Capítulo 4:
GRAPH TRAVERSE SCHEDULING, Paralelización de Recurrencias

En este capítulo se presentan un conjunto de técnicas que permiten obtener el máximo paralelismo del bucle en presencia o no de recurrencias.

La distribución de operaciones en tareas, que se denomina GTS *(Graph Traverse Scheduling: distribución de operaciones mediante recorridos del grafo), se basa en ejecutar en cada instante de tiempo todas aquellas instanciaciones de sentencias del bucle libres de dependencia.*

En este capítulo se considera en primer lugar GTS para grafos que incluyen una única recurrencia hamiltoniana. En este caso se consigue una distribución de operaciones que genera tareas totalmente independientes y consigue el máximo paralelismo del bucle. A continuación se considera la aplicación de GTS a grafos hamiltonianos que incluyen más de una recurrencia, para los cuales es necesario introducir sincronización explícita entre tareas generadas. La generación de código paralelo para estos casos se describe también en la sección 4.

En la sección 5 se describen las técnicas que permiten realizar modificaciones en el grafo de dependencias previas a la aplicación de GTS. En la última sección de este capítulo se considera la aplicación de GTS a grafos que incluyen dependencias de control debidas a sentencias estructuradas condicionales.

4.1 GRAPH TRAVERSE SCHEDULING

La distribución de operaciones.-.en tareas, que se denomina-.GTS (*Graph Traverse Scheduling:* distribución de operaciones mediante recorridos del grafo), se basa en ejecutar en cada instante de tiempo todas aquellas instanciaciones de sentencias del bucle libres de dependencia. Algunas de estas dependencias quedan aseguradas por la propia secuencialidad de las tareas generadas y el resto por sincronización explícita entre tareas. Para ello, GTS efectúa (a) la reescritura de las expresiones de indexación, de las sentencias que incluyen variables estructuradas tipo vector, en función de las distancias asociadas a dependencias que la propia secuencialidad de las tareas va a asegurar, y (b) la inserción de primitivas de sincronización entre tareas, intentando minimizar su coste.

En principio GTS se desarrolla como una técnica que permite la paralelización de recurrencias (aplicable por ejemplo a π-blocks cíclicos) aunque puede aplicarse a cualquier tipo de grafo. GTS realiza la distribución de operaciones basándose en recorridos del grafo a través de una recurrencia hamiltoniana. Para poder realizar esta distribución, GTS se ve soportado por otras técnicas que permiten realizar modificaciones en el grafo de dependencias con la finalidad de (a) conseguir una recurrencia hamiltoniana, (b) reducir el número de sincronizaciones explícitas a base de reducir el paralelismo del bucle o (c) minimizar el número de procesadores necesarios para conseguir el paralelismo máximo disponible.

4.2 GRAFOS CON UNA ÚNICA RECURRENCIA HAMILTONIANA

El número de tareas que GTS genera queda determinado por el paralelismo del bucle, es decir, por la suma de las distancias asociadas a los arcos que forman la recurrencia hamiltoniana.

El algoritmo utilizado por GTS para asignar sentencias e iteraciones a cada una de las tareas generadas es el siguiente:

(a) Inicialmente se asigna a cada tarea una sentencia e iteración sin dependencias iniciales. Tal como se ha visto en la sección 3.2, el número de iteraciones de una sentencia que pueden ser inicialmente

ejecutadas viene determinado por la distancia asociada al arco que entra en el nodo del grafo que la representa a dicha sentencia.

La distribución de operaciones se realiza recorriendo el grafo de dependencias en sentido contrario al que indica la recurrencia, asignando a tareas consecutivas todas las iteraciones libres de dependencia de cada una de las sentencias. Para una recurrencia general

$$\mathbf{R}=\{\mathbf{d}_{1j}, \mathbf{d}_{jk}, ..., \mathbf{d}_{mn}, \mathbf{d}_{n1}\} \quad | \quad w(\mathbf{R}) = P$$

incluida en un bucle normalizado, GTS genera P tareas, numeradas desde 0 a P-l, asignándoles las siguientes instanciaciones de las sentencias:

$$S_{11}, S_{12}, \dots, S_{1d_{n1}}, S_{n1}, S_{n2}, \dots, S_{nd_{mn}}, \dots, S_{k1}, S_{k2}, \dots, S_{kd_{jk}}, S_{j1}, S_{j2}, \dots, S_{jd_{1j}}$$

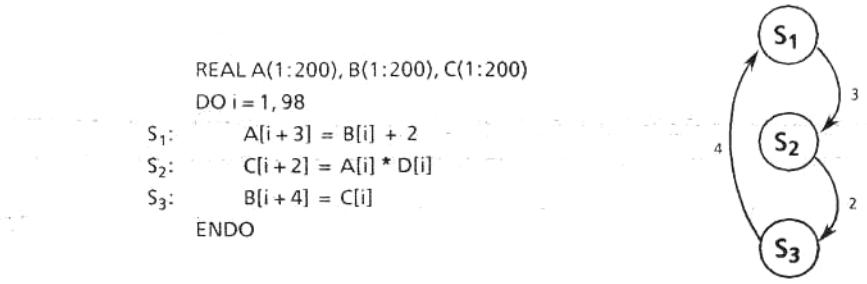
siendo S_{ij} la instanciación de la sentencia S_i en la iteración j.

(b) A partir de la sentencia e iteración inicial asignada a cada tarea, se le asignan todas aquellas instanciaciones de sentencias que forman una cadena de dependencias.

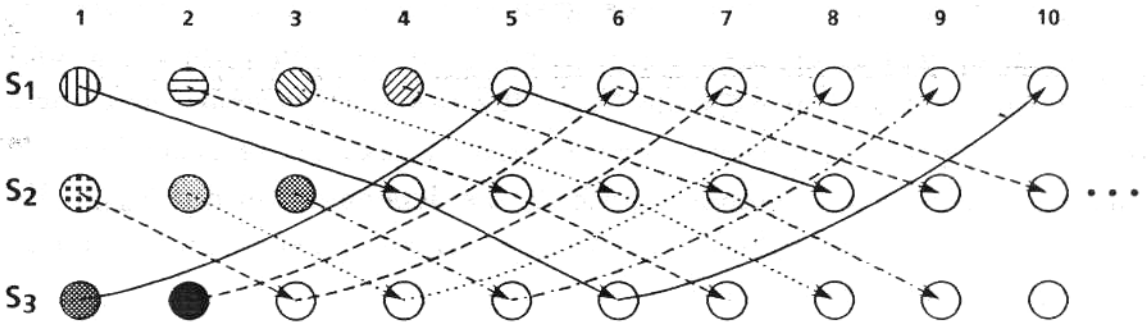
La distribución de operaciones se obtiene recorriendo el grafo a través de la recurrencia hamiltoniana. De esta manera las dependencias de la recurrencia quedan aseguradas por la propia secuencialidad de la tarea.

Observar que la numeración anterior propuesta cumple que si en la tarea *i* se ejecuta la instanciación de la sentencia S_k asociada a la iteración j, entonces en la tarea *((i + 1) mod P)* se ejecuta la instanciación asociada a la iteración *(j + 1)* de la misma sentencia. Esto facilita la generación de las primitivas de sincronización entre tareas cuando existen varias recurrencias en el grafo de dependencias.

La figura 4.1 muestra un bucle normalizado, su grafo de dependencias asociado, parte de su ejecución desarrollada y la distribución de operaciones realizada por GTS. Inicialmente GTS asigna a tareas diferentes todas aquellas instanciaciones de sentencias libres de dependencia y que en la figura 4.1(b) se corresponden con aquellas que no son destino de ninguna flecha (instanciaciones sombreadas). En este caso, GTS genera 9 tareas tal y como muestra la primera fila de la figura 4.1(c). A partir de cada una de ellas, GTS asigna a cada tarea todas las instanciaciones de sentencias que forman una



(a)



(b)

$tarea_0$	$tarea_1$	$tarea_2$	$tarea_3$	$tarea_4$	$tarea_5$	$tarea_6$	$tarea_7$	$tarea_8$
$S_{1\,1}$	$S_{1\,2}$	$S_{1\,3}$	$S_{1\,4}$	$S_{3\,1}$	$S_{3\,2}$	$S_{2\,1}$	$S_{2\,2}$	$S_{2\,3}$
$S_{2\,4}$	$S_{2\,5}$	$S_{2\,6}$	$S_{2\,7}$	$S_{1\,5}$	$S_{1\,6}$	$S_{3\,3}$	$S_{3\,4}$	$S_{3\,5}$
$S_{3\,6}$	$S_{3\,7}$	$S_{3\,8}$	$S_{3\,9}$	$S_{2\,8}$	$S_{2\,9}$	$S_{1\,7}$	$S_{1\,8}$	$S_{1\,9}$
$S_{1\,10}$	$S_{1\,11}$	$S_{1\,12}$	$S_{1\,13}$	$S_{3\,10}$	$S_{3\,11}$	$S_{2\,10}$	$S_{2\,11}$	$S_{2\,12}$
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
$S_{1\,91}$	$S_{1\,92}$	$S_{1\,93}$	$S_{1\,94}$	$S_{3\,91}$	$S_{3\,92}$	$S_{2\,91}$	$S_{2\,92}$	$S_{2\,93}$
$S_{2\,94}$	$S_{2\,95}$	$S_{2\,96}$	$S_{2\,97}$	$S_{1\,95}$	$S_{1\,96}$	$S_{3\,93}$	$S_{3\,94}$	$S_{3\,95}$
$S_{3\,96}$	$S_{3\,97}$	$S_{3\,98}$		$S_{2\,98}$		$S_{1\,97}$	$S_{1\,98}$	

(c)

Figura 4.1: Bucle con grafo de dependencias con una única recurrencia hamiltoniana.

cadena de dependencias. En este caso, todas las cadenas que aparecen en la ejecución desarrollada de la figura 4.1(c) son totalmente independientes, y por lo tanto, pueden ser ejecutadas sin necesidad de sincronización.

Notar que el paralelismo del bucle es constante excepto en el último ciclo, que depende de los límites de iteración.

4.3 GRAFOS HAMILTONIANOS CON VARIAS RECURRENCIAS

En esta sección se presenta la aplicación de GTS a grafos hamiltonianos que incluyen varias recurrencias. En este caso, las tareas generadas pueden no ser totalmente independientes y por lo tanto requerir mecanismos de sincronización que aseguren las dependencias del grafo.

4.3.1 Distribución de operaciones

Si B es el conjunto de recurrencias del grafo G, la distribución de operaciones se realiza aplicando el mismo algoritmo de la sección anterior a una recurrencia hamiltoniana del grafo, $R_{sch} \in B$, que denominaremos recurrencia de "scheduling" o de distribución. De esta manera, la propia secuencialidad de cada tarea garantiza el cumplimiento de las dependencias $d_{ij} \in R_{sch}$. Sin embargo, el resto de dependencias deben garantizarse con algún mecanismo de sincronización entre tareas.

Tal como se ha descrito en la sección anterior, el número de tareas generadas P depende del peso de la recurrencia R_{sch} es decir,

$$P = w(\mathbf{R_{sch}}) = \sum_{\mathbf{d_{ij}} \in \mathbf{R_{sch}}} d_{ij}.$$

Por lo tanto, pueden presentarse las siguientes situaciones:

* $P = ||$, si R_{sch} es una de las recurrencias más restrictivas;
* $P > ||$, si R_{sch} no limita el paralelismo del bucle. En este caso la distribución de operaciones utiliza más procesadores de los necesarios.

Si existen varias recurrencias $R \in B$ hamiltonianas, una de ellas debe de ser elegida de manera que se minimice el número de tareas generadas, es decir,

$$P = w(\mathbf{R_{sch}}) \leq w(\mathbf{R}) \quad \forall\, \mathbf{R_{sch}}, \mathbf{R} \in \mathbf{B} \quad | \quad |\mathbf{R_{sch}}| = |\mathbf{R}| = n.$$

4.3.2 Sincronización

La figura 4.2(b) muestra la distribución de operaciones obtenida por GTS para el bucle y grafo de dependencias asociado de la figura 4.2(a). Para este bucle, GTS genera 8 tareas y el paralelismo evaluado utilizando las expresiones del

apartado 3.1.3 es 6, limitado por la recurrencia $\{d_{23}, d_{32}\}$. Observar que las tareas obtenidas no son totalmente independientes ya que existen dependencias entre ellas.

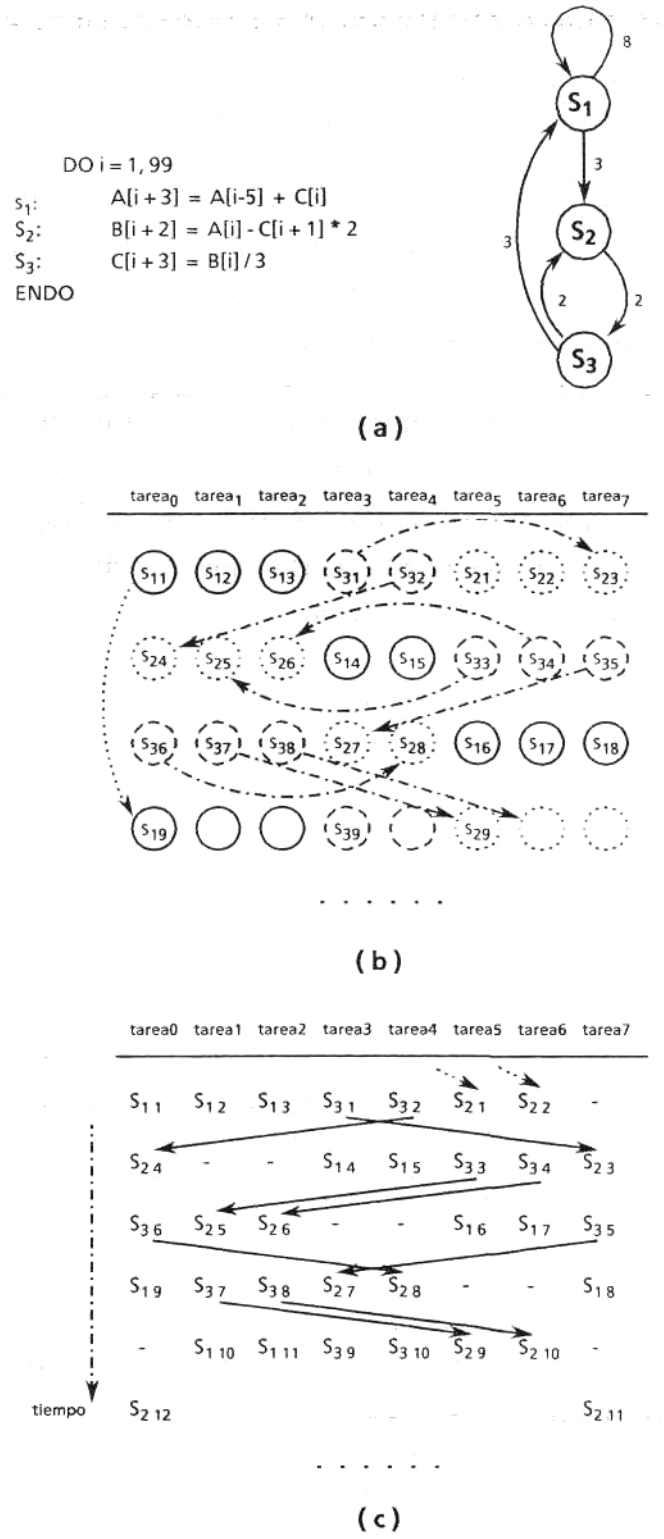


Figura 4.2: Bucle con grafo de dependencias hamiltoniano con varias recurrencias.

Así por ejemplo, en la primera fila de la figura 4.2(b) se puede observar que la instanciación S_{23} en la tarea 7 no puede ejecutarse hasta que haya finalizado la ejecución de la instanciación S_{31} en la tarea 3, debido a la relación de dependencia d_{32} no incluida en la $R_{sch}=\{d_{12},d_{23},d_{31}\}$. Lo mismo ocurre en todas las tareas, con lo cual es necesario utilizar algún mecanismo de sincronización entre tareas que asegure esta relación de dependencia d_{32}.

Observar por otro lado que d_{11} no necesita ser explícitamente sincronizada pues queda incluida en la propia secuencialidad de las tareas. Por lo tanto necesitaremos algún criterio que permita decidir cuando una dependencia debe ser explícitamente sincronizada o no.

La figura 4.2(c) muestra la ejecución sincronizada de las tareas indicando con un guión aquellos ciclos en los que la ejecución de una tarea debe ser retardada con la finalidad de no violar la relación de dependencia d_{32}. Observar que en cada ciclo se ejecutan 6 instanciaciones distintas, obteniendo por lo tanto un paralelismo igual al evaluado.

A continuación se describe el algoritmo utilizado por GTS para insertar las primitivas de sincronización adecuadas que aseguren el cumplimiento de aquellas relaciones de dependencia no incluidas en la R_{sch}, y por lo tanto, no aseguradas por la propia secuencialidad de las tareas generadas. Posteriormente se presenta la condición que decide si un arco del grafo necesita o no sincronización explícita para ser asegurado.

Algoritmo

En general, a fin de asegurar el cumplimiento de las relaciones de dependencia $d_{ij} \notin R_{sch}$, la sentencia Si fuente de la dependencia debe indicar que ha finalizado su ejecución para que así la sentencia S_j destino de la dependencia pueda iniciar su ejecución.

El algoritmo utilizado por GTS para asegurar estas relaciones de dependencia, teniendo en cuenta que van a utilizarse semáforos como primitivas de sincronización entre tareas es el siguiente:

(a) Para cada arco $d_{ij} \notin R_{sch}$, se necesitan P semáforos

$$sem_{ij}(t) \qquad 0 \le t < P.$$

(b) La primitiva signal sobre el semáforo $sem_{ij}(i)$ se inserta después de la sentencia fuente de la dependencia Si en la tarea *t*, mientras que la primitiva wait se inserta antes de la sentencia destino de la dependencia Sj en la tarea

$$(t + d_{ij} - w(\mathbf{C_{ij}})) \bmod P ; \qquad \mathbf{C_{ij}} \subset \mathbf{R_{sch}} .$$

(c) Los P semáforos necesarios para sincronizar cada arco $d_{ij} \notin R_{sch}$ se deben inicializar a:

$$sem_{ij}(t) = \begin{cases} \lfloor d_{ij}/P \rfloor + 1 & \text{si } (t + d_{ij} - w(\mathbf{C_{i1}})) \bmod P < d_{ij} \bmod P \\ \lfloor d_{ij}/P \rfloor & \text{si } (t + d_{ij} - w(\mathbf{C_{i1}})) \bmod P \geq d_{ij} \bmod P \end{cases}$$

dado que inicialmente algunas iteraciones de la sentencia destino de la dependencia S_j pueden ser ejecutadas sin violar la relación de dependencia d_{ij}.

La justificación del algoritmo presentado se realiza a continuación. En primer lugar se tiene que, el número de semáforos requerido es P, dado que cada par de tareas ejecutan las primitivas de sincronización complementarias sobre un objeto de sincronización distinto.

Por otro lado se obtiene la relación entre tareas que ejecutan las primitivas de sincronización asociadas a un determinado objeto de sincronización. Si la iteración *k* de la sentencia fuente S_i se ejecuta en la tarea *t*, la iteración de la sentencia S_j destino de la dependencia que va a ser ejecutada en la misma tarea será

$$k + w(\mathbf{C_{ij}}) \qquad \mathbf{C_{ij}} \subset \mathbf{R_{sch}}$$

La numeración de tareas propuesta en la sección anterior cumple que tareas consecutivas ejecutan iteraciones consecutivas de una determinada sentencia, y por lo tanto, la iteración *(k + dij)* de la sentencia destino S_j se ejecutará en la tarea

$$t' = (t + d_{ij} - w(\mathbf{C_{ij}})) \bmod P ; \qquad \mathbf{C_{ij}} \subset \mathbf{R_{sch}}$$

La relación inversa es fácil de obtener a partir de la expresión anterior. Si *t'* es la tarea que ejecuta la primitiva **wait**, la primitiva **signal** asociada se ejecutará en la tarea *t* sobre el semáforo $sem_{ij}(t)$ dado por:

$$t = (t' + w(\mathbf{C_{ij}}) - d_{ij}) \bmod P \qquad \mathbf{C_{ij}} \subset \mathbf{R_{sch}}$$

Por otro lado se obtiene la expresión que determina la **inicialización de los elementos de sincronización** utilizados. Cualquier arco d_{ij} permite la ejecución de las primeras d_{ij} iteraciones de la sentencia destino de la dependencia. Dado que tareas consecutivas ejecutan iteraciones consecutivas de una misma sentencia y que la tarea que ejecuta la primera iteración de la sentencia destino de la dependencia S_j viene dada, según la distribución de tareas presentada, por $w(C_{j1})$, se tiene que las tareas que van a ejecutar estas iteraciones son:

$$(w(\mathbf{C_{j1}}) + k) \bmod P \quad | \quad 0 \leq k < d_{ij}$$

y por lo tanto las tareas que deben ejecutar la primitiva signal asociada son:

$$t = (w(\mathbf{C_{j1}}) + k + w(\mathbf{C_{ij}}) - d_{ij}) \bmod P \quad | \quad 0 \leq k < d_{ij}\,.$$

Dada una recurrencia R_{sch} se cumple que:

$$w(\mathbf{C_{j1}}) + w(\mathbf{C_{ij}}) = P + w(\mathbf{C_{i1}})$$

y por lo tanto:

$$t = (w(\mathbf{C_{i1}}) + k - d_{ij}) \bmod P \quad | \quad 0 \leq k < d_{ij}$$

que también puede escribirse como:

$$(t + d_{ij} - w(\mathbf{C_{i1}})) \bmod P = k \quad | \quad 0 \leq k < d_{ij}$$

Si dij $\leqq$ P, los semáforos $sem_{ij}(t)$ con t cumpliendo

$$(t + d_{ij} - w(\mathbf{C_{i1}})) \bmod P < d_{ij}$$

deberán ser inicializados a 1 mientras que el resto lo deberán ser a 0.

Si d_{ij}>P, todos los semáforos $sem_{jj}(t)$ se inicializarán a $\lfloor d_{ij} / P \rfloor$ y sólo aquellos que cumplan

$$(t - w(\mathbf{C_{i1}}) + d_{ij}) \bmod P < d_{ij} \bmod P$$

se inicializarán en una unidad más.

La figura 4.2(c) muestra la distribución de operaciones y necesidad de sincronización de la relación de dependencia d_{32} para el bucle asociado. Notar que las tareas 5 y 6 ejecutan dos primeras iteraciones de la sentencia destino S_3 y que pueden ser inicialmente ejecutadas dado que no dependen de ninguna ejecución previa. Por lo tanto, los semáforos asociados $sem_{32}(1)$ y $sem_{32}(2)$ deben ser inicializados a 1 y son los que cumplen la condición anterior.

Reducción del número de sincronizaciones

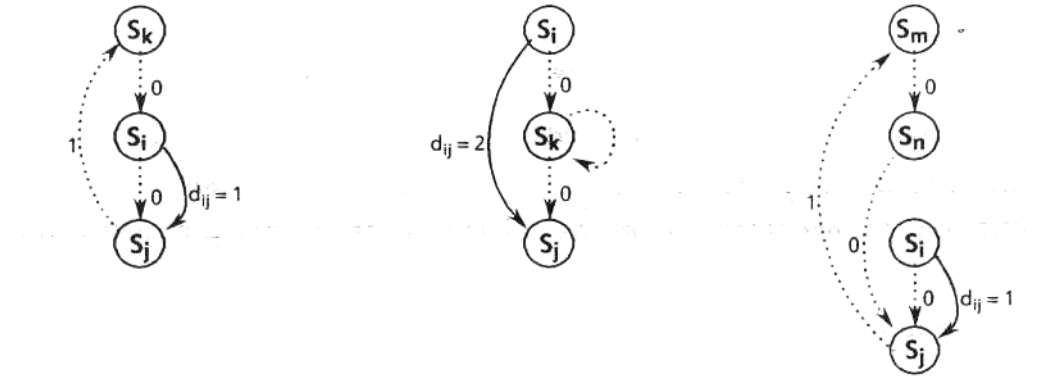
A continuación se presenta el criterio que permite al compilador decidir si una relación de dependencia dijnecesita sincronización explícita o no, es decir, queda incluida en la propia secuencialidad de las tareas oes sincronizada por otras sincronizaciones ya existentes.

Un arco $d_{ij} \notin R_{sch}$ no necesita ser explícitamente sincronizado si existe una cadena de arcos del grafo (pasando incluso varias veces por un mismo arco) que cumple la siguiente condición:

$$(d_{ij} - (w(C_{ij}) + \sum_{S_k \in C^*_{ij}} m \cdot w(C_{kk}))) \bmod P = 0 \quad | \quad d_{ij} \geq w(C_{ij}) \wedge m \in \mathbb{N}$$

Este es el caso de la relación de dependencia d_{11} en el ejemplo de la figura 4.2. La propia secuencialidad de la tarea asegura el cumplimiento de esta dependencia.

Así por ejemplo, los siguientes arcos d_{ij} (marcados con trazo continuo) pertenecientes a fragmentos de grafos de dependencias no requieren ser explícitamente sincronizados, dada la existencia de las cadenas indicadas:



Este proceso de reducción de arcos puede ser realizado antes de seleccionar la recurrencia R_{sch}, dejando así el grafo con el mínimo número de arcos posibles.

4.4 GENERACIÓN DE CÓDIGO PARALELO

Tal como se ha descrito en los apartados anteriores, GTS obtiene tantas tareas como indica el peso de la recurrencia R_{sch}: Estas tareas pueden ser independientes y por lo tanto ejecutarse como bucle DOALL o dependientes en cuyo caso ejecutarse como bucle DOACROSS. En ambos casos, la variable de control de estos bucles paralelos indica el número de tarea.

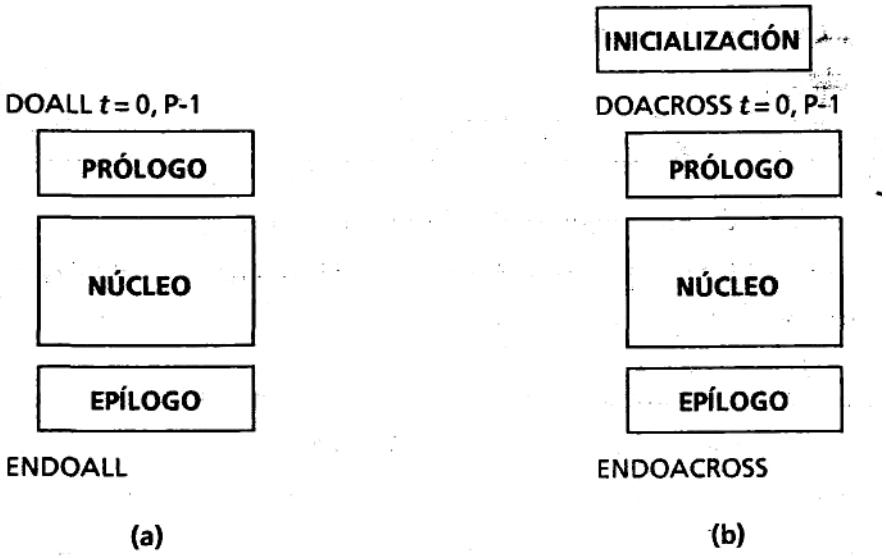


Figura 4.3: (a) Estructura general del código generado por GTS para tareas independientes o (b) sincronizadas.

En el proceso de generación de código paralelo se pueden distinguir dos fases: (a) reescritura del bucle e (b) inserción de instrucciones de sincronización. La estructura general del código generado se muestra en la figura 4.3. La estructura propuesta pretende obtener un código único para todas las tareas generadas.

4.4.1 Reescritura del bucle secuencial

La distribución de operaciones realizada por GTS asigna a una misma tarea todas las sentencias del bucle original empezando por una de las que inicialmente pueden ejecutarse y siguiendo el orden indicado por la recurrencia R_{sch} elegida.

Las expresiones de indexación de los vectores que aparecen en cada sentencia se ven afectadas por el peso, a través de la R_{sch}, desde la sentencia inicial asignada a la tarea hasta la sentencia considerada.

Prólogo

El primer problema que se plantea es que las tareas generadas pueden tener asignada una sentencia inicial distinta, que depende, según el algoritmo descrito en la sección 4.1, del número de tarea. Si se elige como sentencia inicial del núcleo la sentencia S_i, se tiene que cada tarea deberá ejecutar en el prólogo las iteraciones asignadas de las sentencias comprendidas entre la inicial de la tarea y S_i a través de R_{sch}-

Por ejemplo, para la distribución de operaciones realizada por GTS mostrada en la figura 4.4 para el bucle de la figura 4.1, si se elige como sentencia inicial del núcleo la S_i, se tiene que todas aquellas instanciaciones de sentencias por encima de la línea a puntos deberán ser ejecutadas en el prólogo.

La figura 4.5(b) muestra la estructura general del prólogo para un bucle general (figura 4.5(a)) con una recurrencia R_{sch} tal que $S_i \delta S_{i+1}$ ($1 \leq i \leq n-1$) y $S_n \delta S_i$. Por simplicidad en la escritura se denota $w_{ij} = w(C_{ij})$. Por otro lado el segundo subíndice *f* de S_{if} indica por lo que debe de sustituirse la variable de control que aparece en la función de indexación original. Notar que cada sentencia condicional decide, en función del número de tarea *t* (índice del bucle DOALL o DOACROSS), si una determinada sentencia debe ser o no ejecutada.

Núcleo

El núcleo ejecuta de forma secuencial todas aquellas iteraciones y sentencias asignadas a la tarea a partir de una determinada sentencia e iteración inicial.

En la distribución de operaciones de la figura 4.1 se puede observar que la iteración inicial depende del número de tarea y que el número de iteraciones que separan la ejecución de una misma sentencia es múltiplo del número total de tareas generadas P.

En la figura 4.5(c) se muestra la estructura general para el núcleo. Notar que la iteración final del bucle generado no es N, dado que determinadas tareas no ejecutan completamente la última iteración del bucle (iteraciones encuadradas a puntos en la distribución de operaciones de la figura 4.4 (b)).

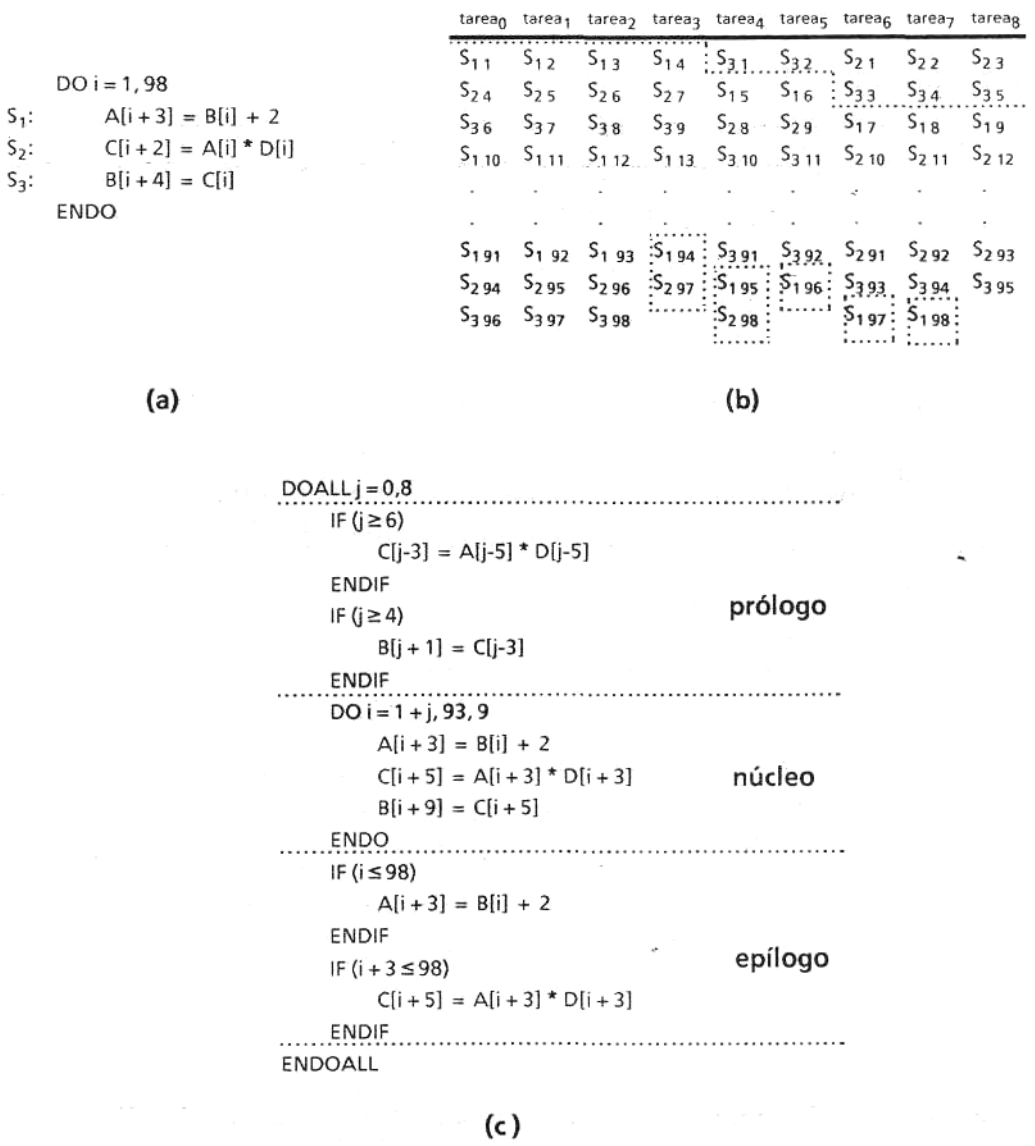


Figura 4.4: Código paralelo generado para el bucle secuencial de la figura 4.1.

Epílogo

El epílogo ejecuta el resto de iteraciones y sentencias que no pueden ser ejecutadas en el núcleo. A partir de la última iteración *i* que ha provocado la salida del núcleo y siguiendo la recurrencia R_{sch} se van ejecutando aquellas sentencias S_j que cumplen que *(i + w_{Ij} i≤N).* La figura 4.5(d) muestra la estructura general para el epílogo. Notar que sólo depende del número de iteraciones N del bucle original.

La figura 4.4(c) muestra el código paralelo obtenido para el bucle secuencial de la figura 4.4(a).

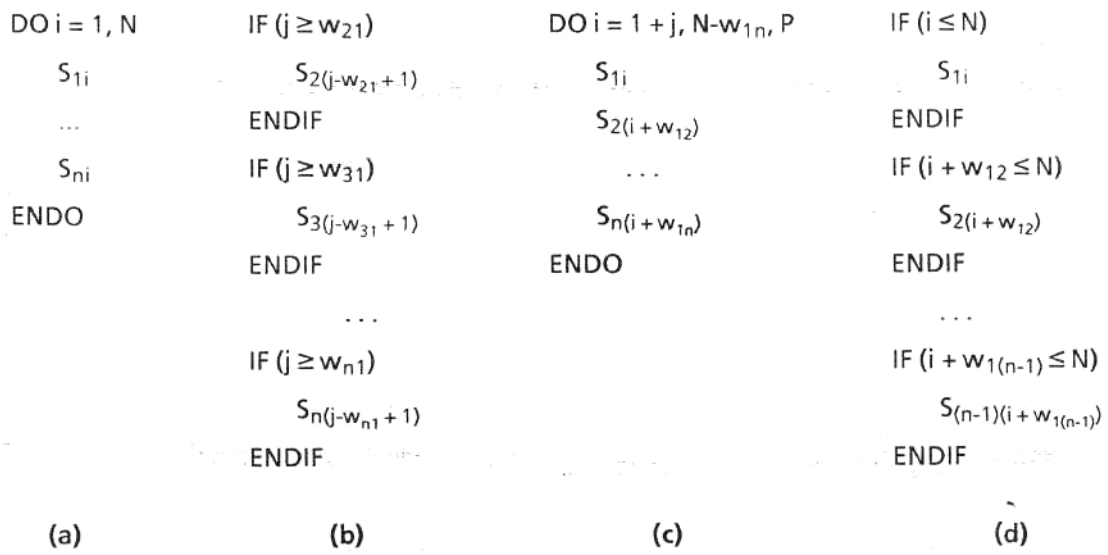


Figura 4.5: (a) Bucle general, (b) esquema general para el prólogo, (c) núcleo y (d) epílogo.

4.4.2 Instrucciones de sincronización

Tal como se ha descrito en la sección 4.3, las dependencias no incluidas en la recurrencia R_{sch} deben de ser aseguradas mediante algún mecanismo de sincronización entre tareas. Aunque otros de los mecanismos descritos en el apartado 1.3.1 pueden ser utilizados con GTS, en este trabajo se utilizan semáforos como primitivas de sincronización.

La generación de estas instrucciones es una implementación directa del algoritmo presentado en la sección 4.3. El compilador inserta una primitiva **signal** después de la sentencia fuente de la dependencia y una primitiva **wait** antes de la sentencia destino de la misma dependencia. Para cada relación de dependencia a sincronizar se requieren P semáforos con la inicialización descrita. Estas primitivas se insertan en el prólogo, núcleo y epílogo. La inicialización de los semáforos se realiza antes de la ejecución del bucle DOACROSS siguiendo la estructura general del código indicada en la figura 4.3. La figura 4.6 muestra el código paralelo generado para el bucle secuencial de la figura 4.2.

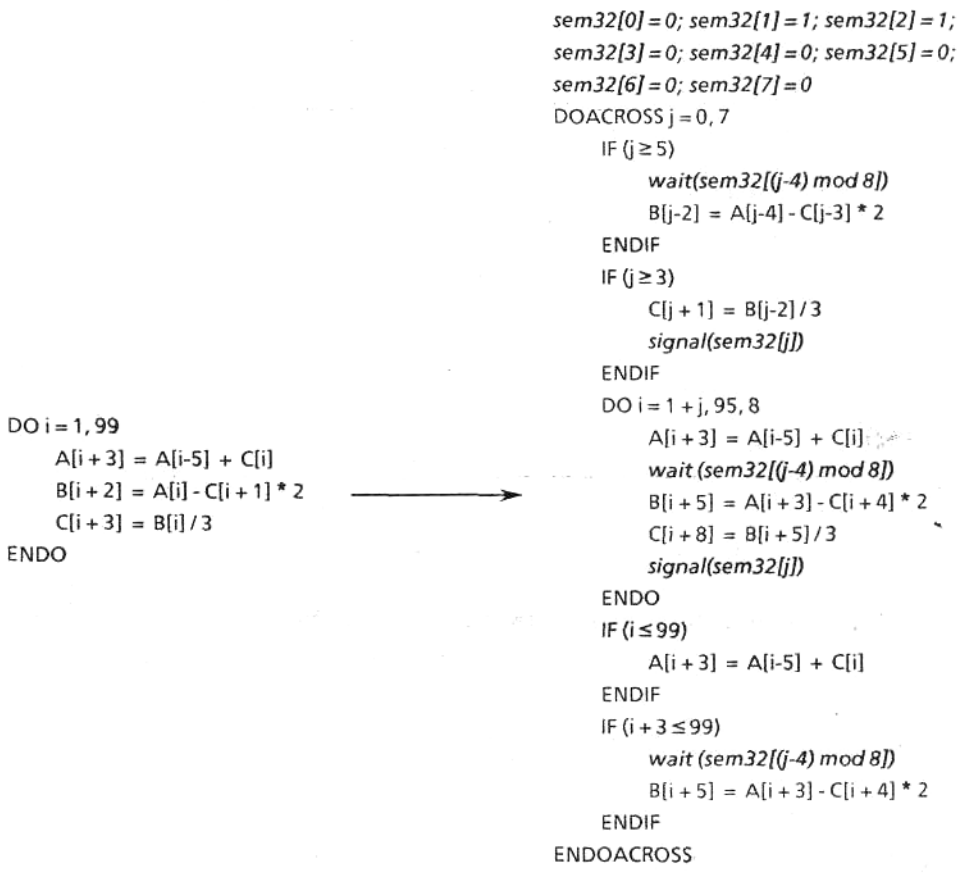


Figura 4.6: Código generado por GTS para el bucle secuencial de la figura 4.2.

4.5 TRANSFORMACIONES DEL GRAFO DE DEPENDENCIAS

En esta sección se presentan las transformaciones previas a la aplicación de GTS que pueden realizarse en el grafo de dependencias con la finalidad de:

(a) obtener una R_{sch} en el caso de grafos de dependencias no hamiltonianos.

(b) reducir el número de procesadores a fin de obtener una eficiencia de paralelización dada o adaptarse al número de procesadores disponible.

(c) reducir el número de sincronizaciones a base de reducir el paralelismo obtenido. También se considera la posibilidad de obtener tareas totalmente independientes.

Todas las transformaciones se basan en añadir *relaciones de dependencia ficticias* en el grafo de manera que se cumplan unas determinadas condiciones según la transformación requerida. Estas transformaciones se realizan en principio mediante búsqueda exhaustiva de todas las soluciones posibles. Este proceso es costoso aunque en la práctica el número de nodos del grafo es pequeño. En los apartados siguientes se presentan las condiciones que en cada caso permiten acotar el espacio de búsqueda. La obtención de heurísticas que reduzcan el tiempo empleado para realizar estas transformaciones se deja como línea abierta de este trabajo.

4.5.1 Obtención de una recurrencia Rsch

En caso de no existir una recurrencia que sea ciclo hamiltoniano del grafo, sobre la cual poder aplicar el algoritmo descrito en la sección 4.2, es posible añadir relaciones de dependencia ficticias.

Definimos E' como un conjunto de arcos d_{ij}' tales que E + E' permite encontrar una recurrencia R_{sch}. La distancia asociada a los arcos de E' debe ser tal que no se limite el paralelismo del bucle, es decir,

$$\|_s(R_1) \geq \|_s(R_2)\ ,\ \forall R_1 \in B' - B\ ,\ R_2 \in B$$

siendo B' el conjunto de recurrencias de G' (V, E + E'). Siguiendo el mismo algoritmo descrito en la sección 4.3, los arcos de E no incluidos en la R_{sch} deben ser explícitamente sincronizados.

. Si existen varios conjuntos E' que permitan obtener una R_{sch}, se eligen aquellos que minimicen el número de tareas ($P = w(R_{sch}) \geqq ||$) y de entre ellos los que introduzcan el menor número de arcos adicionales a fin de reducir el coste de sincronización, dado que se utilizan más arcos originales del conjunto E.

La figura 4.7(a) muestra un grafo de dependencias que no posee una recurrencia hamiltoniana que pueda ser elegida como R_{sch}. Observar que el paralelismo del bucle asociado es 4,5 limitado por la recurrencia $\{d_{12}, d_{21}\}$. Por lo tanto, cualquier arco adicional que genere una hueva recurrencia debe de tener asociada una distancia que no limite el paralelismo anterior. La figura 4.7(b) muestra una posible solución en la que se obtiene una $R_{sch} = \{d_{12}, d_{23}', d_{31}\}$. En este caso la distancia asociada a d_{23}' se ha calculado de manera que la dependencia original d_{13} quede asegurada por el camino $Ci3 \subset R_{sch}$. La

distancia asociada a d_{31} se ha calculado de manera que el número de tareas generadas sea mínimo, es decir, 5 en este caso para poder conseguir el paralelismo de 4,5. Por lo tanto, sólo se requiere sincronizar la dependencia original d_{21} de forma explícita.

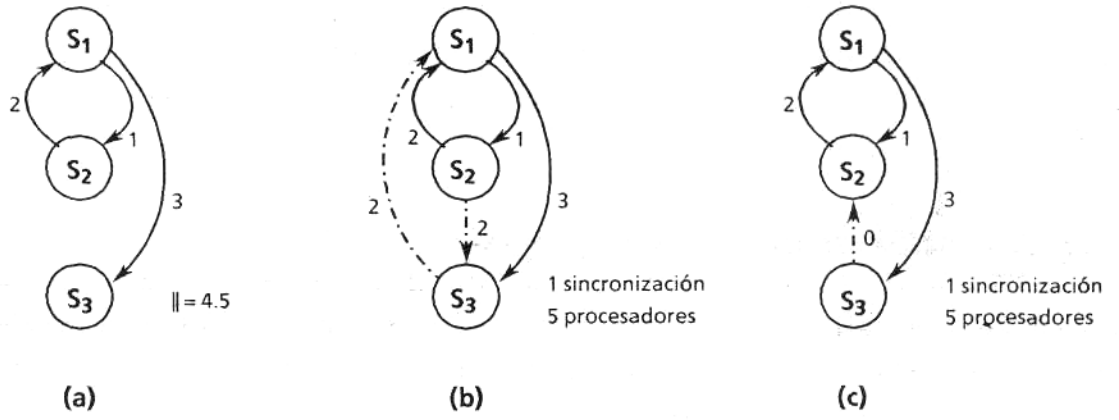


Figura 4.7: Grafo sin recurrencia hamiltoniana y dos posibles soluciones.

La figura 4.7(c) muestra otra posible solución en la cual se ha intentado minimizar el número de arcos adicionales para obtener una R_{sch}. En este caso se obtiene una $R_{sch}=\{d_{13},d_{32}',d_{21}\}$. En esta solución no es posible asegurar d_{12} por el camino $C_{12}\subset R_{sch}$. La distancia asociada a d_{32} se ha calculado de manera que el número de tareas generadas sea también mínimo. Por lo tanto, en este caso, ambas soluciones son equivalentes y requieren una sincronización explícita.

4.5.2 Reducción del número de tareas

Tal y como se ha descrito en la sección 4.2, el número de tareas viene determinado por el peso de la recurrencia R_{sch} elegida, de manera que, si ésta no limita el paralelismo del bucle, se tiene que $P>||$ y por lo tanto, se generan más tareas de las estrictamente necesarias para conseguir el paralelismo del bucle.

Se puede reducir el número de tareas añadiendo arcos adicionales d_{ij}' tales que no limiten el paralelismo del bucle y que además $d_{ij}\in R_{sch}$. Esta reducción del número de procesadores implica sincronizar de forma explícita los arcos $d_{ij} \in R_{sch}$ para los cuales se ha añadido un arco adicional d_{ij}'.

Así por ejemplo, para el grafo de dependencias de la figura 4.8(a) se tiene un paralelismo de 6 aunque a base de generar 8 tareas y una sincronización explícita. La figura 4.8(b) muestra el grafo de dependencias modificado en el que la nueva recurrencia de distribución es $R_{sch} = \{d_{12},d_{23},d_{31}\}$. Observar que el nuevo arco d_{31}' con distancia asociada 1 no limita el paralelismo del bucle y permite reducir el número de tareas generadas a 6. Sin embargo, el arco d_{31} original requiere ser ahora explícitamente sincronizado.

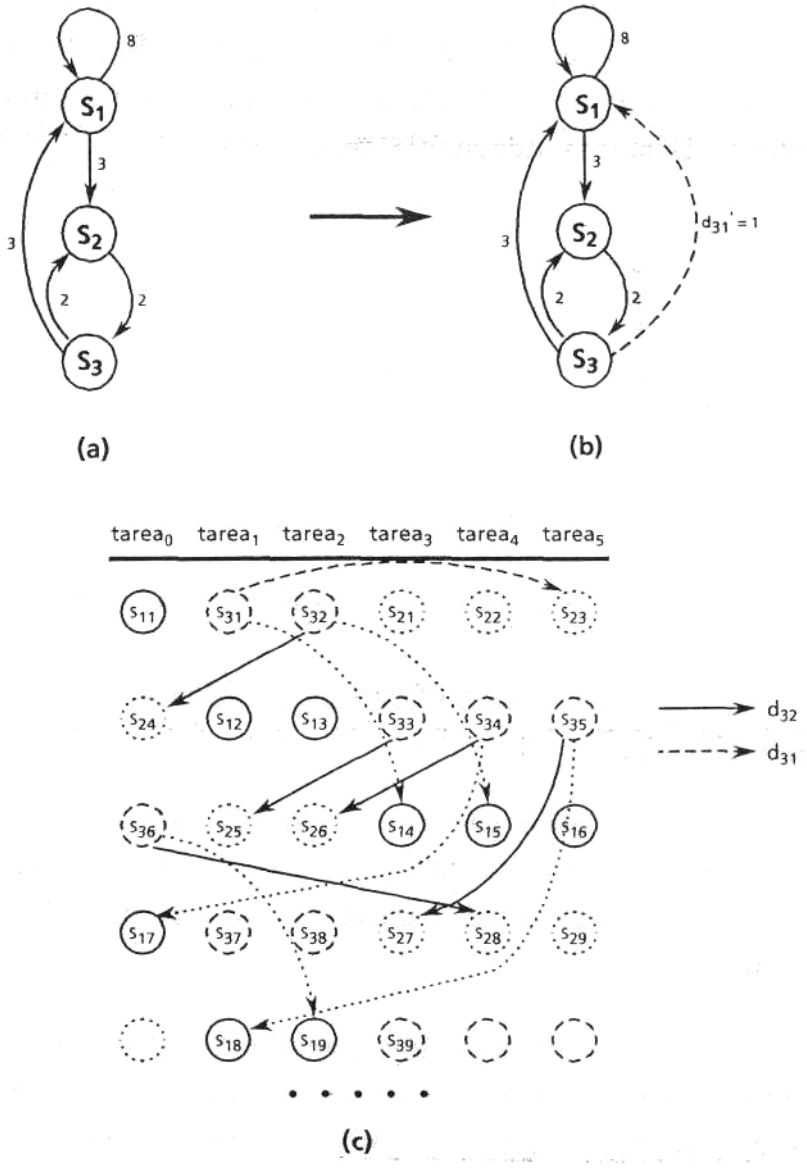


Figura 4.8: Reducción del número de tareas generadas a base de añadir nuevos arcos.

Observar en la distribución de tareas mostrada en la figura 4.8(c), obtenida a partir del grafo modificado, que el arco d_{11} tampoco requiere ser

explícitamente sincronizado pues lo es de forma indirecta por la propia secuencialidad de la tarea y por la sincronización explícita de d_{31}.

Esta transformación también puede aplicarse con la finalidad de conseguir una distribución de operaciones óptima cuando se tiene una limitación en el número de procesadores disponible.

Por ejemplo, para el grafo de dependencias acíclico de la figura 4.9 se tiene una paralelismo de N, siendo N el número de iteraciones a ejecutar del bucle asociado. Si sólo se dispone de 8 procesadores, es posible aplicar esta transformación a fin de conseguir una R_{sch} con peso $w(R_{sch}) = 87$ Para ello se puede añadir un arco d_{31}' con distancia asociada 6, de manera que GTS obtenga 8 tareas totalmente independientes.

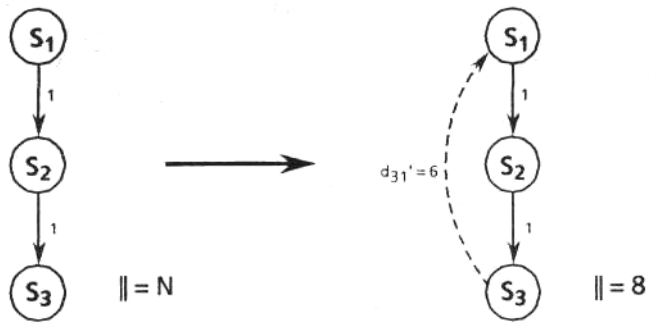


Figura 4.9: Limitación del número de tareas a generar.

4.5.3 Paralelismo sin sincronización y reducción del número de sincronizaciones

Se define *paralelismo libre de sincronización* ||f como el número de procesadores activos que ejecutan de forma independiente diferentes instanciaciones de sentencias del bucle, siguiendo una distribución de operaciones óptima.

La distribución de operaciones para bucles con una única recurrencia hamiltoniana descrita en la sección 4.1 siempre obtiene este paralelismo. Para los otros grafos, este paralelismo ||f es siempre menor o igual al paralelismo evaluado ||.

La transformación consiste en encontrar un conjunto de arcos $E' = \{ d_{ij}' \mid S_i, S_j \in V\}$ tal que exista una R_{sch} y ninguno de los arcos de E necesite ser explícitamente sincronizado.

Para cada posible R_{sch} que pueda generarse, y con la finalidad de acotar el espacio de búsqueda de soluciones válidas, se evalúan las distancias $d_{ij}' \in E'$ de manera que se cumplan las tres condiciones siguientes:

(a) todas las distancias asociadas a los arcos de E' deben ser positivas, es decir,

$$d_{ij}' \geq 0 \qquad \forall\, \mathbf{d_{ij}'} \in \mathbf{E'}.$$

(b) inicialmente sólo pueden ejecutarse tantas iteraciones de una determinada sentencia S_i como indique la mínima de las distancias que entran al nodo que la representa, es decir,

$$d_{ij}' \leq \min_{d_{kj} \in E}(d_{kj}) \qquad \forall\, \mathbf{d_{ij}'} \in \mathbf{E'}.$$

De esta manera se asegura que inicialmente se intente ejecutar alguna instanciación de S_j dependiente de otra ejecución previa.

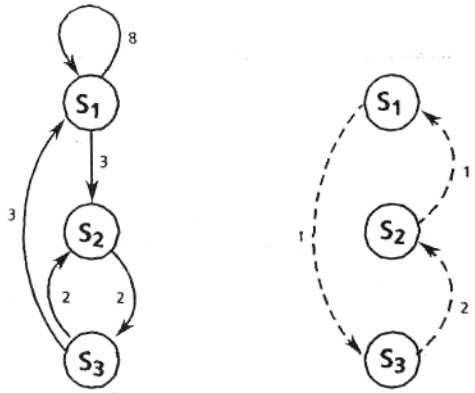
(c) la condición de no sincronización de la sección 4.2 debe de cumplirse para todos los arcos $d_{ij} \in E$, es decir,

$$(d_{ij} - w(\mathbf{C_{ij}})) \bmod w(\mathbf{R_{sch}}) = 0, \qquad \forall\, \mathbf{d_{ij}} \in \mathbf{E}.$$

Siempre existen soluciones triviales en las que todas las $d_{ij}' = 0$ excepto una de ellas que vale 1, y que equivalen a la ejecución secuencial del bucle. Sin embargo, a veces es posible encontrar una solución tal que $\|f = w(R_{sch}) > 1$, en cuyo caso, la eficiencia del proceso de paralelización será

$$\eta = \|_f / \|.$$

Esta transformación también puede ser utilizada para obtener el paralelismo que se consigue sincronizando un determinado número de arcos. En este caso la condición (b) no se debe de imponer. Esto nos permite establecer un compromiso entre número de arcos a sincronizar y paralelismo que se obtiene con la distribución de operaciones propuesta. Para el bucle de la figura 4.2 es posible obtener una solución que, en vez de generar 8 tareas para conseguir un paralelismo $\| = 6$ sincronizando el arco d_{32}, obtiene paralelismo $\|f=4$ sin necesidad de sincronización y generando sólo 4 tareas. La figura 4.10 muestra el grafo de dependencias original y transformado para ese bucle, junto con el nuevo código paralelo generado. Esta transformación es evaluada en el capítulo 6.




```
DOALL j = 0, 3
    IF (j ≥ 3)
        C[j + 1] = B[j-2] / 3
    ENDIF
    IF (j ≥ 1)
        B[j-2] = A[j-4] - C[j-3] * 2
    ENDIF
    DO i = 1 + j, 96, 4
        A[i + 3] = A[i-5] + C[i]
        C[i + 4] = B[i + 1] / 3
        B[i + 5] = A[i + 3] - C[i + 4] * 2
    ENDO
    IF (i ≤ 99)
        A[i + 3] = A[i-5] + C[i]
    ENDIF
    IF (i + 1 ≤ 99)
        C[i + 4] = B[i + 1] / 3
    ENDIF
ENDOALL
```

(a) (b) (c)

Figura 4.10: (a) Grafo original, (b) versión modificada y (c) código paralelo generado sin sincronización.

4.6 SENTENCIAS CONDICIONALES

La existencia de sentencias estructuradas condicionales en el bucle equivale a que existan varias ramas de ejecución posibles. La distribución de operaciones realizada por GTS para este tipo de bucles es la descrita en las secciones anteriores pero con algunas consideraciones previas al buscar la recurrencia R_{sch}. Con la finalidad de evitar bloqueos (deadlocks), se debe operar en todas las posibles ramas de ejecución sobre cada elemento de sincronización.

4.6.1 Distribución de operaciones

A continuación se describe como realizar la elección de la recurrencia R_{sch} y la posterior distribución de operaciones, para el caso de bucles con sentencias estructuradas condicionales. La distribución de operaciones presentada consigue el máximo paralelismo que puede conseguirse en cada instante de tiempo sea cual sea la secuencia de ejecución de las posibles ramas.

Elección de R_{sch}

En principio se tienen tantas R_{sch} alternativas como ramas de ejecución posibles. La elección de una de ellas puede basarse en:

(a) las probabilidades de ejecución de cada rama (por ejemplo, elegir Rsch atravesando los nodos pertenecientes a la rama más probable).

(b) en el paralelismo de las mismas (por ejemplo, elegir R_{sch} atravesando los nodos pertenecientes a la rama de ejecución con más paralelismo).

(c) según la probabilidad estimada y basándose en las cotas probabilísticas estudiadas en el apartado 3.4.2.

En cualquier caso puede ser necesario añadir relaciones de dependencia ficticias con la finalidad de conseguir dicha R_{sch}, tal como se ha descrito en el apartado 4.4.1.

Modificaciones en el grafo

En una determinada tarea debe ejecutarse la misma iteración, tanto de la sentencia que evalúa la condición, como de las sentencias incluidas en el cuerpo de la sentencia estructurada condicional, dado que existe* una dependencia de control entre ellas.

La aplicación del algoritmo de distribución de operaciones requiere por lo tanto garantizar que cualquier camino $C_{ij} \subset R_{sch}$ entre dos sentencias S_i y S_j incluidas en la sentencia estructurada condicional cumpla que $w(C_{ij})=0$. Si no es así, se deberá añadir arcos de dependencia ficticios d_{lm} con distancia asociada 0 entre sentencias S_i y S_m consecutivas en la R_{sch} de una misma rama. Estos arcos d_{lm} no limitan el paralelismo de la rama de ejecución pero obligan a sincronizar los arcos d_{lm} del grafo original.

La elección de R_{sch} siguiendo una determinada rama de ejecución determina qué rama va a tener las relaciones de dependencia incluidas en la secuencialidad de las tareas generadas. Sin embargo, GTS considera toda la sentencia estructurada condicional como un solo nodo del grafo de manera que cada procesador la ejecuta completamente para una determinada iteración del bucle. Por lo tanto, también se añaden arcos d_{lm}' con distancia 0 entre

sentencias no incluidas en la R_{sch} para las cuales exista alguna dependencia original d_{lm}.

La figura 4.11(a) muestra el grafo de dependencias correspondiente a un bucle que incluye una sentencia condicional IF-THEN-ELSE. Para cada una de las posibles ramas de ejecución, las figuras 4.1 l(b) y 4.1 l(c) muestran los subgrafos de dependencias asociados y paralelismo evaluado. Si se elige como R_{sch} la rama que más paralelismo posee se debe de añadir un arco ficticio d_{31}' con distancia asociada 2 a fin de conseguir dicha recurrencia R_{sch}, tal como muestra la figura 4.11(d). La distribución de operaciones obtenida por GTS se muestra en la figura 4.11(e).

4.6.2 Sincronización

En este apartado se presenta el algoritmo que permite insertar las primitivas de sincronización asociadas a todas aquellas relaciones de dependencia no incluidas en la R_{sch} que necesiten ser explícitamente sincronizadas. Las relaciones de dependencia que puedan existir entre sentencias incluidas en la sentencia estructurada condicional con distancia asociada 0 y aquellas que cumplan la condición de no sincronización explícita descrita en el apartado 4.2.2 no necesitan ser explícitamente sincronizadas.

Con la finalidad de evitar bloqueos, cualquier relación de dependencia debe cumplirse si se ejecuta la rama que incluye las sentencias que la generan o darla por cumplida en el caso de ejecutar cualquier otra rama, para una iteración del bucle concreta. Esto implica que se debe de operar sobre el elemento de sincronización asociado en todas las posibles ramas de ejecución.

Primitiva Signal

En general, para cualquier dependencia d_{ij} a sincronizar debe realizarse:

(a) signal al finalizar la ejecución de Si al ejecutar cualquier rama V_m que la contenga, es decir,

$$\forall V_m \mid S_i \in V_m ;$$

(b) signal al ejecutar cualquier rama V_n que no contenga a S_i, es decir,

$$\forall V_n \mid S_i \notin V_n ,$$

antes de la ejecución de la primera sentencia en orden léxico $S_k \in V_n$ que cumpla

$$\exists \mathbf{S_c} \in (\mathbf{V_m} \cap \mathbf{V_n}) \mid (S_c \, \delta^c \, S_k \wedge S_c \, \delta^c \, S_i \wedge \mathbf{S_k} \notin \mathbf{V_m}) .$$

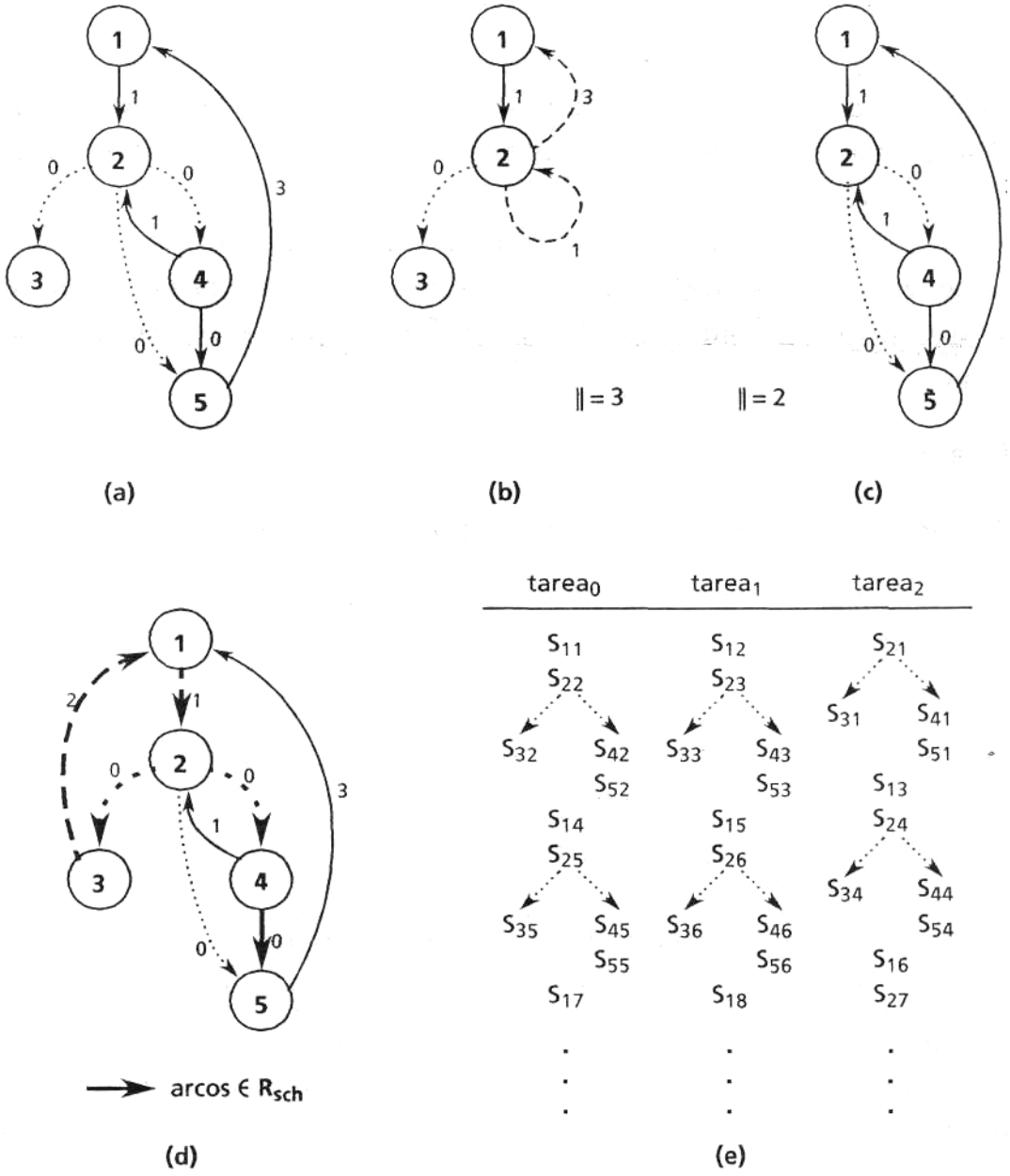


Figura 4.11: Grafo de dependencias con arcos condicionales, subgrafos asociados a las ramas de ejecución y distribución de operaciones obtenida por GTS.

Por ejemplo, en el grafo de la figura 4.12 en el cual sólo se muestran las dependencias de control y algunas dependencias de datos a sincronizar se observan 3 ramas de ejecución: $V_1 = \{S_1, S_2, S_3, S_4, S_{10}\}$, $V_2 = \{S_1, S_2, S_5, S_6, S_9, S_{10}\}$ y $V_2 = \{S_1, S_2, S_5, S_7, S_8, S_9, S_{10}\}$. Si se considera, por ejemplo, la dependencia d_{34}, según la regla (a) debe insertarse un signal después de S_3 y

según la regla (b) debe insertarse un signal antes de la ejecución de S_5 (por estar en otra rama de ejecución y por ser la primera sentencia en orden léxico con dependencia de control de la misma sentencia S_2 que decide la ejecución de S_3). En la figura 4.12 se corresponden con las primitivas signal encerradas en un círculo.

Si se considera la dependencia d_{78}, según la regla (a) debe insertarse un signal después de S_7 y según la regla (b) deben insertarse signáis antes de S_6, y antes de S_3 (por estar en una rama diferente y por ser la primera sentencia en orden léxico que depende de control de la misma sentencia S_2 que por propiedad transitiva de este tipo de dependencias también depende S_7). En la figura 4.12 se corresponden con las primitivas signal encerradas en un cuadrado).

Primitiva Wait

Por otro lado, para cualquier dependencia d_{ij} a sincronizar debe realizarse:

(c) **wait** antes de ejecutar S_j en cualquier rama V_m que la contenga, es decir,

$$\forall V_m \mid S_j \in V_m;$$

(d) **wait** al ejecutar cualquier rama V_n que no contenga a S_j, es decir,

$$\forall V_n \mid S_j \notin V_n ;$$

después de la ejecución de la última sentencia en orden léxico $S_k \in V_n$ que cumpla

$$\exists S_c \in (V_m \cap V_n) \mid (S_c \, \delta^c \, S_k \wedge S_c \, \delta^c \, S_j \wedge S_k \notin V_m)$$

En el ejemplo de la figura 4.12, si se considera la dependencia d_{34}, según la regla (c) debe insertarse un wait antes de S_4 y por la regla (d) debe insertarse un wait después de S_9 (por estar en otras ramas de ejecución y ser la última sentencia en orden léxico que depende de control de la misma sentencia S_2 que decide la ejecución de S_4). En la figura 4.12 se corresponden con las primitivas **wait** encerradas en un círculo.

Si se considera la dependencia d_{75}, según la regla (c) debe insertarse un wait antes de la ejecución de S_5 y por la regla (d) debe insertarse un wait después de S_4. En la figura se corresponden con las primitivas **wait** encerradas en un triángulo.

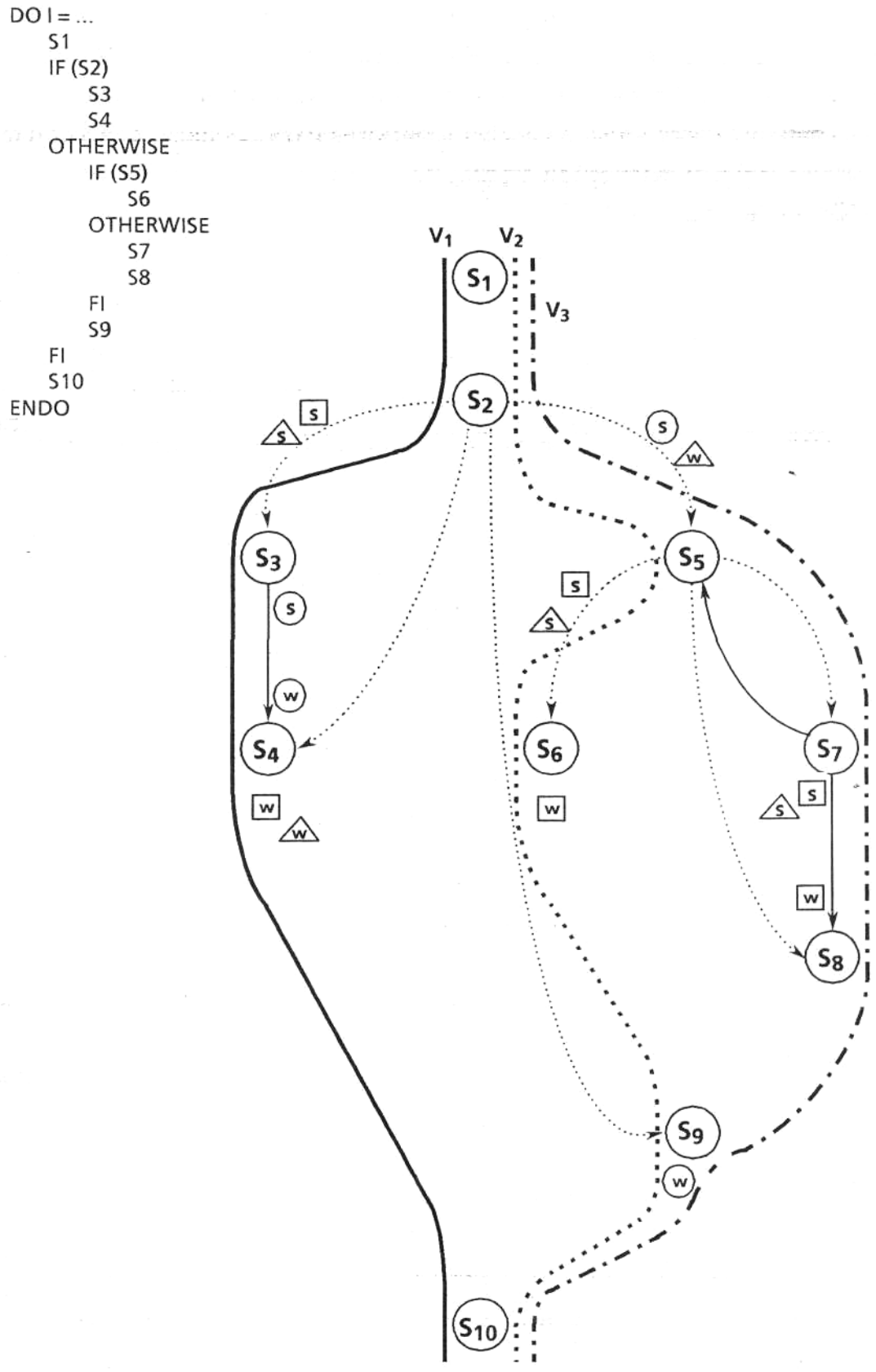


Figura 4.12: Puntos de inserción de las primitivas de sincronización en presencia de condicionales.

La figura 4.13 muestra el bucle secuencial cuyo grafo de dependencias se muestra en la figura 4.11(a) y el código paralelo generado por GTS. En el se observan las primitivas de sincronización insertadas en los puntos correspondientes siguiendo-las reglas-anteriores para las dependencias d_{42} y d_{51}. Estas primitivas actúan sobre los semáforos evaluados según las expresiones del apartado 4.2.2.

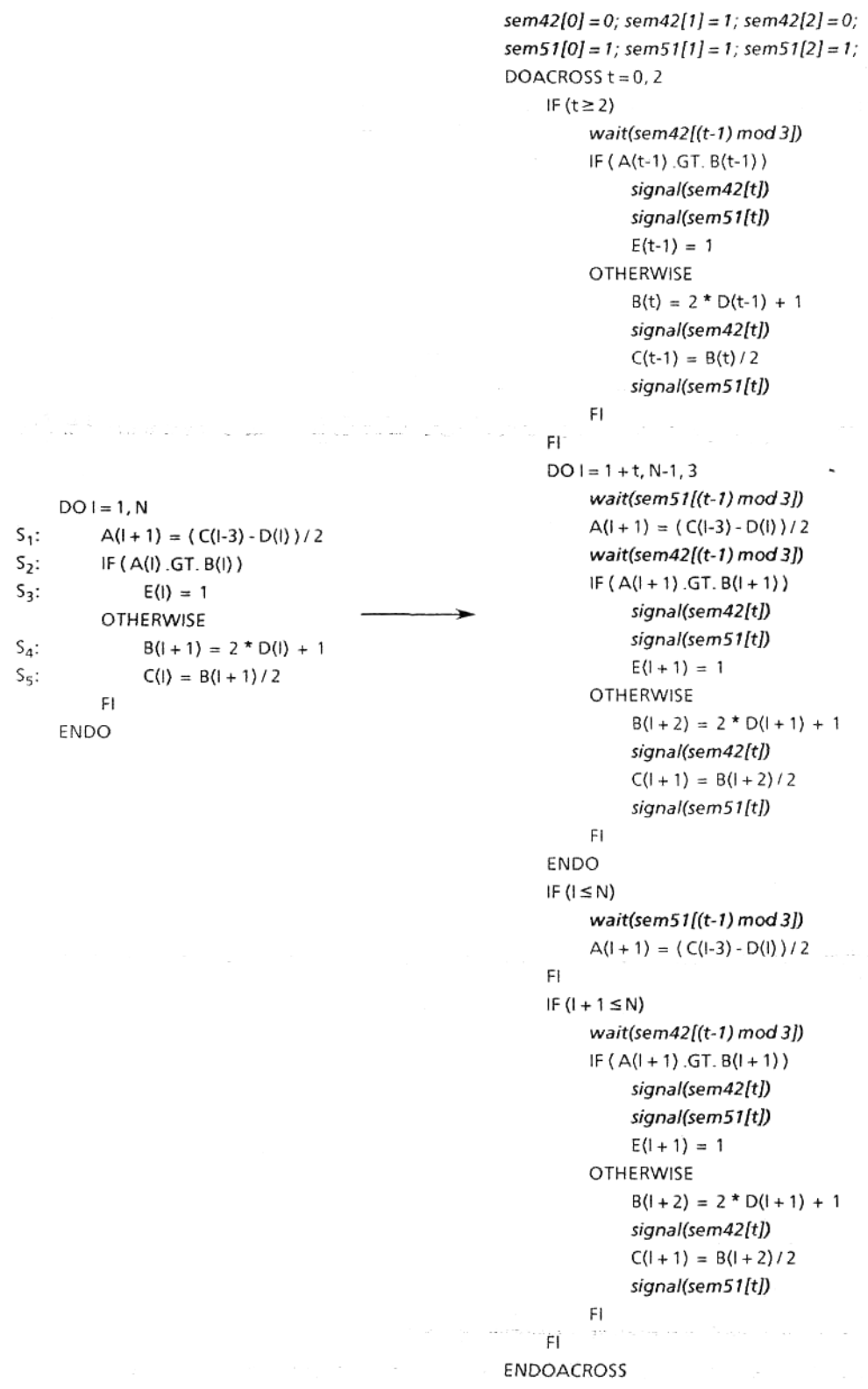


Figura 4.13: Bucle secuencial cuyo grafo se muestra en la figura 4.11(a) y código paralelo generado por GTS.