportamiento de las tareas que forman la aplicación.

Las heurísticas de mapping desarrolladas en la literatura contemplan como modelo de la aplicación los grafos clásicos TPG (Task Precedence Graph) y TIG (Task Interaction Graph). Ambos modelos asumen un comportamiento temporal concreto para las tareas del grafo. Así pues, el grafo TPG es indicado para aplicaciones cuyas tareas se comunican por el principio y el final de las mismas y se establece un orden parcial en la ejecución. En el grafo TIG se asume que todas las tareas son completamente concurrentes.

En este trabajo se define un nuevo modelo denominado TTIG (Temporal Task Interaction Graph), que captura el comportamiento de las tareas con un patrón de comunicaciones arbitrario, ya sea con paralelismo de tareas o de datos. Usando como modelo el grafo TTIG se proponen los siguientes objetivos a desarrollar:

- Definir formalmente el grafo TTIG y los parámetros que incluye de tiempo de cómputo, volumen de comunicación y grado de paralelismo.
- Proponer el mecanismo automático de obtención del grafo TTIG a partir de los programas con paso de mensajes.
- Establecer un método que permita calcular un valor de cota sobre el número de procesadores necesario en la asignación de tareas.
- Definir políticas de mapping estático adecuadas al nuevo modelo, que permitan explotar la ejecución concurrente de las tareas.
- Analizar la influencia de usar el nuevo parámetro del grado de paralelismo en la sensibilidad de las políticas de mapping frente a posibles variaciones en la estimación de los parámetros de entrada.
- Evaluar la bondad de las políticas basadas en el modelo TTIG comparando con la asignación óptima, para un conjunto de grafos de dimensiones reducidas.

- Comparar la efectividad de las políticas basadas en el nuevo modelo TTIG frente a las políticas basadas en los modelos clásicos TPG y TIG.
- Mostrar la escalabilidad de las políticas basadas en el modelo TTIG.
- Mostrar, como resultado adicional, la utilidad de usar el modelo TTIG en la segunda fase de mapping de los algoritmos de scheduling que se desarrollan en dos etapas para grafos TPG .