Capítulo 4

Algoritmos de mapping basados en el modelo TTIG

En los capítulos precedentes se ha estudiado el nuevo modelo TTIG, propuesto para la resolución del problema del mapping, y se han analizado las ventajas que aporta respecto a los modelos clásicos, para la toma de decisiones eficientes.

En los algoritmos de mapping desarrollados para los modelos clásicos TPG y TIG, las estrategias de asignación difieren en función de la información que aporta el modelo en sí mismo. Para los grafos TPG, las políticas de mapping contemplan, en su mecanismo de decisión, los tiempos de cómputo de las tareas, los volúmenes de comunicación y el orden de precedencias en la ejecución. Para grafos basados en el modelo TIG, las políticas de mapping determinan la asignación mediante una función de coste previamente definida, que intenta un equilibrio entre la distribución uniforme del cómputo y la minimización de las comunicaciones.

La principal diferencia existente entre las políticas propuestas para los dos modelos clásicos, estriba en el hecho de poder o no contemplar la información temporal referente al orden de ejecución de las tareas de la aplicación. El modelo TTIG propuesto en el presente trabajo, representa una alternativa que unifica los dos modelos clásicos. El grado de paralelismo incluido en el TTIG da información respecto al máximo grado de concurrencia de las tareas adyacentes. La información proporcionada por dicho parámetro facilita la definición de políticas de mapping eficientes para aplicaciones cuyas tareas ten-

gan un comportamiento temporal arbitrario. Estas políticas de mapping serán también efectivas cuando las tareas de la aplicación tengan un comportamiento como el previsto en los modelos clásicos TPG y TIG.

En este capítulo se presentan dos algoritmos de mapping basados en el modelo TTIG. Dichos algoritmos se fundamentan en la evaluación de unos mismos parámetros del grafo pero se diferencian básicamente en el mecanismo de elección de las tareas a asignar, proporcionando dos mecanismos de asignación tal como se expone a continuación.

- TASC (Task ASsignment exploiting Concurrency). Realiza la asignación de tareas tratando de explotar la posibilidad de ejecución concurrente de las tareas adyacentes.

- MATE (Mapping Algorithm based on Task dEpendencies). Prioriza la asignación al mismo procesador de las tareas adyacentes más dependientes (es decir, con menos posibilidad de concurrencia).

Ambas heurísticas se definen considerando un número de procesadores dado. Respecto a la arquitectura se asume que es un sistema homogéneo, del cual no se considera la topología de interconexión de los nodos. Bajo estas premisas, para la asignación de tareas se contempla sólo la aplicación, y las diferencias en las asignaciones se analizarán en función de cuáles son las tareas asignadas a un mismo procesador, siendo irrelevante el procesador concreto al que han sido asignadas.

Se estudia además la complejidad computacional asociada a los dos algoritmos y se analiza el problema de la robustez, comprobando los cambios que se dan en las asignaciones al variar los parámetros de entrada del grafo en un determinado porcentaje.

El desarrollo de las políticas de mapping se lleva a cabo a lo largo del capítulo, apoyado en el ejemplo de grafo TTIG de la Figura 4.1. Dicho grafo está formado por siete tareas {T1,..,T7}, donde T1 es la tarea inicial. Se entiende como tarea inicial aquella que puede empezar en el momento de iniciar la ejecución del programa. Así pues, serán tareas iniciales aquellas que no tienen una primitiva de recepción como primera sentencia.

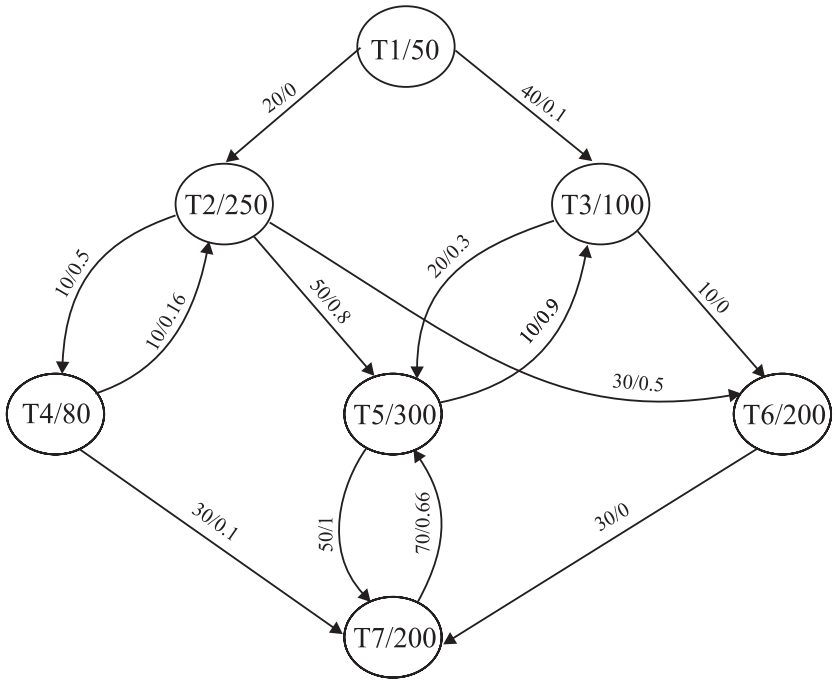

Figura 4.1: Ejemplo de grafo TTIG.

A continuación se expone el mecanismo de asignación de las dos políticas TASC y MATE desarrolladas para el modelo TTIG.

4.1 Fundamentos de la estrategia de asignación

En esta sección se describen los parámetros de comportamiento que se extraen del modelo TTIG, en los que se fundamentan las dos heurísticas de asignación TASC y MATE. En primer lugar, para cada nodo del grafo TTIG se define un valor de *nivel* que será el que se utilice para realizar la asignación de las tareas del grafo con un determinado orden de prioridad. En segundo lugar, para cada par de tareas adyacentes se calculan los valores de *mínimo tiempo de ejecución junto y separado*. Esto permite realizar una valoración de la idoneidad de asignar las tareas adyacentes al mismo procesador o bien a procesadores distintos, en función del volumen de comunicación que se transmiten y de su capacidad de ejecución paralela.

En los siguientes apartados se describe el proceso a seguir para calcular

cada uno de dichos parámetros.

4.1.1 Valor de nivel de los nodos

Dado un grafo TTIG definimos el *nivel* de un nodo T 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.

Cabe señalar que el concepto de nivel ha sido previamente definido y usado en la literatura de forma distinta para grafos TPG. En aquel caso, el nivel de un nodo se definía como la longitud del camino más largo desde un nodo inicial hasta él, siendo indicativo del mínimo tiempo de inicio de la tarea correspondiente.

Para un grafo TTIG, se define el nivel de cada tarea del grafo, $LN(T)$, como el mínimo número de arcos que hay que atravesar desde un nodo inicial T_{in} hasta T, y se calcula como:

$$LN(T) = min_{T_{in} \in S} d(T_{in}, T) \tag{4.1}$$

donde :

S es el conjunto formado por los nodos que representan las tareas iniciales del grafo TTIG.

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

El cálculo del valor de nivel de cada nodo T puede hacerse mediante el algoritmo que se muestra en la Figura 4.2, en el que se establecen los siguientes pasos:

(a) Construir el conjunto de tareas de nivel 0, LN_0, formado por las tareas iniciales.

(b) Construir el conjunto de tareas de nivel i, LN_i, con cada tarea T para la cual existe un arco desde una tarea de nivel $i-1$ hacia T.

Dado que T1 es la única tarea inicial en el grafo TTIG de la Figura 4.1,

se tiene $LN_0 = \{T1\}$. A partir de aquí se desarrollarán tres iteraciones del algoritmo para establecer las tareas de cada nivel que serán las siguientes:

$LN_1 = \{T2, T3\}$

$LN_2 = \{T4, T5, T6\}$

$LN_3 = \{T7\}$

```
N = conjunto de nodos del TTIG
LN_i = ∅ para todo i ∈ {0, .., m − 1}

/* conjunto de tareas de nivel 0 */
LN_0 = cj. de tareas iniciales
N = N - {LN_0}

/* tareas de nivel i */
i = 1;
mientras N ≠ ∅
   para cada T_u ∈ LN_{i−1}
      para cada T ∈ N con arco T_u → T
         LN_i = LN_i ⋃ {T}
         N = N - {T}
      fin_para
   fin_para
   i = i+1;
fin_mientras
```

Figura 4.2: Algoritmo para el cálculo de nivel de los nodos del TTIG.

El algoritmo de cálculo de nivel deberá llevarse a cabo durante tantas iteraciones como tareas tiene el grafo, por lo tanto la complejidad computacional asociada al algoritmo será $O(n)$, donde n es el número de nodos del grafo.

4.1.2 Evaluación de los tiempos de ejecución de las tareas adyacentes según sus dependencias

Para cada par de tareas adyacentes, Ti y Tj, de un grafo TTIG, se tiene la siguiente información relativa a su comportamiento:

- Tiempo de ejecución estimado de cada una de ellas.
- Volumen de comunicación entre ambas.
- Grado de paralelismo.

Teniendo en cuenta los tres parámetros de comportamiento reflejados en el TTIG, y considerando cada par de tareas adyacentes Ti y Tj de forma aislada, se puede calcular el mínimo tiempo necesario para ejecutar ambas tareas en el mismo procesador (tiempo de ejecución junto), o bien separadas en distintos procesadores (tiempo de ejecución separado), mediante el mecanismo que se indica a continuación.

a. *Tiempo de ejecución junto.*

   Corresponde al mínimo tiempo necesario para ejecutar las dos tareas adyacentes Ti y Tj en un mismo procesador, y se calcula como la suma de sus tiempos de cómputo tal como se indica en la expresión 4.2. El coste de comunicación se considera despreciable en este caso.

$$t\_junto(Ti, Tj) = w(Ti) + w(Tj) \tag{4.2}$$

   El valor de tiempo de ejecución junto para las tareas T2 y T4 del grafo TTIG de la Figura 4.1 será $t\_junto(T2, T4) = w(T2) + w(T4) = 250 + 80 = 330$.

b. *Tiempo de ejecución separado.*

   Corresponde al mínimo tiempo necesario para ejecutar las dos tareas adyacentes en procesadores distintos. Admitiendo una situación ideal, el

tiempo invertido en comunicación se traduce directamente en el volumen de datos transmitido. De esta forma, el tiempo de ejecución separado se calcula como la suma de los tiempos de cómputo de ambas tareas más los costes de comunicación entre ambas. A este valor de cómputo más comunicación se le resta el máximo tiempo que las dos tareas pueden paralelizar su ejecución, que viene determinado por el grado de paralelismo, tal como se indica en la siguiente expresión.

$$t\_sep(Ti, Tj) = w(Ti) + w(Tj) + c(Ti, Tj) + c(Tj, Ti) - [w(Tj) \cdot p(Ti, Tj)] \quad (4.3)$$

Aplicando la fórmula 4.3, el mínimo tiempo de ejecución separada de las tareas T2 y T4 del grafo ejemplo se calculará como:

$$\begin{aligned} t\_sep(T2, T4) &= w(T2) + w(T4) + c(T2, T4) + c(T4, T2) - [w(T4) \times p(T2, T4)] \\ &= 250 + 80 + 10 + 10 - (80 \times 0.5) = 310 \end{aligned}$$

Es interesante remarcar que en el caso de tareas con dependencias bidireccionales, el cálculo de t_sep solo es necesario hacerlo con el grado de paralelismo indicado en uno de los dos sentidos, puesto que será equivalente al cálculo realizado en el sentido inverso. Así vemos que $t\_sep(T2, T4)$ da el mismo valor de tiempo que $t\_sep(T4, T2)$ como se muestra a continuación.

$$\begin{aligned} t\_sep(T4, T2) &= w(T4) + w(T2) + c(T4, T2) + c(T2, T4) - [w(T2) \times p(T4, T2)] \\ &= 80 + 250 + 10 + 10 - (250 \times 0.16) = 310 \end{aligned}$$

Los valores calculados de $t\_junto(T2, T4)$ y $t\_sep(T4, T2)$ indican que, considerando las dos tareas aisladas del resto del grafo con sus dependencias mutuas, se dará una estimación favorable a ejecutarlas separadas en distintos procesadores, con una diferencia de 20 unidades de tiempo favorable a esta opción respecto al tiempo de ejecución junto.

En la Tabla 4.1 se indican los valores de tiempo junto y tiempo separado que corresponden a cada par de tareas adyacentes del grafo TTIG, junto con la valoración que en principio corresponde a cada par de tareas según estos dos valores de tiempo.

Tabla 4.1: Valoración inicial de asignación de las tareas adyacentes.

(Ti,Tj)	$t\_junto(Ti,Tj)$	$t\_sep(Ti,Tj)$	**valoración**
(T1,T2)	300	320	juntar
(T1,T3)	150	180	juntar
(T2,T4)	330	310	separar
(T2,T5)	550	360	separar
(T2,T6)	450	380	separar
(T3,T5)	400	340	separar
(T3,T6)	300	310	juntar
(T4,T7)	280	290	juntar
(T5,T7)	500	420	separar
(T6,T7)	400	430	juntar

4.2 Estrategia de asignación basada en explotar la concurrencia de las tareas adyacentes (TASC)

A partir de la información temporal que se captura en el modelo TTIG se ha definido una primera estrategia de asignación de tareas denominada TASC (Task ASsignment exploiting Concurrency), cuyo objetivo es el de maximizar la ejecución concurrente de las tareas adyacentes. La idea subyacente al definir la heurística TASC ha sido la de implementar un algoritmo de mapping que permite llegar a asignaciones eficientes con una baja complejidad [Gui01a]. La estrategia TASC se compone de las dos fases siguientes:

Fase 1. Definición de la lista de pares de tareas adyacentes según un orden de prioridad.

Fase 2. Asignación de las tareas a procesadores a partir del orden de lista.

A continuación se describe el proceso para llevar a cabo cada fase.

4.2.1 Fase 1. Definición de la lista de prioridades

Para cada par de tareas adyacentes (Ti, Tj) del grafo TTIG, se definen los siguientes valores, que se utilizarán para crear la lista de prioridad.

(a) Nivel del par.

El nivel del par de tareas (Ti,Tj), que denotamos como $nivel\_par(Ti, Tj)$, corresponde al mínimo valor de nivel de ambas tareas.

$$nivel\_par(Ti, Tj) = min\{LN(Ti), LN(Tj)\} \tag{4.4}$$

Tenemos pues por ejemplo, para el grafo TTIG de la Figura 4.1, que $nivel\_par(T2, T4) = min\{1, 2\} = 1$.

(b) Tiempo máximo.

El valor de tiempo máximo para dos tareas adyacentes, denominado como $t\_max(Ti, Tj)$, corresponde al máximo valor entre tiempo junto y tiempo separado calculado para ambas tareas, según se indica en la siguiente expresión:

$$t\_max(Ti, Tj) = max\{t\_junto(Ti, Tj), t\_sep(Ti, Tj)\} \tag{4.5}$$

Según los valores calculados en la Tabla 4.1, para el par de tareas (T2,T4) se obtiene $t\_max(T2, T4) = max(330, 310) = 330$.

Con los valores de nivel del par y tiempo máximo, se crea la lista de pares de tareas adyacentes (Ti, Tj), ordenada de mayor a menor prioridad, siguiendo en primer lugar un orden creciente de $nivel\_par(Ti, Tj)$, y dentro de cada nivel se ordena de mayor a menor según el valor de $t\_max(Ti, Tj)$. Siguiendo este mecanismo se obtiene la lista ordenada de tareas adyacentes que muestra la Tabla 4.2.

El objetivo que se persigue mediante la definición de la lista de prioridades, es por un lado hacer un recorrido del grafo de arriba a abajo según el orden en que pueden dar comienzo las tareas. Por otro lado, dentro de cada nivel, se

Tabla 4.2: Lista ordenada de los pares de tareas adyacentes.

prioridad	**(Ti,Tj)**	**nivel_par**	$t\_max(Ti,Tj)$
1 (mayor)	(T1,T2)	0	320
2	(T1,T3)	0	180
3	(T2,T5)	1	550
4	(T2,T6)	1	450
5	(T3,T5)	1	400
6	(T2,T4)	1	330
7	(T3,T6)	1	310
8	(T5,T7)	2	500
9	(T6,T7)	2	430
10 (menor)	(T4,T7)	2	290

da mayor prioridad a la valoración obtenida para las tareas que tienen mayor valor de $t\_max(Ti,Tj)$, puesto que una asignación apropiada de las mismas va a contribuir en mayor medida a la reducción del tiempo de ejecución global.

En las distintas alternativas de ordenación probadas durante la etapa preliminar de definición del algoritmo TASC, se evaluó también la opción de crear la lista en orden creciente de valor de diferencia obtenido como $|t\_junto(Ti,Tj) - t\_sep(Ti,Tj)|$, dando mejores resultados la opción escogida de ordenar por el valor de $t\_max(Ti,Tj)$. Esto puede ser explicado porque los pares que tienen mayor valor de diferencia, pueden muy bien corresponder a los que tienen menor valor de cómputo y comunicación y en consecuencia tienen menor incidencia en el tiempo de ejecución final.

4.2.2 Fase 2: Asignación de tareas a procesadores

Una vez generada la lista de tareas adyacentes por orden de prioridad, la segunda fase de la estrategia TASC consiste en asignar cada tarea del grafo a uno de los procesadores recorriendo las tareas adyacentes por orden de lista, para determinar la decisión de juntarlas al mismo procesador o bien separarlas. Dicha decisión se toma en base a los siguientes criterios:

(a) Cuando la valoración que corresponde al par (Ti,Tj) en la lista es la de separar, se decide separarlas para explotar su ejecución concurrente.

(b) Si la valoración es la de juntar, la decisión favorable sería en principio la de juntar, pero antes de llevarla a cabo se aplica el siguiente refinamiento:

- Si Ti o Tj se habían juntado previamente con otra adyacente, se decide separarlas, ya que de lo contrario para grafos TTIG con un grado de paralelismo muy bajo, y por lo tanto con un comportamiento similar al del modelo TPG, se asignarían todas las tareas al mismo procesador.

  Esta situación es la que se da en el subgrafo formado por las tareas T1, T2 y T3 del grafo TTIG de la Figura 4.1. La valoración que se da en la lista para los pares de tareas adyacentes (T1,T2) y (T1,T3) es en ambos casos de juntar, pero si se asignan las tres tareas al mismo procesador se va a evitar de entrada la posibilidad de ejecución concurrente de las tareas T2 y T3. El refinamiento que aplica el algoritmo en este caso será separar T3 de T1 si anteriormente T1 ya se había juntado con T2.

- En el caso que ni Ti ni Tj se hayan asignado previamente, se decide juntarlas a un único procesador.

A partir de las tareas Ti y Tj y la decisión tomada se selecciona el procesador más adecuado para almacenar cada tarea. Para ello se realiza en primer lugar la comprobación de si una de las dos tareas ha sido previamente asignada y se almacena cada tarea a su correspondiente procesador de la siguiente forma:

- Si una de las dos tareas, Ti o Tj, está ya asignada y la decisión es juntar, se asignará la tarea pendiente al mismo procesador que su adyacente. Si la decisión es separar, se asigna la tarea pendiente al procesador con mínimo cómputo acumulado.

- Si ninguna de las dos tareas Ti y Tj ha sido previamente asignada y la decisión es juntar, se ubican ambas juntas al procesador con mínimo cómputo acumulado. Si la decisión es separar, se asigna cada una de ellas al procesador con menos cómputo acumulado.

La política TASC se implementa mediante un algoritmo principal cuyo pseudocódigo se muestra en la Figura 4.3, que recorre las tareas de la lista y pasa la decisión de juntarlas o separarlas al procedimiento *ubicar*. El procedimiento *ubicar* (Figura 4.4), toma como parámetros de entrada las tareas Ti y Tj del par de adyacentes junto con la decisión que les corresponde, y a partir de estos valores realiza la asignación de cada tarea a un procesador concreto.

```
N = conjunto de nodos del TTIG
P = conjunto de procesadores
Pi=0 para todo Pi ∈ P
n=1
mientras N≠ ∅
   (Ti,Tj)=par con número de orden en la lista igual a n
   si (Ti ∈ N or Tj ∈ N)
      si (valoración = separar)
          ubicar(Ti,Tj,separar)
      si no
         si (Ti or Tj juntada anteriormente) ubicar(Ti,Tj,separar)
         si_no ubicar(Ti,Tj,juntar)
      fin_si
   fin_si
   n=n+1
fin_mientras
```

Figura 4.3: Pseudocódigo del algoritmo de la estrategia TASC.

En la Tabla 4.3 se muestran los sucesivos pasos de asignación que se obtienen al aplicar la heurística TASC al grafo TTIG de la Figura 4.1 para dos procesadores {P0,P1}. Los datos que se muestran en la tabla, para cada par de tareas adyacentes, referentes al desarrollo del algoritmo de mapping, son los siguientes:

- paso: Número de paso del algoritmo.

```
ubicar(Ti,Tj,decisión)

si Ti or Tj está asignada

   /* una de las dos tareas está ya asignada */
   T_ass=tarea asignada
   T_no_ass=tarea no asignada
   si (decisión=juntar) P(T_no_ass)=P(T_ass)
   si no P(T_no_ass)=procesador con mínimo cómputo
   N=N-{T_no_ass}
   actualizar cómputo P(T_no_ass)

si no

   /* ni Ti ni Tj han sido aun asignadas */
   si (decisión=juntar)
      P(Ti)=procesador con mínimo cómputo
      P(Tj)=P(Ti)
   si no
      P(Ti)=procesador con mínimo cómputo
      P(Tj)=procesador con mínimo cómputo
   fin_si

   N=N-{Ti,Tj}
   actualizar cómputo de P(Ti) y P(Tj)

fin_si
```

Figura 4.4: Procedimiento ubicar.

- (Ti,Tj): Par de tareas adyacentes (Ti,Tj) evaluados por orden de lista.
- P(T): Procesador al que está asignado la tarea T del par (Ti,Tj). Se indica con un guión cuando la tarea T no ha sido aún asignada.
- j_prev: Se indica con un * cuando la decisión de juntar ha sido ya aplicada en algún paso anterior para alguna de las dos tareas Ti o Tj.
- decisión: Indica la decisión de juntar o separar que el algoritmo principal le pasa al procedimiento *ubicar*.
- asignación al procesador concreto que se realiza de las tareas pendientes

de asignar dentro del par (Ti,Tj).

- Pi: Valor de cómputo total acumulado asignado al procesador Pi.

Tabla 4.3: Desarrollo del algoritmo TASC para dos procesadores.

paso	**(Ti,Tj)**	**P(Ti)**	**P(Tj)**	**j_prev**	**decisión**	**asignación**	**P0**	**P1**
1	(T1,T2)	-	-		juntar	P0:T1,T2	300	0
2	(T1,T3)	P0	-	*	separar	P1:T3	300	100
3	(T2,T5)	P0	-		separar	P1:T5	300	400
4	(T2,T6)	P0	-		separar	P0:T6	500	400
5	(T3,T5)	P1	P1				500	400
6	(T2,T4)	P0	-		separar	P1:T4	500	480
7	(T3,T6)	P1	P0				500	480
8	(T5,T7)	P1	-		separar	P1:T7	500	680

El algoritmo se desarrolla evaluando los pares de tareas adyacentes según el orden marcado por la lista de prioridad, hasta que han sido todas asignadas, momento en el cual detendrá su ejecución. Siguiendo los pasos del algoritmo mostrados en la tabla, vemos que para el primer par de tareas evaluado, (T1,T2), corresponde la decisión de juntar y se asignan las dos al procesador P0, puesto que ninguna de ellas tenia procesador asignado previamente. Para el siguiente par de adyacentes de la lista, (T1,T3), correspondería también la decisión de juntar ya que esta era la valoración obtenida en la lista, pero T1 se había ya juntado con una adyacente, por lo tanto la decisión final es separar y se asigna T3 al procesador con menos cómputo acumulado que en este momento es P1. Siguiendo los pasos sucesivos se obtiene la asignación final, que para el ejemplo que nos ocupa será P0:(T1,T2,T6) P1:(T3,T4,T5,T7).

En la Tabla 4.4 se muestran las distintas asignaciones que se obtienen para el mismo grafo TTIG con el cómputo acumulado en cada procesador, al variar el número de procesadores de dos hasta cinco. Como se observa, exceptuando las tareas T1 y T2 que siempre se asignan al mismo procesador, el resto de tareas se distribuyen entre los procesadores con el fin explotar la concurrencia equilibrando al mismo tiempo el cómputo, tal como se muestra en la columna de cómputo acumulado.

Tabla 4.4: Asignaciones obtenidas para el algoritmo TASC al variar el número de procesadores.

proc.	Asignación	Cómputo acumulado
2	P0:(T1,T2,T6) P1:(T3,T4,T5,T7)	P0:500 P1:680
3	P0:(T1,T2,T7) P1:(T3,T4,T6) P2:(T5)	P0:500 P1:380 P2:300
4	P0:(T1,T2) P1:(T3,T4,T7) P2:(T5) P3:(T6)	P0:300 P1:380 P2:300 P3:200
5	P0:(T1,T2) P1:(T3) P2:(T5) P3:(T6) P4:(T4,T7)	P0:300 P1:100 P2:300 P3:200 P4:280

4.2.3 Análisis de la complejidad

El algoritmo que implementa la estrategia TASC consta de dos fases claramente diferenciadas: creación de la lista de prioridad y asignación de tareas a procesadores.

Considerando un grafo TTIG de n nodos, habrá como mínimo $(n-1)$ pares de tareas adyacentes si tiene una conectividad mínima. El máximo número de pares de adyacentes a evaluar será $\binom{n}{2} = \frac{n^2-n}{2}$, que corresponderá al caso de máxima conectividad. Estas dos situaciones de conectividad dan lugar a los casos de mínima y máxima complejidad para la fase de creación de la lista de prioridad, tal como se muestra a continuación.

El cálculo de los valores de $t\_junto(Ti, Tj)$ y $t\_sep(Ti, Tj)$ se realiza para cada par de tareas adyacentes, por lo tanto el número de pasos será del orden de $O(n)$ en el caso de mínima complejidad y $O(n^2)$ en el caso de máxima complejidad.

Para la creación de la lista de prioridad, se realiza una ordenación dentro de cada nivel según el valor de $t\_max(Ti, Tj)$. Por lo tanto, la complejidad asociada dependerá del número de niveles y será:

$$num.pasos = num.niveles \times complejidad\ alg.ordenación \tag{4.6}$$

donde el algoritmo de ordenación para una lista de m elementos tiene asociada una complejidad de $m \cdot \log m$.

En el caso de máximo número de niveles, un grafo tendrá n niveles con una

única tarea por nivel. El algoritmo de ordenación dentro de cada nivel se realiza en un único paso, y la complejidad asociada a este paso puede considerarse despreciable. En el caso de mínimo número de niveles, el grafo estará formado por un único nivel de n tareas, lo que según la expresión 4.6 conduce a un valor de número de pasos de $1 \cdot n \log n$. Así pues, la máxima complejidad asociada a la creación de la lista de prioridad será de $O(n \log n)$.

La fase de asignación, el algoritmo evalúa los pares de tareas (Ti,Tj) de la lista mientras quedan tareas por asignar. Esto hace que el mínimo número de iteraciones se dé cuando en cada una se asignan ambas tareas Ti y Tj, a lo que corresponde una complejidad de $O(n/2)$. El máximo número de iteraciones se dará cuando haya que evaluar todos los pares de adyacentes de la lista hasta finalizar, lo que comporta una complejidad máxima de $O(n^2)$.

Resumiendo el análisis realizado, y considerando las dos fases, la estrategia TASC tiene asociada una complejidad que oscila entre $O(n)$ y $O(n^2)$ para las situaciones más y menos favorables respectivamente.

En general, la estructura de tareas que tienen en su mayoría las aplicaciones no corresponde a ninguno de los dos casos extremos, y habitualmente una tarea se comunica con un número reducido de tareas adyacentes. Por lo tanto, se puede asumir sin pérdida de generalidad, que la complejidad media del algoritmo es del orden de $O(k \times n)$, donde k es un valor positivo.

4.3 Estrategia de asignación priorizando las adyacencias más dependientes (MATE)

Mediante el mecanismo de la estrategia TASC desarrollada en la sección anterior, se lleva a cabo una asignación de tareas de tal forma que se prima tanto la separación de las tareas adyacentes con posibilidad de concurrencia, como la asignación al mismo procesador de las tareas cuya ejecución junta resulta más eficiente. Mediante dicha estrategia se ha conseguido definir un algoritmo de mapping con una baja complejidad, que proporciona asignaciones bastante eficientes tal como se mostrará en los posteriores capítulos de experimentación. A pesar de todo, en el análisis de las asignaciones que proporciona la estrategia

TASC, pueden observarse los siguientes hechos.

- A cada iteración del algoritmo se evalúa un par de tareas adyacentes y se asignan las que están pendientes de asignación. Esto provoca que determinados pares no sean ya evaluados porque sus tareas han sido ya asignadas en pasos precedentes. Este es el caso del par de tareas (T3,T6) que tienen entre sí una valoración de juntar que no llega a ser tratada, porque dichas tareas se han asignado con anterioridad al evaluar los pares que ocupan las posiciones 2 y 4 respectivamente en la lista de prioridad.

- Con el objetivo de explotar la concurrencia, las cadenas de tareas adyacentes con una valoración de juntar pierden importancia, cuando en realidad puede ser un buen criterio asignarlas juntas al mismo procesador. Este es el caso de la cadena T1 $\rightarrow$ T3 $\rightarrow$ T6 $\rightarrow$ T7 del grafo estudiado, que debería ser asignada al mismo procesador ya que los grados de paralelismo son muy bajos.

Con la idea de evitar estas situaciones, se ha desarrollado una segunda estrategia de mapping denominada MATE (Mapping Algorithm based on Task dEpendencies), que detecta las cadenas de tareas más dependientes y las ubica en el mismo procesador [RRS+02].

Basándose en este objetivo, la estrategia MATE introduce los siguientes refinamientos comparada con la anterior.

- El tratamiento de las tareas no se hace directamente por pares de adyacentes, si no que se realiza para cada tarea de forma individual siguiendo un orden creciente de nivel.

- Se prioriza la asignación de las tareas que dan un mejor tiempo de ejecución junto, frente a las que dan mejor tiempo de ejecución separado.

- Esta es una estrategia basada en lista dinámica, ya que el valor de prioridad de las tareas de cada nivel se va actualizando conforme se van asignando las mismas.

4.3.1 Algoritmo de asignación

El algoritmo de asignación que implementa la estrategia MATE parte de una lista inicial donde las tareas están ordenadas de menor a mayor valor de nivel $LN(Ti)$. A partir de dicha lista, las tareas se tratan y se asignan por niveles tal como se expone a continuación.

Para un nivel concreto las tareas se ordenan de forma decreciente según el valor de máxima ganancia, denotado como $max\_gain(Ti)$. Dicho valor indica la ganancia en tiempo que puede ser obtenida cuando una tarea Ti se ubica al mismo procesador que una tarea adyacente Tj previamente asignada. Para calcular el valor de máxima ganancia se definen los siguientes parámetros de tiempo.

- *Tiempo acumulado junto (tac_junto(Tj,Ti)).* Corresponde al mínimo tiempo necesario para ejecutar la tarea Ti en el mismo procesador asignado a una tarea adyacente Tj. El valor de $tac\_junto(Tj, Ti)$ para cada par de tareas se calcula como la suma de los tiempos de cómputo de las dos tareas más el tiempo acumulado de Tj, $acum(Tj)$, como consecuencia de asignar otras tareas adyacentes a Tj en su mismo procesador.

  En realidad el valor de tiempo acumulado junto es el de tiempo junto dado en la expresión 4.2 más el acumulado. El hecho de acumular para cada tarea el tiempo de las adyacentes asignadas en su mismo procesador, es el refinamiento que utilizará el algoritmo para evitar asignar demasiadas tareas a un mismo procesador cuando se dan fuertes dependencias en la mayoría de ellas, como es el caso de situaciones cercanas al modelo TPG.

- *Tiempo separado ($t\_sep(Tj, Ti)$).* Este es el tiempo de ejecución separado de las tareas adyacentes que se calcula mediante la expresión 4.3 descrita anteriormente.

- *Tiempo diferencia ($t\_dif(Tj, Ti)$).* Para cada par de tareas adyacentes (Tj, Ti), con Tj ya asignada, el valor de $t\_dif(Tj, Ti)$ indica la diferencia de tiempo existente entre tiempo separado y tiempo acumulado junto.

$$t\_dif(Tj, Ti) = t\_sep(Tj, Ti) - tac\_junto(Tj, Ti) \tag{4.7}$$

La máxima ganancia de cada tarea Ti, $max\_gain(Ti)$, se define como el máximo valor de tiempo diferencia de la tarea Ti con sus adyacentes ya asignadas, y se calcula mediante la siguiente expresión.

$$max\_gain(Ti) = max_{Tj \in P\_as(Ti)}(t\_dif(Tj, Ti)) \tag{4.8}$$

donde $P\_as(Ti)$ es el conjunto de tareas adyacentes a Ti ya asignadas.

El algoritmo de la Figura 4.5 muestra los pasos para realizar el cómputo de $max\_gain(Ti)$, y además indica mediante la variable de T_ad_i la tarea adyacente que ha producido la máxima ganancia.

```
max_gain(Ti) = 0
para cada Tj → Ti con Tj ya asignada
  tac_junto=w(Tj)+w(Ti)+acum(Tj)
  t_sep=w(Tj)+w(Ti)+c(Tj,Ti)+c(Ti,Tj)-p(Tj,Ti)×w(Ti)
  t_dif=t_sep - tac_junto
  si (t_dif>0 and max_gain(Ti) <t_dif)
    max_gain(Ti)=t_dif
    T_ad_i=Tj
  fin_si
fin_para
```

Figura 4.5: Cómputo de $max\_gain(Ti)$

La ubicación específica de las tareas a los procesadores se lleva a cabo con el algoritmo de la Figura 4.6, que se realiza con los siguientes pasos:

(1) Inicialización de los valores de nivel de cada tarea, máximo nivel (max_level) del grafo, cómputo acumulado ($acum(Ti)$) de cada tarea y cómputo acumulado a cada procesador Pi.

(2) Asignación de las tareas en orden creciente de nivel.

(3) Para un determinado nivel LN_k, se crea una lista de tareas pendientes de asignar en orden decreciente de máxima ganancia.

(4) Asigna la primera tarea Ti de la lista 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, T_ad_i, y ambas se asignan al mismo procesador (P(Ti)=P(T_ad_i)). En caso contrario la tarea Ti se asigna al procesador menos cargado.

```
(1)  sea LN_k el conjunto de tareas con LN(T) = k
     max_level = max(k)
     acum(Ti) = 0 para cada tarea Ti
     Pi=0 para cada Pi ∈ P
(2)  para k=0 hasta max_level
(3)    mientras LN_k ≠ ∅
         para cada T_i ∈ LN_k calcular max_gain(Ti)
         crear lista ordenada de LN_k
(4)      Ti=primera tarea de la lista
         si (max_gain(Ti) > 0)
           P(Ti) = P(T_ad_i)
           acum(T_ad_i) = acum(T_ad_i) + w(Ti)
         si no
           P(Ti)=procesador menos cargado
         fin_si
         actualizar cómputo P(Ti)
         LN_k = LN_k − {Ti}
       fin_mientras
     fin_para
```

Figura 4.6: Algoritmo de la estrategia MATE.

En la Tabla 4.5 se ilustran los pasos de asignación que sigue el algoritmo de la estrategia MATE para asignar el grafo TTIG de la Figura 4.1 a dos procesadores {P0,P1}. Los datos que se muestran en la tabla, relativos al desarrollo del algoritmo, son los siguientes:

- Paso: número de paso de asignación del algoritmo.

- LN_i: número de nivel de las tareas de la lista.
- pend: corresponde al conjunto de tareas pendientes de asignar del nivel LN_i.
- $max\_gain(Ti)$: valor de $max\_gain(Ti)$ para cada una de las tareas de la lista. Se indica además la tarea T_ad_i en los casos en que se obtiene un valor positivo de $max\_gain(Ti)$, con la notación $T\_ad_i : max\_gain(Ti)$.
- asig.: Asignación que se realiza de la primera tarea de la lista, que será la tarea con mayor valor de $max\_gain(Ti)$.
- acum(Ti): valor de cómputo acumulado de la tarea Ti como consecuencia de asignar tareas adyacentes a ella al mismo procesador. En la tabla se indica únicamente el momento en que una tarea actualiza este valor de cómputo acumulado.
- Pi: cómputo acumulado asignado al procesador Pi.

Tabla 4.5: Desarrollo del algoritmo MATE para dos procesadores.

Paso	**LN$_i$**	**pend.**	***max_gain(Ti)***	**asig.**	**acum(Ti)**	**P0**	**P1**
1	0	{T1}	{0}	P0:T1		50	0
2	1	{T2,T3}	{T1:20,T1:30}	P0:T3	T1=100	150	0
3	1	{T2}	{0}	P1:T2		150	250
4	2	{T4,T5,T6}	{0,0,T3:10}	P0:T6	T3=200	350	250
5	2	{T4,T5}	{0,0}	P1:T4		350	330
6	2	{T5}	{0}	P1:T5		350	630
7	3	{T7}	{T6:30}	P0:T7	T6=200	550	630

Mediante el análisis de los datos reflejados en la tabla, vemos que empezando por el nivel 0, en el primer paso el algoritmo realiza la asignación de la tarea T1, al procesador P0. En el segundo paso, las tareas $T2$ y $T3$ dan un valor de $max\_gain(T2) = 20$ y $max\_gain(T3) = 30$ respecto a $T1$, por lo tanto T3 será la primera tarea de la lista y se asigna al mismo procesador que T1, provocando una valor acumulado(T1) de 100 unidades de tiempo. Este valor acumulado hace que en el tercer paso el valor de $max\_gain(T2)$ sea 0 y se asigna la tarea al procesador menos cargado que es P1.

Siguiendo este mecanismo en los pasos sucesivos del algoritmo, se han asignado las tareas por orden de nivel y prioridad dentro del nivel, dando una asignación final, que para el grafo del ejemplo será: P0:(T1,T3,T6,T7) y P1:(T2,T4,T5).

Como puede verse en este ejemplo, el algoritmo MATE detecta la cadena de tareas más dependientes del grafo y las asigna al mismo procesador. Esta característica se continua dando cuando la asignación se realiza con más procesadores, tal como se muestra en la Tabla 4.6, en la que se dan las asignaciones que se obtienen al ir aumentando los procesadores hasta cinco. Se muestra también el cómputo acumulado de cada procesador para cada asignación. Puede observarse que al conservar juntas las tareas más dependientes, al aumentar a cinco procesadores hay uno que queda libre.

Tabla 4.6: Asignaciones obtenidas para el algoritmo MATE al variar el número de procesadores.

N. proc.	**Asignación**	**Cómputo acumulado**
2	P0:(T1,T3,T6,T7) P1:(T2,T4,T5)	P0:550 P1:630
3	P0:(T1,T3,T6,T7) P1:(T2) P2:(T4,T5)	P0:550 P1:250 P2:380
4	P0:(T1,T3,T6,T7) P1:(T2) P2:(T4) P3:(T5)	P0:550 P1:250 P2:80 P3:300
5	P0:(T1,T3,T6,T7) P1:(T2) P2:(T4) P3:(T5) P4:( )	P0:550 P1:250 P2:80 P3:300 P4:0

4.3.2 Análisis de la complejidad

Para el análisis de la complejidad consideramos un grafo TTIG de n nodos repartidos en l niveles con $1 \leq l \leq n$. Esto implica una media de n/l tareas por cada nivel. El algoritmo desarrollado para implementar la estrategia MATE, realiza los siguientes pasos para cada tarea:

1. *Calcular máxima ganancia de las tareas del nivel pendientes de asignar.*

   Para cada tarea T, el valor de máxima ganancia se deduce a partir de la evaluación de los tiempos acumulado junto y separado con todas sus adyacentes. Por lo tanto, este paso necesitará un número de iteraciones

igual a $(n/l) \cdot k$, donde k es el número de pares de tareas adyacentes del nivel.

2. *Crear lista ordenada.*

   El algoritmo de ordenación utilizado tiene una complejidad media asociada de $m \cdot \log m$, donde m es el número de elementos a ordenar. Considerando una repartición equilibrada de las tareas en niveles, la complejidad asociada a la ordenación será $(n/l)(\log n/l)$.

3. *Asignar primera tarea de la lista.*

   Este paso se realiza para una única tarea por lo tanto el coste asociado será 1.

El número de pasos que lleva a cabo el algoritmo globalmente puede calcularse mediante la siguiente expresión:

$$n.niveles \times n.tareas\_nivel(n.pasos\_tarea)$$

A partir de esta expresión, y de los pasos previos estudiados para cada tarea, se deduce que el número de iteraciones que realiza el algoritmo será:

$$l \cdot \frac{n}{l}(\frac{n}{l} \cdot k + \frac{n}{l} \log \frac{n}{l}) \tag{4.9}$$

Considerando los dos casos extremos en cuanto a topología y conectividad del grafo, pueden darse los siguientes casos:

a) $l = n$ y $k = 1$. Grafo de n niveles con una tarea en cada nivel, donde cada tarea tiene una adyacente de nivel superior. En este caso el número de iteraciones asociado al algoritmo será n.

b) $l = 1$ y $k = \frac{n^2-n}{2}$. Grafo con un nivel de n tareas y máxima conectividad, es decir cada tarea tiene $(n-1)$ adyacentes. Sustituyendo estos valores en la expresión 4.9 se obtiene un número de iteraciones igual a $\frac{(n^4-n^2)}{2} + n^2 \log n$.

Vemos pues que en función de la distribución de las tareas en niveles, y la topología de interconexión de las tareas, el orden complejidad asociada al algoritmo puede sufrir grandes variaciones, oscilando entre una complejidad mínima de $O(n)$ hasta una complejidad máxima de $O(n^4)$.

Para una suposición intermedia más realista, en la que se trata de un grafo TTIG con l niveles y un número de adyacentes de cada tarea igual a k con $n > k > 0$, operando con los términos de la expresión 4.9, se obtiene la complejidad media siguiente.

$$\frac{k}{l}n^2 + \frac{n^2}{l}\log\frac{n}{l} \tag{4.10}$$

Podemos deducir pues que en su caso más general este algoritmo tendrá una complejidad media del orden de $O(n^2 \log n)$.

4.4 Robustez de la política de mapping

En el diseño de algoritmos automáticos de mapping, una característica importante de analizar es su robustez. Entendemos como aspectos relativos a la robustez, los que tratan de la sensibilidad de la política frente a las variaciones en la estimación de los parámetros del modelo de entrada. En nuestro caso, los parámetros que se extraen del programa para construir el modelo son el tiempo de cómputo y el volumen de comunicación. El grado de paralelismo se deduce a partir del cómputo y la situación de las interacciones dentro de las tareas.

Puesto que nos movemos en el campo del mapping estático, presuponemos que las variaciones que se pueden dar en la estimación de cómputo y comunicación son pequeñas, y estudiamos la influencia de estas variaciones en la asignación para la política MATE que, de las dos políticas propuestas, es la que tiene mayor nivel de refinamiento.

Como hemos visto en el proceso de asignación, el algoritmo MATE asigna cada par de tareas adyacentes juntas al mismo procesador o bien separadas, en función de los valores de tiempo junto y tiempo separado, siendo irrele-

vante el procesador concreto al que se asignan las tareas. Bajo esta premisa, para el análisis de la robustez se evalúa el número de cambios que se dan en la asignación de las tareas adyacentes respecto a la decisión de juntarlas o separarlas.

4.4.1 Valoración global del grafo TTIG

Para valorar adecuadamente los resultados en cuanto al número de cambios que se producen en las asignaciones, una información importante a tener en cuenta es disponer de las características globales del grafo que se está analizando. Es decir, cuando se trata de grafos TTIG donde la mayoría de las tareas son igual de dependientes, y existe poca diferencia entre los valores de tiempo junto y tiempo separado, es de esperar que las variaciones en los parámetros de entrada podrán afectar a la asignación de cualquier par de adyacentes por igual, sin que ello signifique que se dan cambios importantes en el tiempo de ejecución final. Si por el contrario se trata de grafos cuyas tareas adyacentes tienen claras diferencias en los valores de tiempo junto y tiempo separado, será de esperar que las asignaciones no se vean tan influenciadas por los pequeños cambios en los valores de entrada, puesto que esto podría repercutir en importantes cambios en el tiempo de ejecución.

Para realizar una valoración global del grafo TTIG se aplica en primer lugar un proceso de normalización del mismo, a partir del cual se definen dos métricas relativas al comportamiento global, tal como se expone a continuación.

(a) Normalización del grafo TTIG.

Para normalizar el grafo TTIG los valores de cómputo y comunicación del mismo se expresan dentro del intervalo [0,100]. Ello nos permitirá comparar posteriormente las características de un grafo respecto otro en función de las métricas de valoración global que se definen. Cabe señalar que el grado de paralelismo es un valor ya normalizado entre 0 y 1, y por lo tanto no se le aplica ninguna transformación porque para el grafo normalizado continua valiendo lo mismo.

Los valores de cómputo y comunicación del grafo TTIG normalizado se

calculan como $(x \times 100)/v\_max$, donde x es el valor de cómputo o comunicación del grafo TTIG inicial y v_max corresponde al valor máximo de cómputo y comunicación. En el grafo TTIG de la Figura 4.1 se tiene que $v\_max = 300$, que corresponde al tiempo de cómputo de la tarea T5. A partir de éste valor se obtiene el grafo TTIG normalizado de la Figura 4.7, donde como puede observarse todos los valores normalizados están comprendidos en el intervalo [0,100].

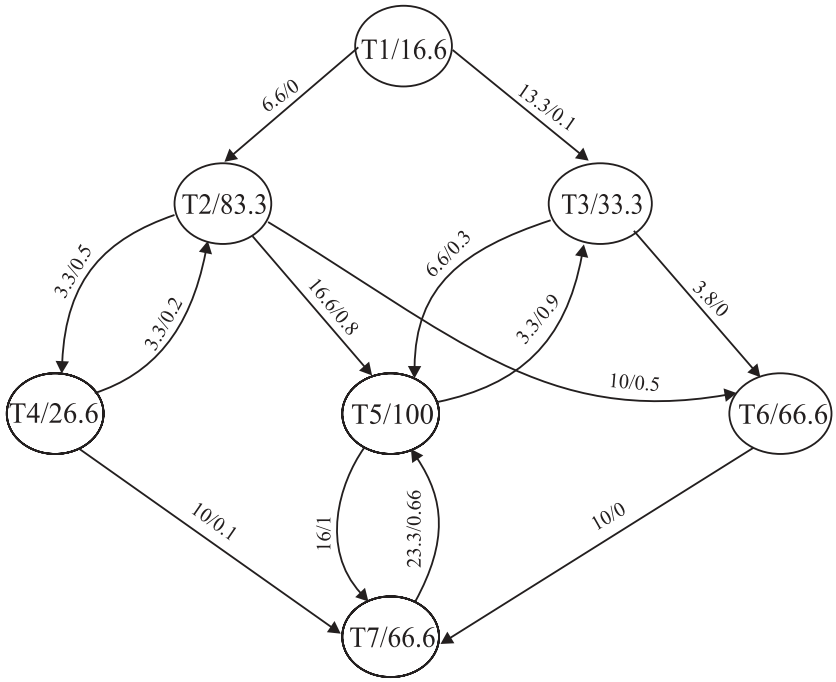

Figura 4.7: Grafo TTIG normalizado.

A partir del grafo normalizado se calculan los valores de tiempo de ejecución junto y separado de las tareas adyacentes, calculados según las expresiones 4.2 y 4.3. En la Tabla 4.7 se reflejan estos valores ordenados de mayor a menor valor de diferencia t_dif_i, donde

$$t\_dif_i = |t\_junto(Ti, Tj) - t\_sep(Ti, Tj)| \tag{4.11}$$

Substituyendo los valores de tiempo junto y tiempo separado por sus expresiones equivalentes 4.2 y 4.3 del Apartado 4.1.2, se obtiene la siguiente

Tabla 4.7: Tareas adyacentes del grafo TTIG normalizado ordenadas por valor de diferencia.

(Ti,Tj)	$t\_junto(Ti,Tj)$	$t\_sep(Ti,Tj)$	t_dif_i
(T2,T5)	183.3	120	63.3
(T5,T7)	166.6	140	26.6
(T2,T6)	150	126.6	23.3
(T3,T5)	133.3	113.3	20
(T6,T7)	133.3	143.3	10
(T1,T3)	50	60	10
(T1,T2)	100	106.6	6.6
(T2,T4)	110	103.3	6.6
(T4,T7)	93.3	96.6	3.3
(T3,T6)	100	103.3	3.3

expresión:

$$t\_dif_i = |p(Ti,Tj) \cdot w(Tj) - (c(Ti,Tj) + c(Tj,Ti))| \tag{4.12}$$

En el caso general de un grafo TTIG normalizado, el valor de t_dif_i puede alcanzar un máximo de 200, que corresponde al caso en que $c(Ti,Tj) = c(Tj,Ti) = 100$ y $p(Ti,Tj) \cdot w(Tj) = 0$. Así pues, la diferencia de tiempo junto y tiempo separado asociados a los arcos del TTIG normalizado será un valor comprendido dentro del intervalo [0,200]. Las tareas adyacentes con valor t_dif_i en la zona baja de este intervalo serán aquellas cuya asignación puede ser cambiada al aplicar pequeñas variaciones en los parámetros de entrada.

(b) Métricas de valoración de diferencias.

Para dar una valoración de cada grafo en su conjunto respecto a los valores de diferencia t_dif_i, y la relación existente entre estas diferencias, se definen los siguientes parámetros:

- *Diferencia media (dif_media(G)).* Será la media aritmética de los valores de diferencia de todo el grafo. Para un grafo G con m pares de tareas adyacentes se calcula como:

$$dif\_media(G) = \frac{t\_dif_1 + t\_dif_2 + .. + t\_dif_m}{m} \tag{4.13}$$

El valor de diferencia media da una idea de cómo son en su mayoría las tareas adyacentes, es decir si marcan una diferencia clara o no, con sus valores de tiempo junto o separado. Cuanto mayor es el valor de diferencia media del grafo, significa que ha de ser menos propicio a tener variaciones en la asignación.

- *Decremento progresivo (dec_prog(G))*. A partir de los valores de diferencia ordenados de mayor a menor de la forma $t\_dif_1 > t\_dif_2 > .. > t\_dif_m$, se calcula el valor de decremento de cada una con la siguiente, según se indica a continuación.

  $dec\_dif_1 = t\_dif_1 - t\_dif_2$
  $dec\_dif_2 = t\_dif_2 - t\_dif_3$
  .....
  $dec\_dif_{m-1} = t\_dif_{m-1} - t\_dif_m$

  Definimos el decremento progresivo del grafo, como la media aritmética respecto al número de adyacentes de los valores dec_dif_i calculados. Dicho valor se calcula mediante la siguiente expresión:

$$dec\_prog(G) = \frac{dec\_dif_1 + dec\_dif_2 + .. + dec\_dif_{m-1}}{m} \tag{4.14}$$

  El valor $dec\_prog(G)$ da información de cómo son unas diferencias respecto a otras en el grafo. Así pues, valores altos de decremento progresivo indicarán que coexisten valores de diferencia altos con valores de diferencia bajos en el mismo grafo. Por otro lado, valores bajos de *dec_prog* serán indicativos de que todas las diferencias son similares y por lo tanto los cambios de los parámetros del TTIG pueden afectar a cualquiera de ellas por igual.

Para ilustrar el tipo de información que muestran los valores de diferencia y decremento global, consideramos a modo de ejemplo dos grafos G1 y G2 con tres tareas adyacentes para las cuales se tienen los siguientes valores de diferencia.

G1: $t\_dif_1 = 101$ $t\_dif_2 = 1$ $t\_dif_3 = 1$
G2: $t\_dif_1 = 35$ $t\_dif_2 = 35$ $t\_dif_3 = 33$

Los parámetros de comportamiento global asociados a cada grafo serán:

$dif\_media(G1) = 34.3$ $dec\_prog(G1) = 33.3$
$dif\_media(G2) = 34.3$ $dec\_prog(G2) = 0.6$

A raíz de los valores calculados vemos que ambos grafos tienen un valor de diferencia media de 34.3, que permite pensar que no todas las dependencias entre adyacentes son muy fuertes. En cambio, en el caso de G1 el valor de dec_prog es cercano a dif_media, mientras que para G2 el decremento progresivo es cercano a 0. Esto significa que en el caso del grafo G1 las posibles variaciones en los parámetros de entrada probablemente afectarán solo a algunas de sus adyacentes (que en este caso serían t_dif_2 y t_dif_3), mientras que en el caso de G2 pueden verse todas afectadas en la misma medida.

Las métricas de diferencia media y decremento progresivo se utilizarán en el siguiente apartado para analizar la sensibilidad de las asignaciones de la política MATE en relación a las características globales de cada grafo TTIG.

4.4.2 Sensibilidad de la asignación frente a variaciones en los parámetros de entrada

En este apartado queremos mostrar que la política MATE, basada en el grafo TTIG, proporciona asignaciones que son poco sensibles a las pequeñas variaciones en los parámetros de entrada del grafo. Para ello se realiza una clasificación de los grafos según el valor que les corresponde de las métricas definidas de diferencia media y decremento progresivo, y probamos que para los grafos

con valores grandes de $dif\_media(G)$ y $dec\_prog(G)$ se dan muy pocas variaciones en las asignaciones, mientras que para grafos con valores pequeños de estas dos métricas el índice de variaciones aumenta.

Para realizar este análisis se aplica un porcentaje de ruido al tiempo de cómputo de las tareas, denominado rcp, o bien al volumen de comunicación, denominado rcm. En ambos casos, el ruido aplicado está dentro del intervalo $\pm rcp$ o $\pm rcm$ respectivamente. A partir de ahí, se estudia el número de variaciones que se dan en la asignación de las tareas adyacentes respecto de la asignación inicial. Dicho estudio se ha llevado a cabo para un conjunto de doce grafos {GG1,..,GG12} con un número de tareas entre 6 y 16. En todos ellos el tiempo de cómputo global de las tareas del grafo es mayor que el volumen de comunicación global, oscilando el ratio entre cómputo y comunicación de 1.36 a 5.17. De todos modos, la sensibilidad de la política en cada grafo viene determinada por las métricas de valoración global definidas, y en ellas se basa el estudio que se detalla a continuación.

Para valorar el número de variaciones obtenidas en la asignación, al aplicar un porcentaje de ruido en el tiempo de cómputo de las tareas del grafo, se han realizado 10 asignaciones para cada grafo, aplicando un valor de ruido desde $rcp = 10$ hasta $rcp = 100$ en incrementos de 10 unidades. Esto significa que se han aplicado variaciones desde un 10% hasta un 100% en el cómputo global del grafo.

En la Tabla 4.8 se dan los resultados de variaciones obtenidos más las características de cada grafo utilizado, mediante los siguientes campos:

- DIF. Valor de diferencia media del grafo ($dif\_media(G)$).
- DEC. Valor de decremento progresivo del grafo ($dec\_prog(G)$).
- proc. Número de procesadores usados en la asignación.
- $[rcp_i, rcp_j]$. Porcentaje de pares de adyacentes que han variado su asignación respecto a la asignación inicial, al aplicar un valor de ruido de rcp_i y rcp_j respectivamente. Cuando no se produce ninguna variación se indica con un guión. Vemos por ejemplo en la tabla que para el grafo

Tabla 4.8: Porcentaje de variaciones en las asignaciones al aplicar un porcentaje de ruido a los valores de tiempo de cómputo.

Grafo	DIF	DEC	proc	[10,20]	[30,40]	[50,60]	[70,80]	[90,100]
GG1	69	0	2	-	6	8	4	10
			4	10	12	20	18	10
GG2	51	11	2	-	-	25	12	-
			4	-	-	12	12	-
GG3	48	8	2	5	5	5	11	5
			4	5	5	-	11	5
GG4	46	10	2	-	14	28	-	14
			4	-	-	-	-	-
GG5	33	3	2	-	2	16	25	29
			4	10	21	21	18	16
GG6	27	11	2	-	12	25	-	8
			4	-	4	8	8	4
GG7	23	5	2	-	-	-	-	-
			4	-	-	-	-	-
GG8	17	4	2	-	-	-	-	-
			4	-	-	-	-	-
GG9	17	3	2	-	-	-	10	21
			4	-	-	-	-	-
GG10	14	5	2	-	-	-	-	-
			4	-	-	-	-	-
GG11	11	5	2	22	22	36	27	22
			4	-	-	-	36	27
GG12	11	0	2	14	31	31	43	48
			4	31	27	16	29	37

GG1 con dos procesadores, al aplicar un porcentaje de ruido de $rcp = 50$ y $rcp = 60$ se da un 8% de variaciones en la asignación.

A la vista de los resultados reflejados en la tabla podemos ver que al aplicar un porcentaje de variación de hasta $rcp = 20$, la influencia en la asignación final es escasa o nula, exceptuando si cabe los grafos GG11 y GG12, donde la influencia es mayor. Estos dos grafos coinciden con los que tienen un menor valor de diferencia media y de decremento progresivo. Estos resultados en las variaciones se continúan dando incluso al aplicar hasta un porcentaje de ruido del 40%.

Realizando el mismo procedimiento, al aplicar porcentajes de variación, rcm, a los volúmenes de comunicación del grafo TTIG se obtiene un comportamiento aún mucho más robusto del algoritmo de mapping. Por dicho motivo

ya no se muestra la tabla de variaciones y simplemente es preciso indicar que de los grafos estudiados únicamente los grafos GG5, GG6, GG11 y GG12 dan un porcentaje mayor que cero de los pares de adyacentes que han cambiado su asignación respecto a la inicial. Para estos grafos se obtiene un máximo de un 8% de adyacentes con variación para rcm hasta el 40%, y un máximo de 29% de variaciones cuando rcm alcanza hasta el 100%.

Del análisis de los resultados expuestos en la tabla, se ve que las pequeñas variaciones de hasta un 20% en el cómputo global del grafo, no tienen prácticamente incidencia en la asignación final. Por lo que respecta a las variaciones al aplicar un valor de ruido a los volúmenes de comunicación, se muestra un comportamiento aun más estable, puesto que el volumen de comunicación de todos los grafos es menos importante que el tiempo de cómputo.

Como resumen de lo expuesto podemos concluir que el modelo TTIG con el parámetro de grado de paralelismo, facilita la definición de políticas de mapping más robustas. Esto a la vez permite una cierta relajación en la etapa de generación del modelo, ya que la consecución de una buena asignación no será tan dependiente de la exactitud de los parámetros de entrada.