Conclusiones y principales contribuciones

El objetivo de este trabajo de tesis ha sido contribuir en la resolución del problema del mapping para aplicaciones que tienen un patrón de interacciones arbitrario. Esto ha supuesto la definición de un nuevo modelo para representar el comportamiento de dichas aplicaciones, y la propuesta de nuevos algoritmos de asignación adecuados a dicho modelo.

Para la realización del trabajo ha sido necesario el desarrollo de los siguientes puntos relacionados con la definición del modelo, las políticas de mapping y la experimentación con las mismas. En cada uno de los puntos se citan además las aportaciones a que han dado lugar.

(a) **Con respecto al modelo**, se han analizado los modelos clásicos usados para la representación de los programas paralelos TPG (Task Precedence Graph) y TIG (Task Interaction Graph), junto con las políticas y herramientas de mapping desarrolladas para dichos modelos. De cada modelo se ha razonado su habilidad para capturar el comportamiento temporal de las aplicaciones en función de su patrón de interacción entre tareas.

A partir del análisis de las soluciones clásicas, se ha definido un nuevo modelo de representación de programas paralelos denominado Temporal Task Interaction Graph (TTIG), que incluye como nuevo parámetro el grado de paralelismo entre tareas adyacentes. Con este parámetro el modelo TTIG engloba las ventajas ofrecidas por cada uno de los modelos clásicos TPG y TIG, y constituye una solución unificada de los mismos,

que permite la representación de aplicaciones paralelas con un patrón de interacción de tareas arbitrario. La definición formal del modelo TTIG se ha publicado en:

C. Roig, A. Ripoll, M. A. Senar, F. Guirado, E. Luque.
Modelling Message-Passing Programs for Static Mapping.
IEEE Proc. 8th. Euromicro Workshop on Par. and Distr. Processing (PDP). Jan. 2000. pp. 229-236.

Para la construcción del modelo TTIG se parte de la información del comportamiento temporal de las tareas de la aplicación. Mediante el análisis de dicha información, se ha propuesto un mecanismo para calcular los valores de cota teórica mínima y máxima sobre el número de procesadores que permitan ejecutar la aplicación en un tiempo mínimo.

(b) **Con respecto a las estrategias de mapping**, se han definido dos algoritmos de mapping de distinta complejidad denominados TASC y MATE, que utilizan como modelo el grafo TTIG. El algoritmo TASC basa su estrategia en explotar la concurrencia entre tareas, evaluando la posibilidad de juntar al mismo procesador o separar a distintos procesadores cada par de tareas adyacentes.

El algoritmo MATE constituye una técnica con mayor nivel de refinamiento, cuya estrategia se basa en asignar cada tarea al mismo procesador que su tarea adyacente más dependiente. La política MATE ha sido publicada en:

C. Roig, A. Ripoll, M. A. Senar, F. Guirado, E. Luque.
A new model for static mapping of parallel applications with task and data parallelism.
IEEE proc. Int. Conf. Parallel and Distributed Processing (IPDPS). April. 2002. ISBN 0-7695-1573-8.

De la política MATE, que es la de mayor nivel de refinamiento y com-

plejidad, se ha estudiado además su sensibilidad a las variaciones en la estimación de los parámetros del modelo de entrada. Se ha comprobado que al aplicar unas variaciones de hasta el 40% en el cómputo global del grafo o bien en las comunicaciones globales, las asignaciones se mantienen estables. Esto implica que la generación de una asignación eficiente no se verá influenciada por la exactitud de los valores obtenidos en la fase de obtención de parámetros.

(c) **Con respecto a la experimentación realizada**, se ha evaluado en primer lugar la bondad de las políticas TASC y MATE, comparándolas con la asignación óptima para un conjunto de grafos con dimensiones reducidas (con un máximo de diez tareas). Las desviaciones medias obtenidas respecto de la asignación óptima alcanzan un 14% en los grafos con comportamiento arbitrario y un 5% en los grafos con comportamiento homogéneo. Estos resultados verifican la bondad de las estrategias de mapping propuestas para los grafos estudiados. La asignación óptima nos ha permitido además evaluar la adecuación de los valores de cota teórica del número de procesadores, y la bondad de las asignaciones al usar el número de procesadores indicado por las cotas.

El siguiente paso de experimentación ha consistido en evaluar la bondad de las políticas TASC y MATE para un conjunto de aplicaciones de paso de mensajes en un cluster de PCs. La efectividad de TASC y MATE se ha comparado con las políticas basadas en los modelos clásicos. Por un lado se han usado aplicaciones sintéticas en C+PVM con comportamiento arbitrario y con comportamiento homogéneo. Por otro lado, se ha experimentado con una aplicación real de procesamiento de imágenes denominada BASIZ.

Las políticas clásicas se han visto mejoradas claramente por TASC y MATE para las aplicaciones con comportamiento arbitrario y para la aplicación BASIZ, dando unos porcentajes medios de ganancia, con respecto el mapping basado en el modelo TIG, del 20% para TASC y del 27% para MATE. En cambio, para las aplicaciones cuyo comportamiento se ajusta a uno de los modelos clásicos se obtienen unos resultados que

son comparables a la técnica específica de cada modelo. De este modo hemos verificado que las políticas de mapping basadas en el modelo TTIG ofrecen buenos resultados para cualquier tipo de comportamiento de las aplicaciones.

En las dos siguientes publicaciones se ha mostrado la efectividad de usar el modelo TTIG para el mapping en un entorno cluster, para aplicaciones sintéticas con comportamiento arbitrario y con comportamiento homogéneo respectivamente.

C. Roig, A. Ripoll, M. A. Senar, F. Guirado, E. Luque.
Exploiting Knowledge of Temporal Behaviour in Parallel Programs for Improving Distributed Mapping.
6th. Int. Euro-Par Conference. LNCS Vol. 1900. pp. 262-271. 2000.

C. Roig, A. Ripoll, M. A. Senar, F. Guirado, E. Luque.
Improving static scheduling using inter-task concurrency measures.
IEEE Proc. Int. Conf. Parallel Processing (ICPP-2001). Workshop on scheduling and resource management for cluster computing. Set. 2001. pp. 375-381.

Con respecto a la aplicación real BASIZ, en la siguiente aportación se publicaron sus características de comportamiento, la obtención del modelo TTIG que le corresponde, y el análisis del proceso de mapping para la misma.

C. Roig, A. Ripoll, J. Borrás, E. Luque.
Efficient mapping for Message-passing applications using the TTIG model: a case study in image processing
Proc. 8th Euro PVM/MPI. Sep. 2001. LNCS vol. 2131, pp. 370-377.

La asignación y ejecución de las aplicaciones en los procesadores del cluster se ha realizado de forma automática mediante la herramienta AMEEDA. Esta es una herramienta de mapping de propósito general

para entornos cluster. En dicha herramienta se ha integrado un mecanismo para obtener los parámetros incluidos en el modelo TTIG, partiendo de la traza de ejecución de las aplicaciones con paso de mensajes. La exposición detallada del proceso automático de obtención del modelo TTIG se encuentra en el siguiente report interno.

C. Roig, A. Ripoll, M. A. Senar, F. Guirado, E. Luque.
A new model for static mapping of parallel applications with task and data parallelism.
DIEI-02-RT-1. Tech Rep. Dep. of Computer Science. Univ. de Lleida. (Spain). 2002.

La publicación de la herramienta AMEEDA y sus características ha sido aceptada en:

X. Yuan, C. Roig, A. Ripoll, M. A. Senar, F. Guirado, E. Luque.
AMEEDA: A general-purpose mapping tool for parallel applications on dedicated clusters.
Aceptado en Euro-Par 2002.

Para probar la bondad de las políticas TASC y MATE al aumentar el número de procesadores, se ha realizado un estudio de la escalabilidad en el valor de *speedup* obtenido en las ejecuciones, mediante la herramienta de simulación ESPPADA. Ambas políticas han mostrado ser capaces de ir incrementando el speedup en función de la capacidad de concurrencia intrínseca de cada aplicación.

Finalmente, y como resultado añadido, se ha comprobado la utilidad del modelo TTIG y de las políticas TASC y MATE para resolver la etapa de cluster-mapping para el modelo TPG. En la solución de esta etapa se obtienen tiempos de ejecución claramente mejores que cuando se resuelve con una política basada en el modelo TIG, que es como clásicamente se ha venido haciendo.

Lineas abiertas

A partir de la experiencia adquirida en el desarrollo de esta tesis, se han ido viendo las siguientes nuevas lineas de actuación, que pueden contribuir a completar el nivel de refinamiento y la amplitud de la solución al problema sobre el que se ha trabajado.

- Incorporar las características de la plataforma hardware y la red de interconexión en las políticas de mapping.
- Extender el modelo TTIG para que pueda modelar tanto aplicaciones con paralelismo de tareas, como aplicaciones con paralelismo de datos y con paralelismo mixto.
- Desarrollar una metodología para realizar predicciones de rendimiento a partir del modelo utilizado, que permita hacer estimaciones tanto del número de recursos necesarios para ejecutar una aplicación como del tiempo de ejecución que se obtendrá dados unos recursos limitados.

Apéndice A

Ejemplo de generación del comportamiento temporal a partir de la traza

De las alternativas existentes para extraer estáticamente el comportamiento temporal de un programa resumidas en la Sección 2.2, en nuestro grupo se ha automatizado el proceso de obtención de los parámetros de comportamiento a partir de la traza de ejecución.

En este apéndice se muestra a título de ejemplo, el proceso de obtención del comportamiento de la tarea T2 del programa sintético mostrado en la Figura 2.2. La traza de ejecución se ha generado a partir del código fuente en C+PVM de la tarea que se muestra en la Figura A.1. En dicho código se puede ver que las fases de cómputo se realizan mediante un bucle de sumas, y las comunicaciones se llevan a cabo con las primitivas de envío y recepción *pvm_send* y *pvm_recv* respectivamente. Cabe señalar que las dos primeras sentencias de recepción que se encuentran en el programa, corresponden a la recepción de los identificadores de las dos tareas adyacentes a T2, enviados por una tarea adicional, que no aparece en la Figura 2.2. Dicha tarea únicamente realiza la activación del resto de tareas y envía los identificadores de las mismas.

Mediante la instrumentación del código realizada con TapePVM [Mai95], se ha generado la traza de ejecución, que para la tarea T2 es la que se muestra

```
#include <stdio.h>
#include "pvm3.h"
main() {
   int tasca_jo,cc,task0,task1,j,i,k,info,ptid,vector[1000000];
   pvm_setopt(PvmRoute,PvmRouteDirect);
   tasca_jo=pvm_mytid();

   ptid=pvm_parent();
   cc=pvm_recv(ptid,1);
   info=pvm_upkint(&task0,1,1);
   cc=pvm_recv(ptid,2);
   info=pvm_upkint(&task1,1,1);

   cc=pvm_recv(task1,1);    /* recepcion de task1 */
   info=pvm_upkint(&vector[0],25000,1);
   k=1;
   while (k<=13) {  /* computo */
        j=1;
        while (j<=500000000) j=j+1;
        k=k+1;
   }
   cc=pvm_recv(task0,1);  /* recepcion de task0 */
   info=pvm_upkint(&vector[0],50000,1);
   k=1;
   while (k<=14) {
        j=1;
        while (j<=500000000) j=j+1;
        k=k+1;
   }
   info=pvm_initsend(PvmDataDefault);  /* envio a task0 */
   pvm_pkint(&vector[0],50000,1);
   pvm_send(task0,1);
   j=1;
   while (j<=1000000000) j=j+1;  /* computo */
   pvm_exit();
   exit(0); }
```

Figura A.1: Código C+PVM de la tarea T2 del programa de la Figura 2.2.

en la Figura A.2.

```
Task 2
    Recv message 35 (4 bytes) from task 3 (Send found)
          (task:2, file:4,line:31,date_s:0,date_u:572029)
          (delay_s:0, delay_u:27)
    Recv message 36 (4 bytes) from task 3 (Send found)
          (task:2, file:4,line:35,date_s:0,date_u:572373)
          (delay_s:0, delay_u:265)
    Recv message 37 (100000 bytes) from task 1 (Send found)
          (task:2, file:4,line:39,date_s:0,date_u:572985)
          (delay_s:2107, delay_u:177440)
    Recv message 13 (200000 bytes) from task 0 (Send found)
          (task:2, file:4,line:62,date_s:2510,date_u:975643)
          (delay_s:0, delay_u:6161)
    Send message 34 (200000 bytes) to task 0 (Recv found)
          (task:2, file:4,line:86,date_s:2945,date_u:840870)
          (delay_s:0, delay_u:585518)
    Quit
          (task:2, file:4,line:111,date_s:3070,date_u:892147)
          (delay_s:0, delay_u:0)
```

Figura A.2: Traza de ejecución para la tarea T2.

Como se puede observar, en la traza se capturan los eventos de comunicación con el volumen de bytes involucrados en las transferencias. Los valores de tiempo se dan relativos al tiempo de inicio de la tarea mediante los campos date_s y date_u, que expresan el valor de tiempo en segundos y microsegundos respectivamente. Del mismo modo, los intervalos de tiempo en los cuales la tarea está en espera se denotan como delay_s y delay_u.

El intervalo de tiempo dedicado a ejecutar cada fase de cómputo existente entre dos primitivas de paso de mensajes, se calcula mediante la diferencia de tiempo existente entre ambas, a la que hay que restarle además el tiempo de espera. Así por ejemplo, la duración de la fase de cómputo que se da entre las primitivas Recv message 13 y Send message 34 se calcula como la diferencia de tiempos resultante de restar (2945 segundos y 840870 microsegundos) menos (2510 segundos y 975643 microsegundos). Al valor de diferencia habrá que restarle además el tiempo de espera especificado de 6161 microsegundos. El

tiempo resultante para esta fase de cómputo será de 434.859.066 microsegundos.

Todos los registros de la traza son interpretados automáticamente por una herramienta desarrollada para tal fin, que genera la estructura de tareas del programa codificada en lenguaje intermedio [Cab01]. En nuestro caso, el lenguaje intermedio escogido es el XML (eXtensible Markup Language) [Lan98], puesto que permite una representación fácil de usar y de entender una vez escrito. Otra facilidad que aporta dicho lenguaje es que permite crear una estructura lógica fácilmente tratable al existir herramientas informáticas creadas específicamente para ello.

Utilizando lenguaje XML se han definido un conjunto de estructuras representativas de las primitivas básicas que se utilizan en lenguaje de programación, así como la estructura básica en forma de tareas del programa. En la Figura A.3 se muestran las estructuras definidas y su significado.

<program name="nombre programa">	Definir programa
</program>	Final Programa
<task id="id_tarea">	Definir tarea
</task>	Final tarea
<exec run="microseconds"/>	Fase de cómputo
<send id="id"task="rec_task"vol="volume"/>	Envío de datos
<receive id="id"task="task_emisora"/>	Recepción de mensaje
<join group="group_name"/>	Añadir al grupo de tareas
<leave group="group_name"/>	Abandonar grupo de tareas
<barrier group="group_name"count="n_tasks"/>	Barrera de sincronización de grupo
<if prob="probability"/> <true> ..... </true> <false> ..... </false> </if>	Estructura condicional que permite la ejecución de primitivas englobadas en las subestructuras true y false con una probabilidad
<for count="cont"> ..... </for>	Estructura iterativa que ejecuta las primitivas que engloba un número *cont* de veces

Figura A.3: Primitivas implementadas de XML para modelar el comportamiento de los programas.

Realizando la interpretación automática de los campos reflejados en la traza con el procedimiento descrito anteriormente, se genera el comportamiento de cada tarea expresado en lenguaje XML. En la Figura A.4 se muestra en XML el comportamiento deducido para la tarea T2.

```
<task id="2">
    <exec run="572029"/>
    <receive id="35" task="3"/>
    <exec run="317"/>
    <receive id="36" task="3"/>
    <exec run="347"/>
    <receive id="37" task="1"/>
    <exec run="403225218"/>
    <receive id="13" task="0"/>
    <exec run="434859066"/>
    <send id="34" task="0" vol="4000000"/>
    <exec run="124465759"/>
</task>
```

Figura A.4: Programa XML de la tarea T2.

La información del comportamiento del programa codificado en lenguaje XML se utiliza básicamente en el presente trabajo como entrada a los distintos módulos desarrollados para calcular el modelo TTIG que corresponde a la aplicación. Adicionalmente, el código XML de un programa constituye la entrada a la herramienta de simulación ESPPADA (Entorno de Simulación de Programas Paralelos sobre Arquitecturas DistribuidAs) [Gui01b]. Por lo tanto este tipo de codificación nos da la facilidad de definir la estructura de tareas de un programa, y valorar su rendimiento mediante simulación antes de su implementación real.

Apéndice B

Estudio de porqué el grado de paralelismo se calcula sólo entre tareas adyacentes

Para ser capaces de definir políticas de mapping más eficientes para aplicaciones con una estructura de tareas arbitraria, se ha visto la idoneidad de incluir en el grafo de tareas que modela la aplicación el parámetro del grado de paralelismo, que indica la máxima capacidad de concurrencia de las tareas adyacentes.

A parte de esta posibilidad, que ha sido la escogida para construir el modelo TTIG, se han estudiado otras alternativas para capturar el grado de concurrencia también entre tareas no adyacentes. Dichas alternativas se han desestimado por diferentes razones que se exponen en los siguientes apartados.

B.0.2 Grado de paralelismo entre tareas no adyacentes considerando el subgrafo aislado

Tal como se ha visto en el Capítulo 2, el valor de grado de paralelismo que se incluye en el modelo TTIG se calcula a partir de la especificación de tareas del grafo TFG mediante la simulación del subgrafo aislado de cada par de tareas adyacentes.

Extendiendo la misma técnica, una forma para calcular el grado de pa-

ralelismo entre tareas no adyacentes es considerando el subgrafo aislado que les une y realizar su simulación. Esto nos llevaría a obtener el grado de paralelismo entre las tareas no adyacentes con la misma filosofía que la utilizada en el Capítulo 2, pero implica la evaluación de todos los subgrafos que corresponden a los distintos caminos que unen dos tareas. Mostramos esta idea con el ejemplo de grafo TFG de seis tareas de la Figura B.1.

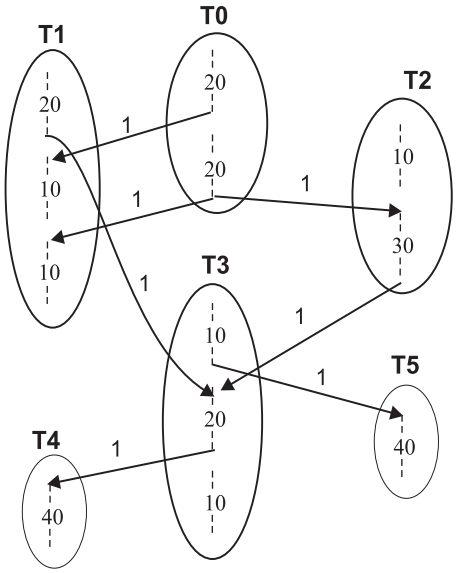

Figura B.1: Grafo TFG de seis tareas.

Vemos en este sencillo ejemplo que para calcular el grado de paralelismo entre las tareas T0 y T3, habría que calcular los grados de paralelismo a partir de la simulación de los subgrafos aislados formados por los caminos (T0 $\rightarrow$ T1 $\rightarrow$ T3) y (T0 $\rightarrow$ T2 $\rightarrow$ T3), y coger el máximo obtenido. Igualmente, para calcular el grado de paralelismo entre T0 y T4 habría que evaluar los subgrafos de los caminos (T0 $\rightarrow$ T1 $\rightarrow$ T3 $\rightarrow$ T4) y (T0 $\rightarrow$ T2 $\rightarrow$ T3 $\rightarrow$ T4).

La evaluación mediante este procedimiento requiere de una gran complejidad, puesto que el número de caminos a evaluar puede ser muy elevado. Por otra parte, en los algoritmos de mapping la posibilidad de concurrencia entre las tareas no adyacentes normalmente sería considerada como un criterio de segundo orden, puesto que tendrá menor influencia que las dependencias entre las tareas que se comunican directamente. Por lo tanto, los beneficios de utilizar estos valores podrían fácilmente no verse compensados por la complejidad

que supone calcularlos.

B.0.3 Grado de paralelismo a partir de la simulación global del grafo

Otra posibilidad planteada para calcular el grado de paralelismo, tanto para tareas adyacentes como no adyacentes, es realizar la simulación global del grafo TFG sin comunicaciones con una tarea por procesador, y calcular el grado de paralelismo entre cualquier par de tareas a partir de dicha simulación. Procediendo de esta forma, la simulación global del grafo de la Figura B.1 se muestra en la Figura B.2.

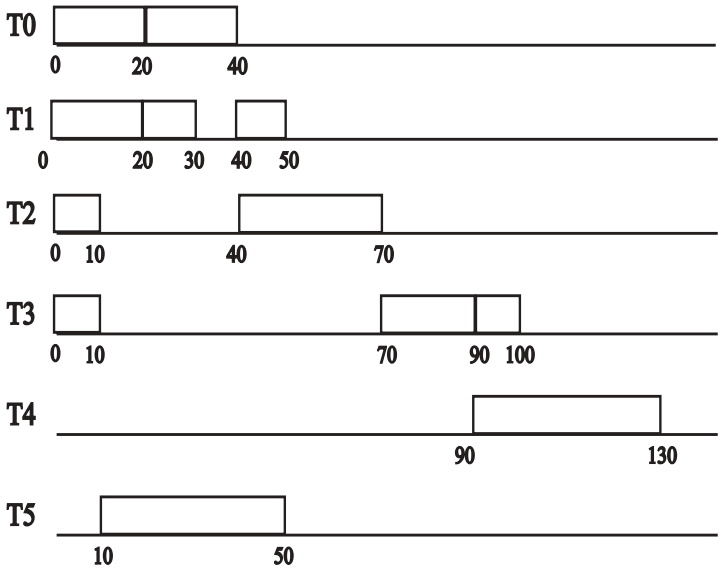

Figura B.2: Simulación global del grafo de la Figura B.1

A partir de los intervalos de ejecución de las tareas se puede calcular el tiempo de ejecución común para cada par de tareas. Vemos por ejemplo que las tareas T2 y T5 ejecutan simultáneamente durante el intervalo de tiempo $[40, 50]$, por lo tanto estas 10 unidades de tiempo de ejecución concurrente supondría un grado de paralelismo entre T2 y T5 igual a 0.25. En la Tabla B.1 se muestra el grado de paralelismo que correspondería a cada par de tareas adyacentes según este método.

Tabla B.1: Grado de paralelismo calculado a partir de la simulación global.

(T0,T1): 0.75	(T1,T2): 0.5	(T2,T4): 0
(T0,T2): 0.25	(T1,T3): 0.25	(T2,T5): 0.25
(T0,T3): 0.25	(T1,T4): 0	(T3,T4): 0.25
(T0,T4): 0	(T1,T5): 0.75	(T3,T5): 0
(T0,T5): 0.75	(T2,T3): 0.25	(T4,T5): 0

El problema que plantean estos valores de grado de paralelismo, es que no siempre son una cota máxima de la posibilidad de ejecución concurrente que se puede dar entre las tareas para cualquier asignación, tanto si se trata de tareas adyacentes como no adyacentes. Veamos dos ejemplos en los que en función de la asignación se puede explotar un paralelismo mayor que el calculado.

Para la siguiente asignación a dos procesadores P0:(T0,T2,T5) P1:(T1,T3,T4), se obtiene el diagrama de tiempo en la ejecución que se muestra en la Figura B.3.

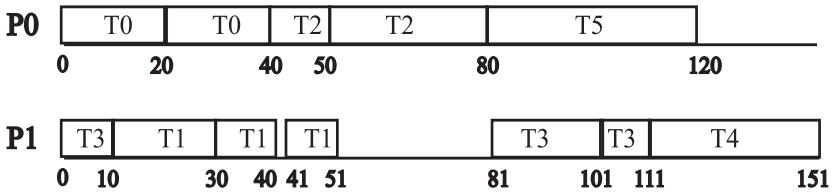

Figura B.3: Simulación de la asignación P0:(T0,T2,T5) P1:(T1,T3,T4).

En el caso de las tareas adyacentes T3 y T5, vemos que se ejecutan concurrentemente durante 30 unidades de tiempo, lo que representa un 75% de su cómputo y es justamente el grado de paralelismo que les corresponde si se calculase a partir del subgrafo aislado de ambas. En cambio en la simulación global se había obtenido un grado de paralelismo igual a 0 (Tabla B.1).

Para tareas no adyacentes ocurre el mismo problema. Considerando por ejemplo la asignación a tres procesadores P0:(T0,T2) P1:(T1,T5) P2:(T3,T4) se obtiene una ejecución como la que se muestra en la Figura B.4. En esta ejecución vemos por ejemplo que las tareas no adyacentes T2 y T5 paralelizan 20 unidades de tiempo, lo que implicaría un posible grado de paralelismo de 0.5 entre estas dos tareas. En cambio el grado global calculado en la Tabla B.1 es únicamente 0.25.

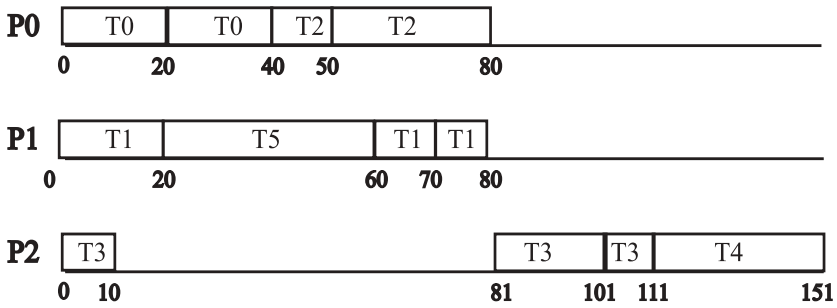

Figura B.4: Simulación de la asignación P0:(T0,T2) P1:(T1,T5) P2:(T3,T4).

Vemos por lo tanto que el mecanismo de adjuntar al modelo un valor de grado de paralelismo basado en la simulación global del grafo es perfectamente viable en cuanto a complejidad, pero no proporciona un valor de grado de paralelismo que sea independiente de la asignación, y por lo tanto no constituye un valor máximo de referencia para utilizar en las decisiones de mapping.